UNIVERSIDADE FEDERAL DE SANTA CATARINA
CENTRO TECNOLÓGICO
PROGRAMA DE PÓS-GRADUAÇÃO EM
ENGENHARIA DE PRODUÇÃO

FUNDAMENTOS CONCEITUAIS PARA A CONSTRUÇÃO DE SISTEMAS OPERACIONAIS BASEADOS EM CONHECIMENTO

Mauro Marcelo Mattos

Roberto Pacheco, Dr.
Orientador

Florianópolis (SC), novembro de 2003.

Universidade Federal de Santa Catarina
Programa de Pós-Graduação em Engenharia de Produção

Mauro Marcelo Mattos

FUNDAMENTOS CONCEITUAIS PARA A CONSTRUÇÃO DE SISTEMAS OPERACIONAIS BASEADOS EM CONHECIMENTO

Tese de Doutorado

Florianópolis
2003

MAURO MARCELO MATTOS

FUNDAMENTOS CONCEITUAIS PARA A CONSTRUÇÃO DE SISTEMAS OPERACIONAIS BASEADOS EM CONHECIMENTO

**Tese apresentada ao
Programa de Pós-Graduação em
Engenharia de Produção da
Universidade Federal de Santa Catarina como requisito
parcial para a obtenção
do título de Doutor em
Engenharia de Produção.**

Orientador: Prof. Roberto Carlos dos Santos Pacheco, Dr.

**Florianópolis
2003**

Ficha Catalográfica elaborada pela

Biblioteca da FURB

Mattos, Mauro Marcelo
M444f Fundamentos conceituais para a construção de sistemas operacionais baseados em conhecimento / Mauro Marcelo Mattos. – 2003.
372p. : il.
Orientador: Roberto Carlos dos Santos Pacheco.
Tese (doutorado) – Universidade Federal de Santa Catarina, Programa de Pós-Graduação em Engenharia de Produção e Sistemas.

1. Sistemas operacionais (Computadores). I. Pacheco, Roberto Carlos dos Santos. II. Universidade Federal de Santa Catarina. Programa de Pós-Graduação em Engenharia de Produção e Sistemas. III. Título.

CDD 005.43

Mauro Marcelo Mattos

FUNDAMENTOS CONCEITUAIS PARA A CONSTRUÇÃO DE SISTEMAS OPERACIONAIS BASEADOS EM CONHECIMENTO

Esta tese foi julgada e aprovada para a obtenção do título de **Doutor em Engenharia de Produção** no **Programa de Pós-Graduação em Engenharia de Produção** da Universidade Federal de Santa Catarina

Florianópolis, 14 de novembro de 2003.

Prof. Edson Pacheco Paladini, Dr.
Coordenador do Curso

BANCA EXAMINADORA

Prof. Luiz Natal Rossi,Dr
Universidade de São Paulo
Examinador externo

Prof. Roberto Pacheco,Dr.
Universidade Federal de Santa Catarina
Orientador

Prof. Marco Antônio Barbosa Cândido, Dr.
Pontifícia Universidade Católica do Paraná
Examinador externo

Prof. Luiz Fernando Friedrich,Dr.
Universidade Federal de Santa Catarina
Moderador

Profª. Elizabeth Sueli Specialski,Dra.
Universidade Federal de Santa Catarina
Membro

Prof. Rômulo Silva de Oliveira,Dr.
Universidade Federal de Santa Catarina
Membro

Prof. João Bosco da Mota Alves,Dr.
Universidade Federal de Santa Catarina
Membro

À minha adorada esposa Mariene,
companheira e grande incentivadora,
presença indispensável em minha vida.

AGRADECIMENTOS

Os meus sinceros agradecimentos e todas as pessoas e instituições, cuja ajuda, direta ou indireta, tornaram possível a realização deste trabalho.

Aos professores Elizabeth Sueli Specialski,Dra., Rômulo Silva de Oliveira,Dr., João Bosco da Mota Alves, Dr., e Roberto Carlos dos Santos Pacheco,Dr., pelas considerações e contribuições no exame de qualificação e na banca examinadora desta tese. Aos professores Luiz Fernando Friedrich,Dr., Luiz Natal Rossi,Dr. E, Marco Antônio Barbosa pelas considerações e contribuições realizadas na banca examinadora desta tese.

Ao Prof. Bernard Ziegler, PhD., pela atenção e pelas importantes considerações que contribuiram para a concepção do modelo proposto.

À Universidade Federal de Santa Catarina – UFSC, aos professores do Curso de Pós-graduação em Engenharia de Produção e Sistemas – PPGEP, pelo apoio, incentivo e colaboração durante a fase do pós-graduação.

À Universidade Regional de Blumenau – FURB pelo apoio logístico e financeiro. Aos amigos e colegas do Departamento de Sistemas e Computação que apoiaram o projeto. À CAPES/PICDT pelo apoio financeiro.

Aos amigos, Sérgio e Daniela Castro, Cláudia Holetz, Rômulo,Simone e Augusto, Roland e Luci Dagnoni, Jorge Sampaio Farias, Miriam e Sérgio Volpi e Marluza Mattos, Diala e Fernando e, outros tantos amigos, pela presença e constante apoio – fator importante em projetos desta envergadura.

Ao Professor e Orientador Roberto Carlos dos Santos Pacheco,Dr., pelo grande incentivo ao desenvolvimento deste projeto, o qual foi viabilizado pela sua filosofia de trabalho. Muito obrigado!

Ao amigo e colega Jomi Fred Hübner,Dr., pela paciência em ouvir as minhas idéias e pelas considerações que determinaram o fechamento do modelo deste trabalho. Muito obrigado!

Ao grande amigo Rômulo Silva de Oliveira,Dr., cuja disponibilidade, incentivo, caráter ético e muita paciência, contribuiram fundamentalmente para a realização deste trabalho. O meu muito obrigado!

Ao grande Prof.Pedro Bertolino, pelas inestimáveis discussões sobre a elaboração do conceito de conhecimento em um sistema operacional baseado em conhecimento. Sua disponibilidade, visão de futuro e caráter científico foram determinantes na realização deste trabalho. Muito Obrigado!

Ao amigo Juarez Estrázulas, meu sogro, pela revisão do português que tornou este texto legível. À amiga Inar Araújo, minha sogra, pela acolhida, torcida e pelas orações. Voces foram muito importantes durante todo o processo. Muito obrigado.

À Mariene, pelo carinho, amor, amizade, paciência e companheirismo nos momentos difíceis, sem os quais seria difícil concluir este trabalho. Muito obrigado!

Um agradecimento especial vai para a minha avó Isabel (in memorian), que desejava que eu fosse "doutor", ao meu pai Alípio (in memórian), que desejava que eu fosse "engenheiro"e, a minha mãe Marília, que desejava que eu fosse "professor". Hoje eu sou Prof.Dr.Eng., graças aos ensinamentos de voces. Muito obrigado!

"Do ponto de vista da física básica, os fenômenos mais interessantes estão, sem dúvida, nos novos lugares, os lugares onde as regras não funcionam – não os lugares onde funcionam! É assim que descobrimos novas regras!"
Richard P. Feynman.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE REDUÇÕES

ACM - Association of Computer Machinery
ADA - Active Document Architecture
AFIPS - American Federation of Information Processing Societies
AOP - Aspect Oriented Programming
API - Application Program Interface
ARTIS - Architecture for Real-time Intelligent Systems
ASIC - Application Specific Integrated Circuit
AUP - Active Universal Plans
BIOS - Basic Input-Output System
BNF - Backus Naur Form
CASE - Computer Aided Software Engineering
CFD - Contexto físico do dispositivo
CFM - Contexto físico do mundo
CLD - Contexto lógico do dispositivo
CLM - Contexto lógico do mundo
COM - Component Object Model
COP - Component-based Operating System Proxy
CORBA - Common Object Request Brocker
DARPA - Defense Advanced Research Projects
DMA - Direct Memory Access
E/S - Entrada e Saída
FAT - File Allocation Table
FBI - Federal Bureau of Investigation
FCM - Fuzzy Cognitive Maps
FLIPS - Fuzzy Logic Inferences per Second
GNU - General Public License
GPS - Global Positioning System
GUI - Graphical User Interface
HAL - Hardware Abstraction Layer
HCI - Human-Computer-Interaction
HPFS - High Performance File System
HTML - Hypertext Markup Language
IA - Inteligência Artificial
IBM - International Business Machines
IDS - Intrusion Detection System
IEEE - Institute of Electrical and Electronics Engineers
IIDS - Intelligent Intrusion Detection System
IOCS - Input/Output Control System

IPC - Inter-Process Communication
IVR - Interactive Voice Response
JVM - Java Virtual Machine
LRU - Least Recently Used
MIPS - Million Instructions per Second
MIT - Massachusetts Institute of Technology
MM - Modelo de Mundo
MMU - Memory Management Unit
MUD - Multiple User Domain
NCR - National Cash Register Company (adquirida pela AT&T em 1991)
NSA - National Security Agency
NC - Network Computer
NTFS - NT File System (ou New Technology File System)
OLE - Object Linking and Embedding
ORB - Object Request Broker
PC - Personal Computer
PDF - Portable Document Format
PKI - Public Key Infrastructure
PGP - Pretty Good Privacy
POSIX - Portable Operating System Interface
PPFS - Portable Parallel File System
QoP - Quality of Protection
RBC - Raciocínio Baseado em Casos
RISC - Reduced Instruction Set Computer
RMI - Remote Method Invocation
RPC - Remote Procedure Call
SE - Sistema Especialista
SMA - Sistemas Multiagentes
SOBC - Sistema Operacional Baseado em Conhecimento
SOM - System Object Model
TIC - Tecnologias de Informação e Comunicação
TLB - Translation Look-aside Buffer
UML - Unified Modeling Language
VFS - Virtual File System
VMM - Virtual Machine Monitor
VPN - Virtual Private Network
WDM - Windows Driver Model
WMI - Windows Management and Instrumentation
XML - Extended Markup Language

RESUMO

MATTOS,Mauro M. Fundamentos conceituais para a construção de sistemas operacionais baseados em conhecimento. 2003. 372 páginas. Tese (Doutorado em Engenharia de Produção) – Programa de Pós-Graduação em Engenharia de Produção, UFSC, Florianópolis.

Um sistema inteligente deve ser capaz de adquirir autonomamente as informações necessárias, através da percepção, e executar as ações apropriadas. Para atingir a esta meta, sistemas artificiais devem acessar diretamente o ambiente que os cerca. O paradigma atual de sistemas operacionais suporta a noção de abstração de hardware, onde cada aplicação supõe possuir seu próprio processador e demais recursos. Esta situação e o fato de que em geral os sistemas operacionais tradicionais são baseados em (i) multitarefa, contribuem para a permanência de problemas identificados há mais de 30 anos. Estes problemas envolvem não só aspectos de segurança e usabilidade como também questões de privacidade das informações. Alem disso, há outros dois conceitos que contribuem para piorar a situação: (ii) o conceito de operador e (iii) o conceito de programa. A função de operador foi necessária nos primórdios da computação, quando os computadores eram enormes e difíceis de usar. O conceito de programa implica virtualmente na extensão das "mãos dos desenvolvedores" para dentro da máquina dos usuários.

Este trabalho introduz uma definição para a noção de sistema operacional baseado em conhecimento, estabelece a possibilidade concreta dessa definição a partir da apresentação de uma base conceitual que sustenta a definição apresentada e fornece indicações sobre a sua necessidade – isto é, dá-lhe um conteúdo objetivo e mostra o interesse e a utilidade que esta nova concepção pode ter para a área de Sistemas Operacionais em particular e para a área de Computação em geral.

Especificamente, realiza-se uma particular leitura sobre o paradigma atual de concepção de sistemas operacionais e confronta-se este modelo com as demandas de computação ubíqua, teletrabalho e comércio eletrônico, resultando numa evidente disparidade entre os requisitos dos projetos de sistemas operacionais e os requisitos dos projetos das áreas elencadas.

Por outro lado, uma mudança de atitude face aos objetivos propostos é também requerida. A definição dada supõe que se reconheça a autonomia operacional dos sistemas. E isto implica em abandonar, ou pelo menos colocar em segundo plano, o que comumente é chamado de artificialismo – a busca de imitação do comportamento inteligente de seres humanos ou animais - e adotar o ponto de vista denominado naturalismo – a consideração de inteligência de máquina como fenômeno natural nas máquinas, digno de ser estudado em si próprio.

O trabalho apresenta os resultados da reflexão através da qual se tentou realizar os objetivos.

Palavras-chave: Sistemas operacionais, sistemas operacionais baseados em conhecimento, modelo de kernel endógeno.

ABSTRACT

MATTOS,Mauro M. Fundamentos conceituais para a construção de sistemas operacionais baseados em conhecimento. 2003. 372 páginas. Tese (Doutorado em Engenharia de Produção) – Programa de Pós-Graduação em Engenharia de Produção, UFSC, Florianópolis.

An intelligent system should be able to autonomously acquire its required information through perception and carry out the contemplated actions. To achieve this goal, artificial systems must have direct access to their surroundings. The paradigm over which we build operating systems today is based on an hardware abstraction where each application is supposed to possess its own processor (and other resources). This situation and the fact that in general all commercial operating systems are based on (i) multitasking, contribute to enforce problems identified since 30 years ago. Those problems range from security to usability, but also include aspects related to privacy. Besides that, there are two other concepts that contribute to make things worse: (ii) the operator concept and (iii) the program concept. The operator function was necessary in the first moments of computing area since the computers were big and difficult to use. Operators at that time were responsible for turning on/off the machine, starting programs, collecting reports, restarting new programs and so on. The program concept can be thought as the programmer's hands virtually inside the user's machines.

This research introduces the definition for the notion of a knowledge-based operating system (KBOS), establishes the real possibility of this definition based on a conceptual basis that supports the definition and, presents some indications about its needs – that is, gives an objective content and shows the interest and usefulness that this new conception can be to operating systems in particular and computing in general. A particular reading is made about the current operating system building paradigm and this model was compared with new demands in ubiquitous computing, teleworking and e-commerce areas resulting in a disparity between operating system's projects and the other projects.

On the other hand, a change in attitude is required in face of the proposed objectives. The presented definition supposes the recognition of the operational autonomy of the KBOS. Thus, this abandons or at least puts in second plan what is called artificialism – the search for imitation of animals or of the human beings' intelligent behavior and to adopt the point of view denominated naturalism – the consideration of machine intelligence as natural phenomenon in the machines worthy of being studied in itself. This research presents the results of the reflection through which the objectives were accomplished.

Keywords: Operating systems, knowledge-based operating systems, endogenous kernel model.

1 INTRODUÇÃO

Este capítulo apresenta a pesquisa. Com este propósito, inicialmente contextualiza-se o tema e define-se o problema que norteia o desenvolvimento da tese. Os objetivos gerais e específicos são apresentados a seguir, juntamente com as justificativas teóricas e práticas. O capítulo é finalizado com a descrição da estrutura da tese.

1.1 Contextualização do tema e definição do problema de pesquisa

> *"Uma análise da eficiência econômica e da viabilidade dos produtos e serviços de informação nos orienta a uma reflexão de apreciação na manifestação do fenômeno da informação. A produção ou geração de conhecimento (no indivíduo, seu grupo, sua instituição ou a sociedade) ocorre em uma articulação mais ampla, mediada por uma função de passagem, a que chamaremos de função de transferência da informação. A assimilação da informação é a finalização de um processo de aceitação da informação, o qual transcende a disponibilidade, o acesso e o uso da mesma. A assimilação da informação faz realizar o fenômeno do conhecimento no indivíduo (receptor) e em sua ambiência. Este é o destino final da informação: criar conhecimento modificador e inovador no indivíduo e no seu contexto. Conhecimento, que o referencie tanto com o seu mundo de convivência, quanto com um melhor estágio de desenvolvimento" Barreto (1999).*

O impacto e a evolução das tecnologias de informação nas últimas décadas têm apresentado sucessivos desafios, tanto aos indivíduos em relação à sociedade, como às empresas em relação ao mercado globalizado.

Segundo Elhajji (1999),

> *"Não há como negar que a conjugação da dinâmica da globalização ao seu correlato tecno-organizacional, cristalizado no processo de convergência dos meios de comunicação, é portadora de uma profunda força transformadora de todas as condições existenciais da vida contemporânea, desde nossas estruturas sociais, nossos modos de produção e de representação política, até as regras de convivilidade, o sentido de cultura ou ainda o entretenimento. Na verdade, essas mudanças estruturais já estão afetando o conjunto de nosso aparato social material e simbólico, tanto na maneira de organizar nossos lares e nossos círculos afetivos, como nos modos de nos relacionarmos com a comunidade".*

Neste sentido complementa Aun (1999) que,

> *"As transformações do mercado, em nível mundial, atingem as economias dos Estados pelos processos de privatizações e da desregulamentação do mercado, limitando a participação dos Estados ao fornecimento de tradicionais bens públicos e à tarefa de regulamentar. Tais processos atingem o campo informacional por meio das Tecnologias de Informação e Comunicação (TIC), cuja valorização, nas últimas décadas, atinge o seu ápice com o surgimento das infovias de informação, ou redes, com destaque todo especial para a Internet".*

Senra (1999) complementa que,

> *"A informação sempre foi importante, promovendo, em diferentes tempos e espaços, expressivas aberturas de mundo. Entretanto, no limiar do terceiro milênio, em que o marcante movimento de globalização agita, para o bem e/ou para o mal, o nosso viver, sua importância se apresenta ainda mais intensa e potente. De fato, em um mundo que concorre, a informação que engendra conhecimento se faz o recurso básico da produção, levando a uma intensa renovação organizacional".*

Neste contexto, a dependência e demanda crescentes da sociedade, em relação às Tecnologias de Informação e Comunicação, têm ressaltado uma série de problemas relacionados ao processo de desenvolvimento de software – um dos componentes fundamentais na concepção das TICs – quais sejam: alto custo, alta complexidade, dificuldade de manutenção e uma disparidade entre as necessidades dos usuários e o produto desenvolvido.

Segundo Cordeiro (2000) apud Bona (2002,pág 14), "empresas de software em todo o mundo empregam perto de 7 milhões de técnicos e geram anualmente uma receita de mais de 600 bilhões de dólares, com taxa de crescimento de mais de 25% nos últimos três anos". Contudo, segundo Semeghini (2001) apud Bona (2002,pág 16) "no Brasil, cerca de 87% das empresas de desenvolvimento de software são de pequeno e médio portes. Isto implica que existem limitações de recursos a serem aplicados em treinamento e consultorias voltadas à implantação de qualidade de software e de um processo que garanta resultados adequados ".

Segundo Tkach e Puttick (1994),

> *"A medida em que a capacidade computacional e as facilidades de acesso a estes recursos computacionais têm aumentado com os avanços tecnológicos, a complexidade dos sistemas a serem desenvolvidos tem aumentado da mesma forma. Mais e mais usuários são atendidos pelos sistemas computacionais e cientes destas facilidades disponíveis, estes usuários estão demandando cada vez mais novas facilidades".*

Além da dificuldade em atender a estas novas demandas, o *backlog* de manutenção também é um aspecto crítico a ser considerado. Várias das técnicas para análise, projeto e documentação de sistemas utilizadas para implementar sistemas que atendam àquelas demandas, não são flexíveis o suficiente para viabilizar um rápido atendimento das mesmas. Esta situação é comumente referida na literatura como: a crise do software. (TKACH e PUTTICK, 1994).

> *"Esta crise demanda para os departamentos de gerenciamento de sistemas de informação soluções que aumentem a produtividade em pelo menos uma ordem de magnitude. Não há dúvidas que a mudança em algum dos elementos - seja ele uma ferramenta, uma habilidade (skill), uma linguagem de programação ou um paradigma – poderiam produzir tal aumento de produtividade. Tkach e Puttick(1994).*

A referida crise tem várias facetas e a afirmação a seguir caracteriza bem a complexidade do problema:

> *"A metáfora do computador transparente descreve um dos principais objetivos da engenharia de software contemporâneo – ramo da engenharia da informação que se preocupa com o desenvolvimento dos programas complexos (software) necessários para converterem um montículo inerte de dispositivos (hardware), num instrumento poderoso tão fácil de ser usado como são o lápis e o papel. Quem já esperou um dia ou mais por um programa de computação convencional que nem sequer chegou a ser processado apenas porque uma vírgula fora mal colocada, poderá testemunhar que ainda não chegou o dia em que a transparência computacional instantânea estará a serviço de todos." Anthony G. Oettinger (1966).*

O problema destacado por Oettinger infelizmente ainda hoje, 37 anos depois, é uma realidade nos ambientes de desenvolvimento de software – um simples ponto, ou vírgula, ou ponto e vírgula, indevidamente posicionado ou ausente, pode ser o fator determinante na produção de um software que funciona corretamente ou não. E são fatores como este que fazem com que a crise do software ainda hoje continue presente nas organizações. Corroborando esta afirmação, é apresentado em Rational (2001) apud Ferreira (2002) um estudo realizado em 1995 pelo Standish Group, abrangendo mais de 352 empresas e envolvendo mais de 8.000 projetos de software, o qual apresentou os seguintes resultados:

- 31% dos projetos de software são cancelados antes de serem concluídos;
- 53% dos projetos ultrapassam os custos estimados e,
- Em pequenas empresas, apenas 16% dos projetos de software são concluídos dentro do tempo e do orçamento previstos. Nas grandes empresas, este número cai para 9%.

Segundo Ferreira (2002), para a solução desta chamada crise do software, foram definidos complexos conjuntos de princípios, técnicas, métodos e ferramentas que suportam atividades de desenvolvimento de software.

No contexto das empresas envolvidas com o desenvolvimento de software, as diversas estruturas propostas resultam em dificuldades. A ausência de simplicidade dos modelos propostos para o software, forma barreiras que inibem a sua aplicação (BACH, 1994 apud FERREIRA,2002). Torna-se difícil a identificação das estruturas mais importantes e de como se relacionam com as demais propostas. Também existem dúvidas em relação

ao real progresso a ser alcançado na implementação das referidas propostas (SHEARD,2001 apud FERREIRA ,2002).

Ainda segundo Ferreira (2002),

> *"No âmbito das jovens empresas em fase de formação e crescimento no mercado de software, estas dificuldades são agravadas. Caracterizadas por processos informais, grande escassez de recursos disponíveis, sobrecarga de trabalho e um imaturo conhecimento de software, estas empresas contribuem para as estatísticas negativas da crise de software e do mercado de novos empreendimentos".*

A partir destas considerações cabe destacar: **um sistema operacional é um produto de software** – portanto, está sujeito aos mesmos problemas. Um exemplo disto é a afirmação recente de Steve Ballmer (CEO da Microsoft) (WATERS,2003):

> *"O sistema operacional Windows 2003 Server é seguro por definição (secure by design) e nós investimos **U$200 milhões** neste projeto. Ele é seguro por definição com 60% menos superfície de ataque em comparação ao NT4 service pack 3".*

A afirmação de Nagle (1994) também caracteriza estes aspectos:

> *"Muitos projetos recentes tais como Windows NT, Mach 3.0, Chorus, System V e Sprite são projetados para minimizar o custo do porte para outras plataformas suportando múltiplas APIs. Contudo, as facilidades adicionais destes projetos possuem um custo associado: são mais lentos que os sistemas tradicionais".*

Desde os primórdios da computação, a comunidade de Sistemas Operacionais estuda e implementa estratégias e soluções em software, conjugadas com soluções em hardware, que visam permitir a utilização do hardware da forma mais adequada e eficiente possível.

Caracterizada como uma área que desenvolve soluções de baixo nível, devido à íntima relação com as características e funcionalidades do hardware sobre o qual estas soluções são construídas, experimentou um período de evidência até o final da década de 80. Este fato pode ser constatado pelo grande número de publicações científicas especializadas na área e corroboradas, inclusive, sob a chancela de organizações como ACM (*Association of Computer Machinery*) e IEEE (*Institute of Electrical and Electronic Engineers*).

Precursora da área de Engenharia de Software, a comunidade de Sistemas Operacionais experimentou todo tipo de dificuldades no que tange ao desenvolvimento de suas soluções de software, visto que, para viabilizá-las, teve que propor e testar, na prática, várias técnicas de gerência de projetos, especificação de software, construção de

compiladores, depuradores, otimizadores de código e até mesmo linguagens de programação.

Restrita a grupos que se foram especializando dentro das instalações fabricantes de hardware, os cientistas de software começaram a entrar em evidência a partir do momento em que a IBM decidiu comercializar tanto hardware como software. Segundo Deitel (1990), este é o marco que estabeleceu o início da área de Engenharia de Software. Isto porque problemas que eram antes restritos aos fabricantes e às poucas instalações que possuíam computadores extrapolaram com o surgimento de empresas denominadas de *software houses*.

Seguindo este processo evolutivo, a área de Engenharia de Software começou a identificar segmentos que necessitavam aprofundamento de pesquisa e vários cientistas que anteriormente trabalhavam com Sistemas Operacionais, paulatinamente começaram a enquadrar suas atividades de pesquisa e desenvolvimento em segmentos da área de Engenharia de Software.

O aspecto filosófico que está por trás da evolução da computação em geral é o de liberar os programadores de detalhes desnecessários. A idéia tem sido de que, quanto maior o nível de abstração permitido, mais complexos são os problemas que eles podem resolver. Isto porque havia no passado uma realidade impondo sérias restrições na forma de utilização das máquinas – custo de hardware e software muito elevados e requisitos de qualidade, confiabilidade e performance incomparavelmente inferiores aos de hoje.

Segundo John McCarty (1966),

> *"Em 1966 o governo americano gastava metade do orçamento de U$844 Mi com software. Além disso, havia aproximadamente 35.200 computadores, 200.000 programadores e 2100 grandes sistemas custando U$ 1Milhão cada (AFIPS - American Federation of Information Processing Societies)".*

Segundo Moeller (1991), alguns dos maiores avanços na história da computação convencional (Figura 1) têm sido motivados por esta filosofia, conforme se pode observar nos exemplos a seguir:

- **Linguagens de montagem (*assembly*):** para evitar que um programa fosse codificado em zeros e uns, foram desenvolvidos os montadores e, a partir daí, os programadores passaram a expressar suas idéias em termos de instruções mnemônicas de mais alto nível;
- **Linguagens de alto nível:** para evitar preocupações com detalhes de hardware e instruções de baixo nível, foram desenvolvidas as linguagens de programação contendo estruturas

lógicas mais abstratas. Isto permitiu que os programadores pudessem expressar suas intenções sem ter que se preocupar com um tipo em particular de hardware e/ou conjunto de instruções;

- **Alocação dinâmica de memória:** para liberar os programadores do planejamento antecipado de suas necessidades de memória, foram desenvolvidas estratégias de alocação dinâmica e disponibilizadas em linguagens de alto nível (ex: Java).
- **Orientação a objetos e aspectos**: para auxiliar o programador a organizar e proteger suas estruturas desenvolveu-se o conceito de objetos, o qual permite manipulações de mais alto nível sobre as mesmas. A técnica de programação orientada a aspectos (AOP) visa solucionar estes problemas através do principio da separação de interesses (*concerns*), no qual são separados o código de um componente do código de um aspecto.

Observa-se que este processo evolutivo tem ocorrido em função de um usuário especializado conhecido como *"o programador"*. Este vem recebendo todas as atenções por décadas, enquanto outro tipo de usuário, conhecido como usuário final, não vem recebendo a atenção devida.

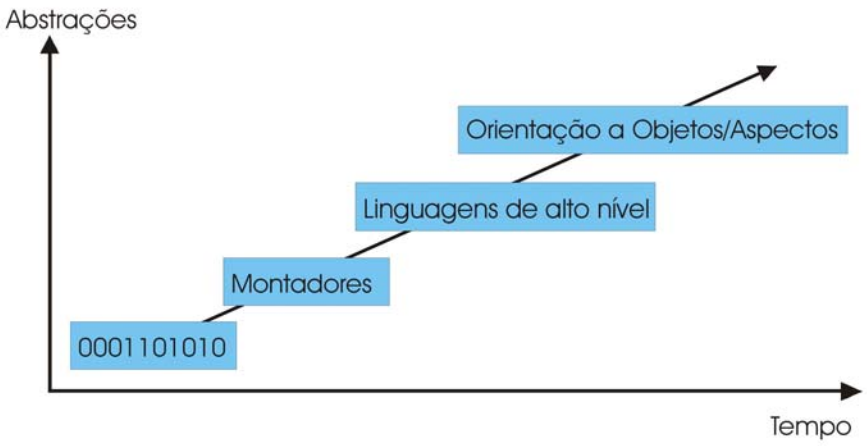


Figura 1 - Evolução das abstrações em favor do programador.

O próprio caminho evolutivo de hardware facilitou o surgimento de desenvolvedores de hardware específicos, os quais importaram desenvolvedores para implementar o que começou a ser denominado de *firmware* (e/ou a camada de *device drivers*[1]) específicos para aquele hardware. À medida que este processo tornou-se mais e mais rápido, em função da explosão do uso de tecnologia de computação na sociedade, mais e mais o conceito de projetistas de sistemas operacionais tornou-se nebuloso.

[1] Camada de software especializada que contém instruções que comandam os componentes controladores dos dispositivos de hardware.

Segundo Cordeiro (2000) apud Bona (2002), inicialmente acumulados na parcela de hardware, os custos de sistemas computacionais deslocaram-se para a parcela de software. Este fenômeno ocorreu em função da estabilidade alcançada pelos projetos de hardware, bem como pelo surgimento de exigências crescentes em complexidade para os sistemas de software. Também contribuíram problemas emergentes nos processos de desenvolvimento e manutenção de software.

Esta migração pode ser constatada a partir da verificação de que a grande maioria dos eventos científicos e das publicações especializadas da área de Sistemas Operacionais, ou acabaram ou foram incorporados como tópicos em eventos ou publicações mais abrangentes.

O mercado experimenta, atualmente, a polarização em basicamente duas linhas de sistemas operacionais[2]: os baseados em **Unix** e os baseados na arquitetura Microsoft. Desapareceram do mercado soluções como **OS/2** e, mais recentemente, o **BeOS**. Hoje as atenções estão voltadas para novas formas de interação com o usuário, soluções envolvendo Internet (como *e-learning, e-commerce* e teletrabalho utilizando tecnologias como CORBA, RMI e Java), computação móvel e computação ubíqua, as quais são concebidas, de uma forma geral, sobre alguma das plataformas comercialmente disponíveis.

A análise da literatura relacionada à identificação de algumas das novas demandas - Capítulo 3 -, bem como à forma de concepção e projeto dos sistemas operacionais (que constituem a base a partir da qual as TICs são tornadas viáveis) – Capítulo 2 - e à forma como a tecnologia de informações é utilizada para superar as limitações dos sistemas operacionais – Capítulo 4 -, indica a inexistência de trabalhos que reconheçam a importância estratégica que uma nova concepção de sistema operacional, baseado em técnicas de IA, Robótica e Engenharia do Conhecimento, pode trazer para a comunidade de Tecnologia de Informação e Comunicação.

---

[2] Estes são os produtos dominantes no mercado de computadores desktop e notebooks, embora existam outras propostas como: **AS-400** da IBM, **Mac OS-X** da Apple, a linha de tempo real como **QNX** e, naturalmente, os sistemas para Mainframe, como o **OS/390** da IBM. Embora não seja o foco deste trabalho discutir os aspectos relativos a estas categorias de sistemas, várias considerações apresentadas são a eles aplicáveis.

1.1.1 Tema de Pesquisa

Segundo Oliveira (2002), a formulação do tema de pesquisa pode ser descritiva ou normativa. O tema desta pesquisa, de caráter descritivo, está centrado nas dificuldades que a área de Sistemas Operacionais tem, historicamente, em atender às novas demandas e exigências que surgem, decorrentes da evolução tecnológica, e à evidente necessidade de propor soluções para amenizar e resolver tais dificuldades. As evidências são apresentadas no capítulo 5.

A linha mestra do trabalho visa a identificação dos fundamentos conceituais e dos requisitos funcionais e não funcionais, bem como a especificação de uma nova estrutura de Sistema Operacional Baseado em Conhecimento (SOBC), de modo a permitir a construção de novas formas de abstrações que possibilitem a superação de problemas clássicos e recorrentes em sistemas computacionais.

1.1.2 Problema de Pesquisa

Buscando uma formulação mais específica, o problema de pesquisa é apresentado indicando de forma mais precisa as dificuldades que se pretende resolver.

Segundo o diretor de tecnologia de processamento de informações da agência DARPA (*Defense Advanced Research Projects*) Ronald J. Brachman:

> *"(...) infelizmente, os melhoramentos constantes na velocidade dos processadores apresenta duas dificuldades: enquanto nos dão a oportunidade de construir grandes e melhores coisas, eles inexoravelmente nos conduzem a fazer coisas grandes e, bem, coisas muito grandes. O tamanho dos sistemas de software está aumentando num espiral virtualmente descontrolado – hoje é comum encontrarem-se aplicações que ocupam 50Mbytes e que possuem 5 milhões de linhas de código – e nós sabemos que este crescimento em complexidade inevitavelmente conduz a sérias e novas vulnerabilidades. Sob a perspectiva de segurança, maior complexidade significa mais possibilidades para um intruso encontrar caminhos para entrar; sob a perspectiva de robustez, mais elementos significam mais caminhos que podem conduzir a erros; sob a perspectiva de manutenção, mais código significa que os custos de manter as coisas rodando crescem continuamente; e, sob a perspectiva de treinamento, significa que mais pessoas precisam gastar mais tempo aprendendo como usar sistemas baseados em computador. E eu ainda não toquei em algo que nos afeta todos os dias – a chamada usabilidade dos nossos sistemas.Tudo isto nos conduz a uma conclusão muito clara:* ***nós necessitamos de alguma coisa dramaticamente diferente****. Nós não podemos meramente aumentar a capacidade e velocidade dos nossos computadores. Nós não podemos somente fazer uma engenharia de software melhor.* ***Nós precisamos mudar nossa perspectiva em sistemas computacionais e mudar a trajetória atual(…)"*** *Brachman(2002).*[3]

[3] Os grifos são nossos.

O cenário apresentado anteriormente deve-se, em parte, à enorme demanda por novas soluções que a sociedade impõe ao mercado e que, portanto, fazem com que as empresas tenham que produzir soluções em software cada vez mais rápido. Este processo conduz às questões de reaproveitamento de esforço (código, conhecimento, recursos financeiros, etc) para "não ser necessário reinventar a roda". Na área da pesquisa também existe muita pressão por prazos e resultados, o que induz a adoção de uma base estável como forma de ganhar tempo e concentrar esforços no alvo dos projetos.

Contudo, há um outro aspecto a ser considerado e que está relacionado à **virtualização do operador** de computador[4]. À medida que ocorrem a evolução tecnológica e a miniaturização dos computadores, várias funções vêm sendo incorporadas ao sistema na forma de comandos e interpretadores de comandos. A figura do operador desapareceu no contexto de computadores *desktops* (*notebooks* e *palmtops*) e surgiu a figura do usuário.

Hoje quem usa um computador tem que saber como operá-lo, ou seja, como preparar os dados, como recuperar arquivos, como configurar programas, onde colocar os arquivos, em que formato armazenar, quando realizar uma cópia de segurança (*backup*), etc. Ou seja, o processo de virtualização do operador transformou o sistema operacional num facilitador de uso (uma espécie de *proxy server*) e o usuário num operador especializado. Esta necessidade de aprender a operar é, por si só, um fator de exclusão social, na medida em que pessoas necessitam ser treinadas a operar "botões lógicos" para serem aproveitadas pelo mercado de trabalho. E como a área de computação evolui muito rapidamente, esta necessidade de (re) treinamento torna-se algo tão corriqueiro que chega a ser óbvia no modelo atual.

Este caráter de virtualização do operador pode ser caracterizado a partir da seguinte afirmação:

> *"Na Engenharia Semiótica, a interface é vista como uma mensagem unidirecional enviada dos projetistas para os usuários. Segundo esta abordagem, a mensagem é um artefato de metacomunicação, já que não apenas os projetistas se comunicam com os usuários, mas a própria interface troca mensagens com os últimos. Neste contexto, a interface é um conjunto de signos que são representações que os projetistas usam para comunicar aos usuários como manipular o sistema para atingir seus objetivos" De Souza (1993).*

[4] Posição ocupada pelo pessoal responsável por efetivamente operar computadores, executando funções como montar discos e fitas, retirar relatórios da(s) impressora(s), disparar a execução de programas e acompanhar o funcionamento do sistema como um todo.

Um outro conceito que influencia a forma de concepção das soluções computacionais hoje é decorrente da introdução do conceito de multiprogramação. Isto porque, até o surgimento deste conceito (como forma de otimização do uso de recursos), os programas eram enfileirados ou agrupados em lotes para serem executados. Cada programa assumia o controle da máquina durante sua execução e somente ao término de um programa é que o próximo da fila iniciava a sua operação.

A **virtualização do hardware,** introduzida pelo conceito de multiprogramação, acabou com o enfileiramento de programas através da divisão do tempo de processador em pequenas fatias denominadas *time-slices* conforme caracterizado pela Figura 2. A decorrência imediata disto foi o surgimento dos problemas de proteção (código e dados) que estão presentes até hoje.

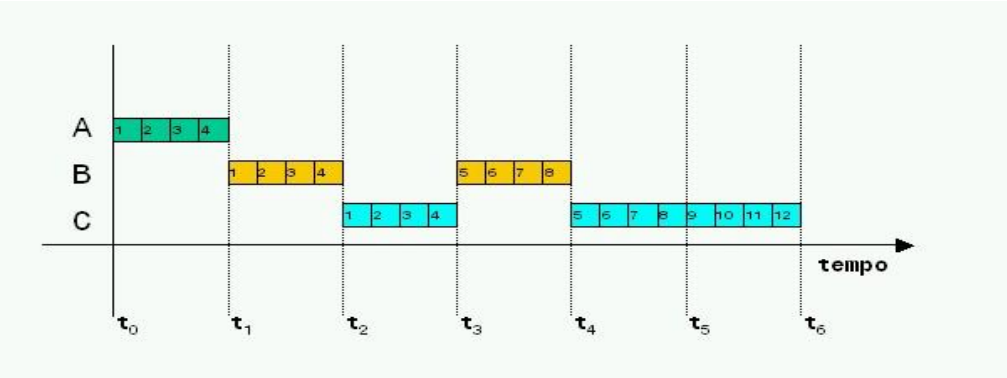


Figura 2 - Virtualização do processador através de multiprogramação

E finalmente, mas não menos importante, é o **conceito de programa**. Um programa é uma abstração muito mais complexa do que aquela considerada nos estudos de Engenharia de Software. Lá são considerados os aspectos funcionais e não funcionais de uma solução computacional para um problema qualquer.

No entanto, a questão é mais complexa. Na realidade, um programa representa a intromissão virtual de uma pessoa[5] (ou grupo de pessoas) dentro da máquina do usuário. Um programa implementa decisões e ações que foram concebidas em contextos diversos, por pessoas com as mais diversas intenções e competências. A Figura 3 caracteriza esta situação demonstrando que a dificuldade de interação é inerente ao próprio modelo. Em verde estão representados os grupos que desenvolvem o nível geralmente denominado

[5] Na figura do programador ou equipe de desenvolvimento.

de: software básico. A seguir aparecem os desenvolvedores do sistema operacional propriamente dito. Acima destes estão os desenvolvedores de software aplicativo. Cada equipe com seus particulares objetivos e requisitos de performance.

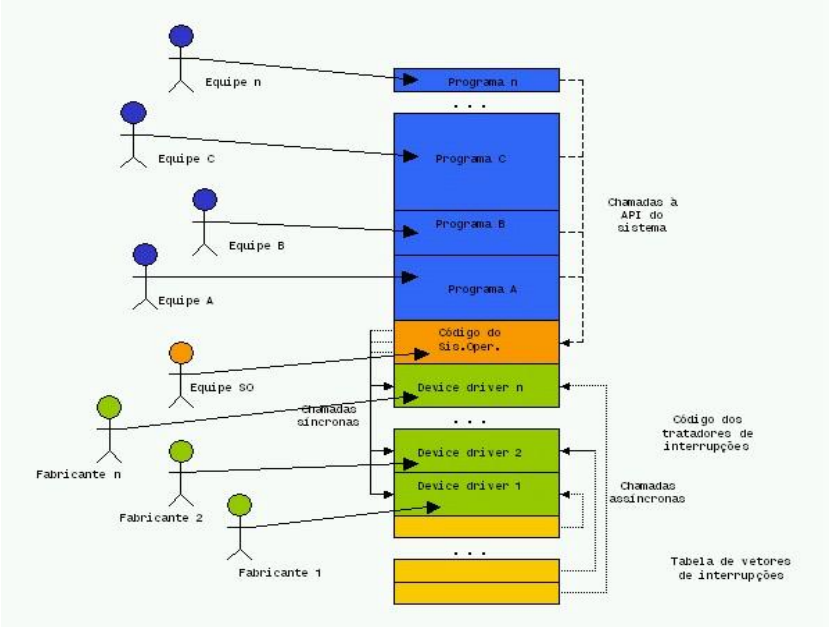


Figura 3 - Intromissão virtual de pessoas na máquina do usuário

O surgimento do conceito de agentes representa uma forma inequívoca desta interpretação. O agente comporta-se como se fosse a incorporação virtual do programador ou da equipe que tem o conhecimento sobre o funcionamento de um programa ou área de aplicação. Ou seja, incorpora o fato de que, na realidade, quem tem o controle da situação é esta entidade externa. Embora seja uma solução interessante, esta metáfora é construída sobre uma base insegura (multiprogramação) onde não são considerados os aspectos de "convivência mútua" entre os "representantes dos vários desenvolvedores externos" [6].

Este aspecto, em particular, colabora de forma importante para agravar as questões de segurança. Isto porque cada programa, individualmente, não sabe o que estão fazendo ou pretendem fazer os demais programas que, em determinado momento, concorrem pelas fatias de tempo do processador. E o sistema operacional também não tem esta informação. Assim, o risco é iminente a cada fatia de tempo.

[6] Sob este aspecto cabe destacar que as primitivas de sincronização, geralmente fornecidas pelos sistemas operacionais, têm a finalidade de sincronizar (e/ou facilitar a comunicação entre) programas, sob o ponto de vista de requisitos funcionais e não funcionais. O conceito de convivência mútua envolve a finalidade e o resultado da ação da execução de programas em um ambiente de multitarefa.

Portanto, o tripé conceitual: **virtualização do operador**, **virtualização do hardware** e **conceito de programa**, caracterizam os fundamentos básicos do atual modelo de construção de sistema operacional, em particular, e de soluções em software, em geral. E, conforme será mostrado neste trabalho, problemas identificados já há quase 3 décadas persistem através das várias gerações de tecnologias desenvolvidas sobre este tripé conceitual.

A preocupação com qualidade em projetos de sistemas operacionais não é recente. Nicol em 1987 afirmava que:

> *"A história recente· destaca o esforço dos projetistas de sistemas em aspectos quantitativos (processadores mais rápidos, maior capacidade de memória, maior eficiência, tempo de resposta mais rápido, etc). Em função disto, menos atenção tem sido dada à qualidade do serviço prestado ao usuário. Como conseqüência, um maior esforço têm sido dedicado a interfaces gráficas de comunicação, facilitando a interação com o usuário".*

Fazendo-se uma analogia entre a situação acima descrita e aquela associada à filosofia de favorecer o programador em detrimento do usuário, pode-se afirmar que, do ponto de vista do usuário final, a atual tecnologia de sistemas operacionais pode ser classificada no mesmo nível evolutivo daquele caracterizado pelo advento das linguagens de alto nível (Figura 4).

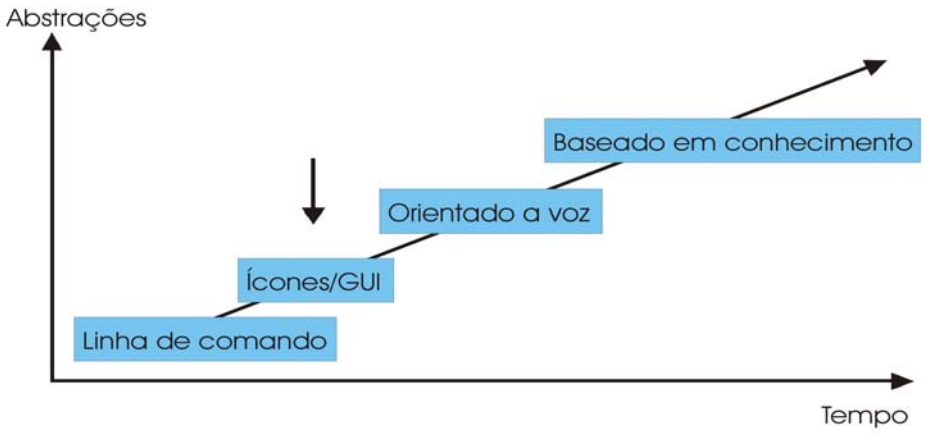


Figura 4 - Estágio atual de evolução dos sistemas operacionais

O usuário já pode usar comandos mais abstratos para operar a máquina, através de ícones e interfaces gráficas, mas ainda tem que se preocupar com o nome de pastas e arquivos e configurações técnicas como url,smtp,dns,etc., as quais muitas vezes fogem do seu escopo de conhecimento e o impedem de ser produtivo em seu ambiente de

trabalho. Para exemplificar esta situação, é apresentada na Figura 5 a tela principal de configuração do browser Internet Explorer o qual é amplamente utilizado atualmente. São sete pastas (algumas contendo sub-pastas) contendo opções de configuração que nem sempre o usuário sabe exatamente qual a sua finalidade.

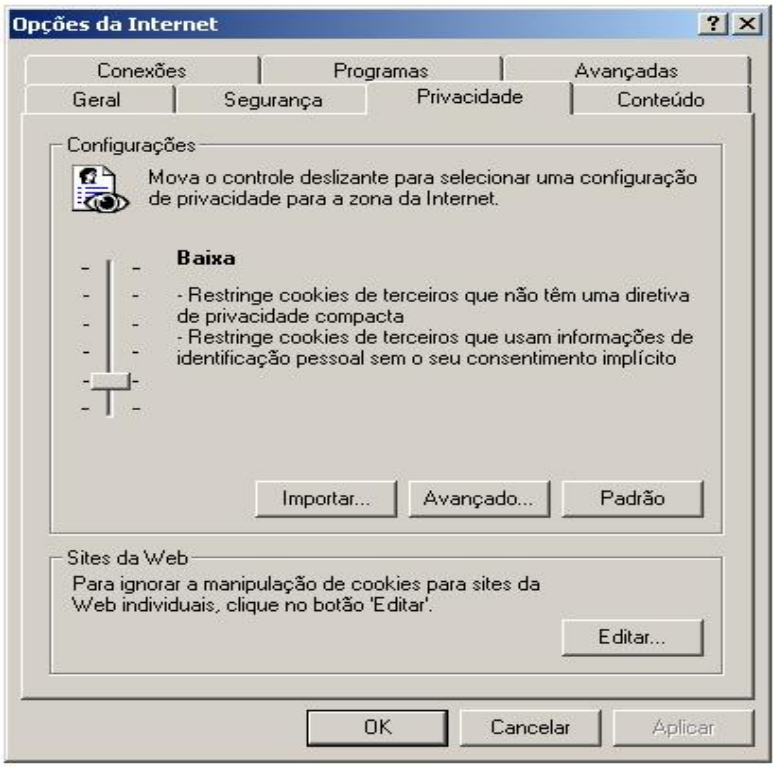


Figura 5 - Exemplo de opções de configuração do browser Internet Explorer

1.1.3 Questões de Pesquisa

Segundo Hoppen, Lapointe e Moreau (1996), a categoria questão de pesquisa corresponde às várias dimensões do estudo. Tem por objetivo permitir a verificação sobre a adequação entre o método utilizado e o problema escolhido.

Observa-se que as várias propostas de sistemas operacionais existentes não enfocam adequadamente a questão do atendimento às demandas crescentes e cada vez maiores por novas funcionalidades. Portanto, esse é um amplo campo a ser explorado. Nele, insere-se a realização deste trabalho, cujo enfoque principal é, exatamente, o de propor os fundamentos de uma nova arquitetura de sistema operacional capaz de atender àquelas demandas.

Sob esta perspectiva, o problema de pesquisa é apresentado na forma interrogativa:

Como construir um novo modelo de sistema operacional baseado em abstrações de conhecimento e comportamento inteligente?

A partir das considerações iniciais apresentadas nas seções anteriores, as seguintes questões devem ser evidenciadas sob a forma interrogativa:

1. Quais as características ao atual modelo de Sistema Operacional?
2. Quais as demandas que deverão surgir no futuro?
3. Como tradicionalmente são atendidas as demandas?
4. Qual é o resultado da utilização do atual modelo de Sistema Operacional?
5. Como superar as deficiências através do emprego de técnicas utilizadas em outras áreas do conhecimento?
6. O que é conhecimento e comportamento inteligente no contexto de um sistema operacional?

1.2 Objetivos

Segundo Ruiz (2002), a explicitação dos objetivos de pesquisa concorre para caracterizar o seu alcance e, portanto, para uma nova forma de delimitação do tema. Os objetivos estabelecidos nesta pesquisa serão apresentados na forma de objetivo geral – forma genérica - e objetivos específicos – forma exata (SILVA NETO,2000,pág.86). Também são identificados alguns pontos críticos e dificuldades para a realização da pesquisa.

1.2.1 Objetivo Geral

Como objetivo geral é proposta a elaboração dos fundamentos conceituais de um novo modelo de sistema operacional baseado em abstração de conhecimento e comportamento inteligente.

1.2.1 Objetivos Específicos

Para um perfeito entendimento do objetivo geral proposto faz-se necessária a definição exata do propósito a que se destina. E, para tanto, deve ser analisado mais detalhadamente o termo: “conceito de mundo”.

Um aspecto de fundamental importância nos projetos de robótica, e que é negligenciado pelos projetistas de sistemas operacionais, é o fato de que, em projetos de robótica (considerando-se robôs autônomos), geralmente há uma função de mapeamento

entre o ambiente e uma representação interna do mesmo, denominada “modelo de mundo". Ou seja, há alguma forma de representação do ambiente onde o robô vai operar. E é com base neste modelo de mundo que as decisões são tomadas.

Embora os sistemas operacionais também utilizem registros (na forma de diretórios especiais **-/proc** no **Linux**; ***registry*** do **Windows**, etc) para descrever seu estado em determinado momento, eles não apresentam um comportamento externo tão inteligente quanto aquele apresentado pelos robôs. Naturalmente, o escopo e a complexidade das aplicações são diferentes. Mas o aspecto filosófico é que deve ser destacado. E é justamente este o ponto de partida do projeto que ora está sendo desenvolvido.

O objetivo geral de pesquisa é tornado operacional na realização dos seguintes objetivos específicos:

1. Estabelecimento de uma base científica para sustentar o conceito de sistema operacional baseado em conhecimento, através de uma revisão bibliográfica dos itens envolvidos (cap 6);
2. Identificação de como o conceito de mundo, adotado em projetos de robótica, pode ser utilizado como base para a construção de uma nova abstração em sistemas operacionais (cap 6);
3. Especificação de uma infra-estrutura que permita a utilização do conceito de mundo como principal abstração sobre a qual as aplicações são desenvolvidas (cap 7);
4. Estabelecimento dos requisitos funcionais que devem ser apresentados por um Sistema Operacional baseado no conceito de mundo (cap 7);
5. Identificação das técnicas utilizadas em outras áreas do conhecimento e que possam ser agregadas de forma a permitir a construção da abstração de comportamento inteligente (cap 6).

1.2.2 Pontos Críticos e Dificuldades

O contexto de desenvolvimento de um novo modelo de sistema operacional implica em um leque muito amplo de assuntos. A multiplicidade de propostas, enfoques de construção, áreas de aplicação, linguagens de desenvolvimento e características de cada enfoque, requerem um extenso levantamento bibliográfico.

A identificação das novas demandas, das tecnologias de informação utilizadas para fazer-lhes frente e dos problemas decorrentes do uso do atual modelo, também são questões importantes.

Contudo, o maior desafio e a maior dificuldade residem no paradoxo que existe em propor um novo modelo, utilizando técnicas, recursos e ambientes computacionais baseados no modelo que está sendo questionado.

1.3 Justificativa

A justificativa teórica e prática, assim como os principais fatores que evidenciam a relevância, o ineditismo e a contribuição desta tese, são elencados a seguir.

Segundo Ahmed (1998), a indústria de microprocessadores está em meio a uma revolução graças a uma série de recursos tais como: reutilização de projetos, melhores práticas em engenharia de lógica digital e eficiência de fabricação. Assim sendo, é possível desenvolver projetos de hardware especializados para atender a um largo espectro de aplicações.

A indústria de software também está vivenciando um processo evolutivo que, apesar de não tão intenso quanto o de hardware, caracteriza-se por: mudanças nos modelos de construção de software, de disponibilidade de recursos de hardware e software de apoio (ferramentas *CASE*[7]), de interfaces de comunicação homem-máquina e de distribuição de recursos de hardware e software (através da facilidade cada vez maior de interligação de equipamentos).

Não obstante este avanço há uma área que tem apresentado pouca evolução perante o quadro que foi apresentado anteriormente – a área de sistemas operacionais. E esta evolução tem ocorrido no sentido de encontrar-se uma solução de compromisso entre a demanda de necessidades cada vez maiores dos usuários e, em contrapartida, a evolução muito rápida do hardware.

É inegável que muitas facilidades tem sido incorporadas na forma de interação entre o usuário e a máquina, desde que os primeiros microcomputadores (Tk-85, Apple, PC-XT – no contexto do Brasil) começaram a surgir, lá pelo início da década de 80. Também se deve considerar que os sistemas produzidos têm proporcionado ganhos de produtividade em relação às gerações anteriores.

Por outro lado, é inegável que o modelo atual tem apresentado deficiências clássicas (conforme será apresentado nos capítulos seguintes). Um dos fatores críticos

[7] Ferramentas para apoio ao projeto, geração de código, testes e depuração.

neste processo permanece sendo o software e, mais particularmente, o sistema operacional. Ele é o responsável imediato por uma grande parcela de contribuição para que o computador ainda seja inacessível a grande parte da sociedade, na medida em que ainda é complicado, instável e varia muito de equipamento para equipamento.

A possibilidade de acesso a informações sob demanda em qualquer local caracteriza uma tendência de pesquisa multidisciplinar cada vez maior, principalmente a partir da disseminação de redes sem fio (*wireless*) e tecnologias de interconexão cada vez mais disponíveis (*hardware* e protocolos). A proposta de espaços inteligentes abre uma nova dimensão, em termos computacionais, comparada àquela aberta com a introdução da tecnologia de redes. Deixou-se de ter equipamentos independentes e passou-se a contemplar o compartilhamento de recursos. Agora se quebra o modelo de interligação de computadores e abre-se uma nova dimensão, permitindo a integração ao ambiente computacional de outros equipamentos que hoje requerem configuração, coordenação e tratamento diferenciados.

Os pesquisadores do laboratório Pablo (PABLO 2001) da Universidade de Illinois, afirmam que a confluência de tecnologias de redes móveis, monitores *head-mounted,* vídeo digital de alta definição e ambientes virtuais imersivos, acenam para uma nova era. É possível prever que, nesse novo cenário, uma poderosa base tecnológica poderá aumentar a capacidade intelectual humana. No futuro, o ambiente de informações deverá apresentar, entre outras características, a mobilidade, a disponibilidade e sensibilidade ao contexto em que operam.

Há várias implicações tanto tecnológicas como sociais nesta visão de ambiente computacional, visto que, de um lado, a proposta envolve alterações em toda a infra-estrutura que conhecemos hoje e, portanto, requer grandes investimentos para alimentá-la; de outro, isola ainda mais sociedades que não dispõem dos recursos financeiros necessários à aquisição e assimilação da tecnologia.

Wang e Garlan (2000) afirmam que a infra-estrutura computacional de hoje não suporta o modelo de computação de forma adequada, tendo em vista que os computadores interagem com os usuários em termos de abstrações de baixo nível: aplicações e dispositivos individuais.

O grande desafio consiste, portanto, em adequar os requisitos identificados nos chamados espaços inteligentes, teletrabalho e comércio eletrônico[8], com aqueles necessários à geração de uma tecnologia que, ao mesmo tempo, dê suporte ao projeto tecnológico e consiga tornar as tecnologias computacionais mais eficientes, amigáveis e adaptáveis, ou seja, possibilitem a reconfiguração do sistema operacional em função de alterações no ambiente, de forma a minimizar a necessidade de intervenção do usuário.

Porém, grande parte do esforço de pesquisa atualmente desenvolvido, adota a abordagem de promover pequenas alterações no modelo atual de sistemas computacionais, partindo do princípio de que este modelo é aparentemente a melhor solução.

Um outro aspecto que justifica esta pesquisa é destacado por Barreto (1999). Segundo ele,

> *"a oferta e a demanda de informação em um determinado contexto informacional são representadas pelos estoques de informação institucionalizados disponíveis e pelas necessidades de informação da realidade onde o consumo se realiza. Assim, em uma realidade que demanda informação, é o processo de transferência que realiza a distribuição da informação estocada, com a intenção de configurar esta demanda.Em um mercado tradicional, oferta e demanda se ajustam considerando as condições específicas deste mercado. A curto prazo, a demanda de determinado produto aumenta ou diminui, a oferta tende a se ajustar a estas variações. O mercado de informação tem características que lhe são peculiares. Pesquisas já realizadas anteriormente permitem indicar que, no mercado de informação é a oferta que determina a demanda por informação" Barreto (1999).*

Esta posição é corroborada em Urqhart (1976) apud Barreto (1999) o qual afirma que:

> *"Estas propostas vêm de uma fonte que acredita implicitamente no homem econômico e no conceito de que uma demanda cria oferta. A ausência de qualquer resultado útil nas tentativas anteriores de pesquisa econômica da transferência da informação sugere que os testes básicos dos economistas não se aplicam a este campo (ciência da informação). A posição parece indicar que o homem da informação é substancialmente diferente do homem econômico. Sem dúvida, ele vive em um mundo onde oferta pode criar demanda".*

Segundo Cardoso Junior (2003), o processo de mudança organizacional nunca foi nem jamais será algo fácil de ser realizado. Contudo, transições sempre se mostraram necessárias para que uma empresa pudesse obter uma melhor posição competitiva no mercado em que atua. A busca pela sobrevivência dentro da sociedade de informação, inserida em ambientes crescentemente competitivos, demanda das empresas o

[8] Ver capítulo 3.

estabelecimento de bases de atuação em níveis cada vez mais intangíveis, peculiares e difíceis de serem reproduzidos.

Partindo do princípio de que a informação cada vez mais constituirá a base da competição entre as organizações, será necessário determinar claramente o papel que a informação vai desempenhar no projeto e execução da estratégia competitiva. O desafio de tirar proveito das possibilidades estratégicas da informação aperfeiçoada é muito mais difícil do que parece. A informação e a tecnologia da informação têm sempre desempenhado papéis tanto na definição quanto na execução de uma estratégia. À medida que a integração da estratégia e sua execução tornam-se o desafio organizacional mais importante, torna-se mais claro o papel da informação como ferramenta essencial para chegar a essa integração (FERREIRA,2002).

Focalizando a informação, as empresas passam a poder abordar a forma pela qual serão capazes de obter desempenho superior e transformar a estratégia em alguma coisa concreta e operativa. Existe uma dualidade em relação à informação. A tecnologia da informação propicia algumas novas alternativas para a elaboração de processos que criam e oferecem produtos e serviços. A informação representa uma das ferramentas mais importantes e maleáveis a serem utilizadas pelos executivos para diferenciar produtos e serviços. Em alguns casos a informação é o próprio produto (CARDOSO JUNIOR ,2003).

Cardoso Junior (2003) complementa que,

> *"Possuir os sistemas de tratamento de informações mais eficazes*[9]*, reconhecer a necessidade de absorver conhecimento externo, fazer conexões adequadas com as pessoas certas e estar alerta para o imprevisível são condições essenciais para que as organizações do conhecimento possam alcançar sucesso em confrontos nos quais a arena muda rapidamente".*

Os argumentos acima apresentados, bem como os que serão abordados no capítulo 5, permitem afirmar que o modelo sobre o qual o desenvolvimento de aplicações está baseado necessita de uma revisão conceitual importante. Isto porque, apesar do enorme avanço tecnológico, problemas identificados especificamente na área de sistemas operacionais há 28 anos atrás, permanecem presentes. Saliente-se que, no passado,

[9] Sistemas de tratamento de informações eficazes são aqueles que permitem a definição preliminar e detalhada da coleta de informações necessárias, segundo as demandas impostas pela situação, e que processam adequadamente os conteúdos reunidos, favorecendo a tomada de decisões estratégicas (CARDOSO JUNIOR,2003).

muitos problemas eram decorrentes da falta de qualidade do software, da rotatividade de pessoal, do elevado número de programadores envolvidos e da pouca prática no desenvolvimento de software e hardware (DEITEL,1990). Hoje, embora contemos com uma área de Engenharia de Software com vasta experiência acumulada, com métodos de gerência de projetos amplamente conhecidos, altos níveis de abstração suportados por ambientes de desenvolvimento e linguagens de programação, os problemas continuam recorrentes.

O enfoque que o mercado selecionou, não necessariamente fundamentando-se em critérios científicos, mas em leis de mercado, baseia-se em propostas originadas na década de 70 (**Unix**) e 80 (**Windows 2000** – baseado no *kernel Mach*). Embora haja alguma independência entre projetos de pesquisa e as soluções comerciais, a base conceitual continua praticamente a mesma: usuário, programa, multitarefa.

Em função dos argumentos arrolados e considerando as questões apresentadas em Bomberger (1992) e Colusa (1995), é possível afirmar que somente será possível minimizar os problemas relacionados à aplicação de tecnologias de informação nas áreas de computação ubíqua, comércio eletrônico e teletrabalho, a partir de um novo modelo de sistemas operacionais.

Se for feita uma comparação entre as projeções futuristas e a situação dos dias de hoje, não é difícil verificar a necessidade de profundas mudanças nos conceitos atuais, sobre os quais está baseada a indústria de software. Devem ser considerados, a partir dessa nova ótica, os problemas ainda não totalmente solucionados no contexto de computadores pessoais (*desktop*, *palmtop*, *pda*, etc), tais como: composição de componentes, reconfiguração automática, adaptação dinâmica, gerência de recursos e energia, intercomunicação transparente e metáforas de acesso às informações.

Segundo Nicol et al. (1989),

> *"Os sistemas operacionais devem ser o fator determinante nesta reformulação, já que é a partir das abstrações fornecidas pelos mesmos que são construídos os programas de aplicação".*

Por outro lado, pesquisas na área de robótica, adotando concepções e abstrações completamente diferentes, já há muito tempo tem demonstrado projetos bem sucedidos que envolvem aspectos como autonomia e comportamento inteligente, os quais os projetos de sistemas operacionais estão longe de atingir.

Estes aspectos, por si sós, caracterizam a necessidade de uma revisão dos conceitos da área de sistemas operacionais, mas este fato fica ainda mais evidenciado a partir da análise das demandas projetadas e da forma como são utilizadas tecnologias de informação para fazer-lhes frente (Capítulos 3 e 4).

Assim sendo, uma nova concepção de sistema operacional, baseado em conhecimento, constitui-se não só como elemento de estoque de informação, mas também como fator de geração de novas demandas.

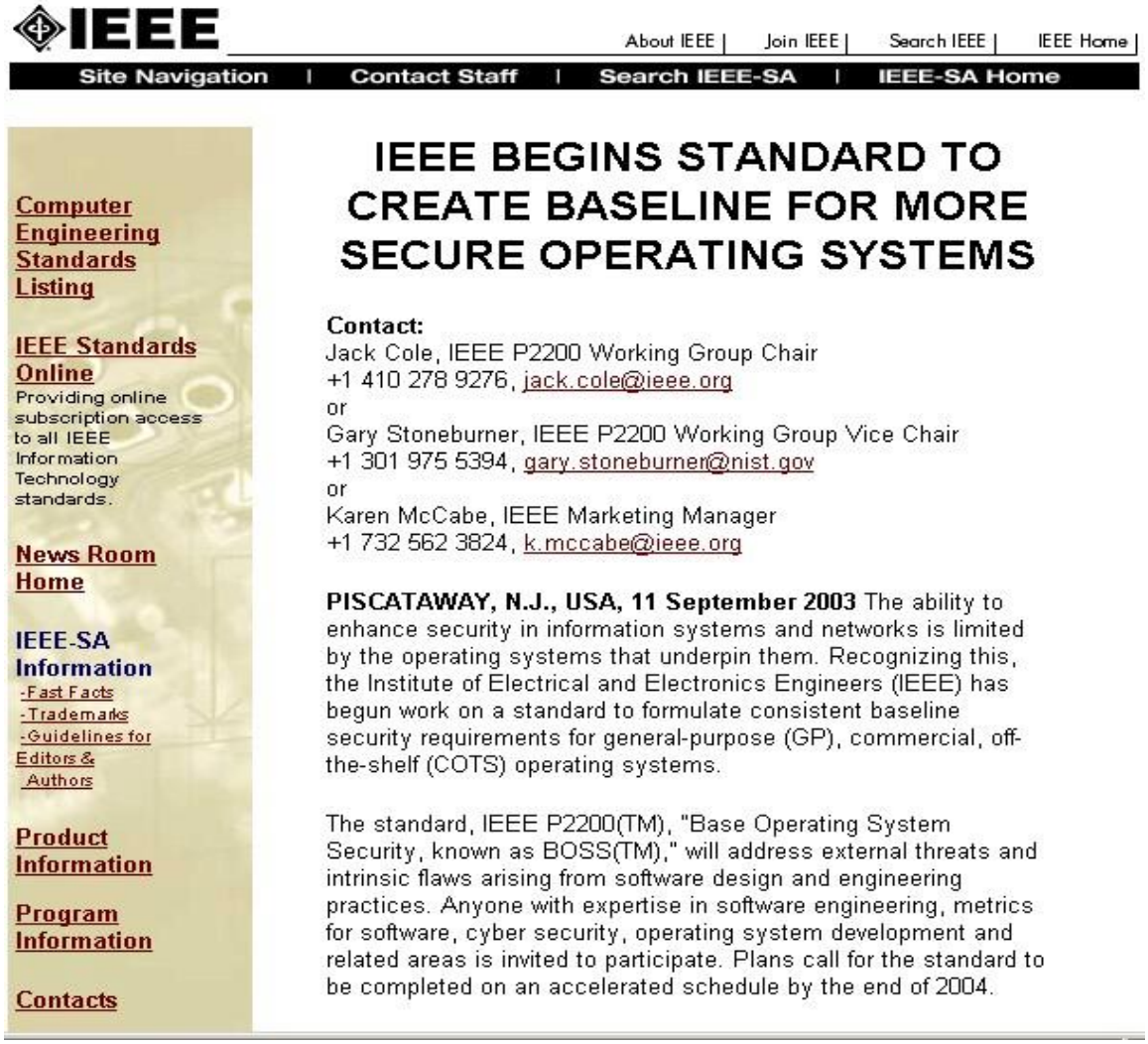


Figura 6 - Chamada da IEEE para criação de um padrão para construção de sistemas operacionais seguros.

Não obstante os argumentos apresentados, a Figura 6 caracteriza a importância de uma nova abordagem na construção de sistemas operacionais. A chamada da IEEE[10]

[10] Realizada dois meses antes da data da defesa da presente tese de doutorado.

reconhece que a base sobre a qual as TICs estão sendo concebidas necessita de reformulação e conclama a comunidade acadêmica e comercial a fazer parte de uma força-tafera encarregada de propor a base para a construção de sistemas operacionais seguros.

Os argumentos apresentados a seguir contribuem também para caracterizar que a engenharia de desenvolvimento de sistemas operacionais, sendo a base sobre a qual toda a tecnologia de informações é construída, está relacionada à área de engenharia de produção, já que a alteração na forma como os sistemas operacionais são concebidos, promove uma propagação em toda a cadeia de suporte ao desenvolvimento das TICs (Figura 7).

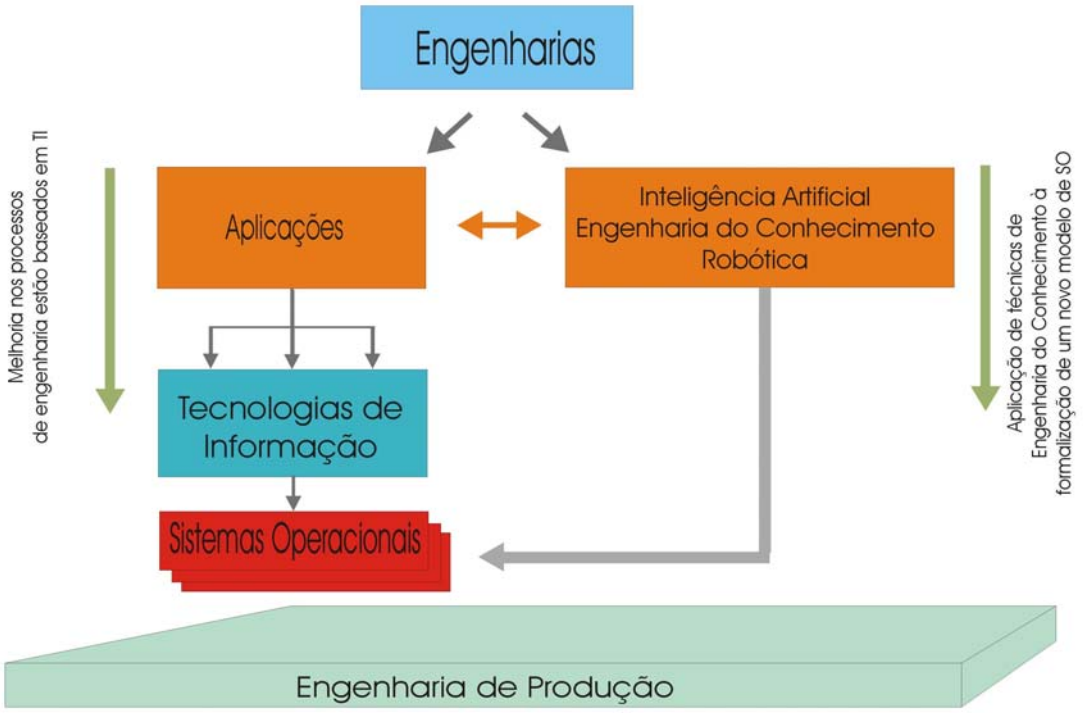


Figura 7 - Delimitação da proposta

Ressalta-se, assim, o ineditismo deste empreendimento teórico, na medida em que além de inexistir na literatura especializada um modelo descritivo da relação direta entre os conceitos de conhecimento, comportamento inteligente e modelo de mundo hiperdimensional como catalisadores de um modelo de sistema operacional, há o reconhecimento formal de instituições do porte da IEEE reconhecendo a necessidade da mudança de paradigma na concepção dos sistemas operacionais.

1.4 Procedimentos Metodológicos

Esta seção apresenta a classificação da pesquisa quanto à sua natureza, à abordagem do problema e aos objetivos e procedimentos utilizados (SILVA e MENEZES 2000). Além disso, define-se como o estudo procedeu o modelo de pesquisa e como o trabalho está estruturado.

Segundo Ruiz (2002), pesquisa científica é o método de abordagem de um problema em estudo que caracteriza o aspecto científico de uma pesquisa.

1.4.1 Delimitação do tema de pesquisa

A delimitação do tema de pesquisa é apresentada como uma etapa de melhor compreensão do assunto proposto (LAKATOS,1992 apud FERREIRA 2002). As subdivisões quanto à delimitação da proposta, à delimitação temporal da pesquisa e ao tipo de estudo realizado, são apresentadas na seqüência.

1.4.1.1 Delimitação da aplicação da proposta

Como delimitação desta abordagem, busca-se a identificação dos requisitos funcionais que permitam o desenvolvimento de uma nova geração de sistemas operacionais cuja abstração principal seja a gerência do conhecimento e do comportamento inteligente.

Limita-se a propor uma nova concepção de sistema operacional para computadores de uso pessoal (*desktops, palmtops, notebooks*), excluindo-se, portanto, toda a classe de computadores usados como servidores corporativos, estações de trabalho e *mainframes,* embora muitas das considerações expendidas possam ser aplicadas também a estas categorias de sistemas computacionais.

Assim sendo, esta tese restringe-se a realizar uma análise conceitual das noções fundamentais que sustentam a concepção de um sistema operacional baseado em conhecimento e, com isso, estabelecer os princípios do trabalho nesta área. Compreende um primeiro estabelecimento de requisitos funcionais e não funcionais de um projeto de software. Não pretende, portanto, desenvolver um sistema operacional completo. Contudo, em função do tamanho e da complexidade do projeto, a técnica da Engenharia

de Software de prototipação é utilizada objetivando validar alguns aspectos que dão sustentação ao modelo proposto.

Esta tese não elabora a teoria da inteligência de máquinas[11], mas antecipa uma variedade de trabalhos experimentais que o desenvolvimento desta estrutura pode comportar. Pensamos, portanto, que todo o trabalho sistemático, relacionado à construção do modelo proposto, está por fazer.

1.4.2 Delimitação temporal da pesquisa

Em relação à delimitação temporal, é apresentada, a seguir, uma breve ilustração do desenvolvimento dos sistemas operacionais, a partir da década de 50. Esta visão cronológica refere-se a alguns marcos originados por modelos, obras e autores que são importantes como referência nos assuntos. O gráfico em azul mapeia o número de referências bibliográficas (utilizadas no texto) em cada ano enquanto as setas identificam eventos e marcos históricos ocorridos na mesma época. Este gráfico permite ao leitor permite ao leitor posicionar a publicação em relação aos horizontes de pesquisa à época da publicação.

Este trabalho pretende abordar trabalhos relevantes publicados sobre os temas envolvidos dentro desta linha de tempo. A Figura 8 apresenta não somente a abrangência temporal dos trabalhos consultados, mas também identifica eventos históricos ocorridos à data das publicações analisadas, permitindo uma localização histórica mais precisa dos eventos e dos fatores que influenciaram as publicações.

1.4.3 Tipo de Estudo Realizado

Segundo Hoppen, Lapointe e Moreau (1996), é importante a identificação do tipo de estudo em questão: trata-se de um estudo longitudinal ou em corte transversal?

O tipo de estudo realizado nesta pesquisa é de caráter longitudinal, pois o alvo da pesquisa é um processo dinâmico, implicando em se conhecer as origens, evolução e conseqüências de um fenômeno (PINSONNEAULT e KRAEMER,1993 apud HOPPEN, LAPOINTE e MOREAU,1996).

[11] Consultar Costa (1993) como um possível ponto de partida neste sentido.

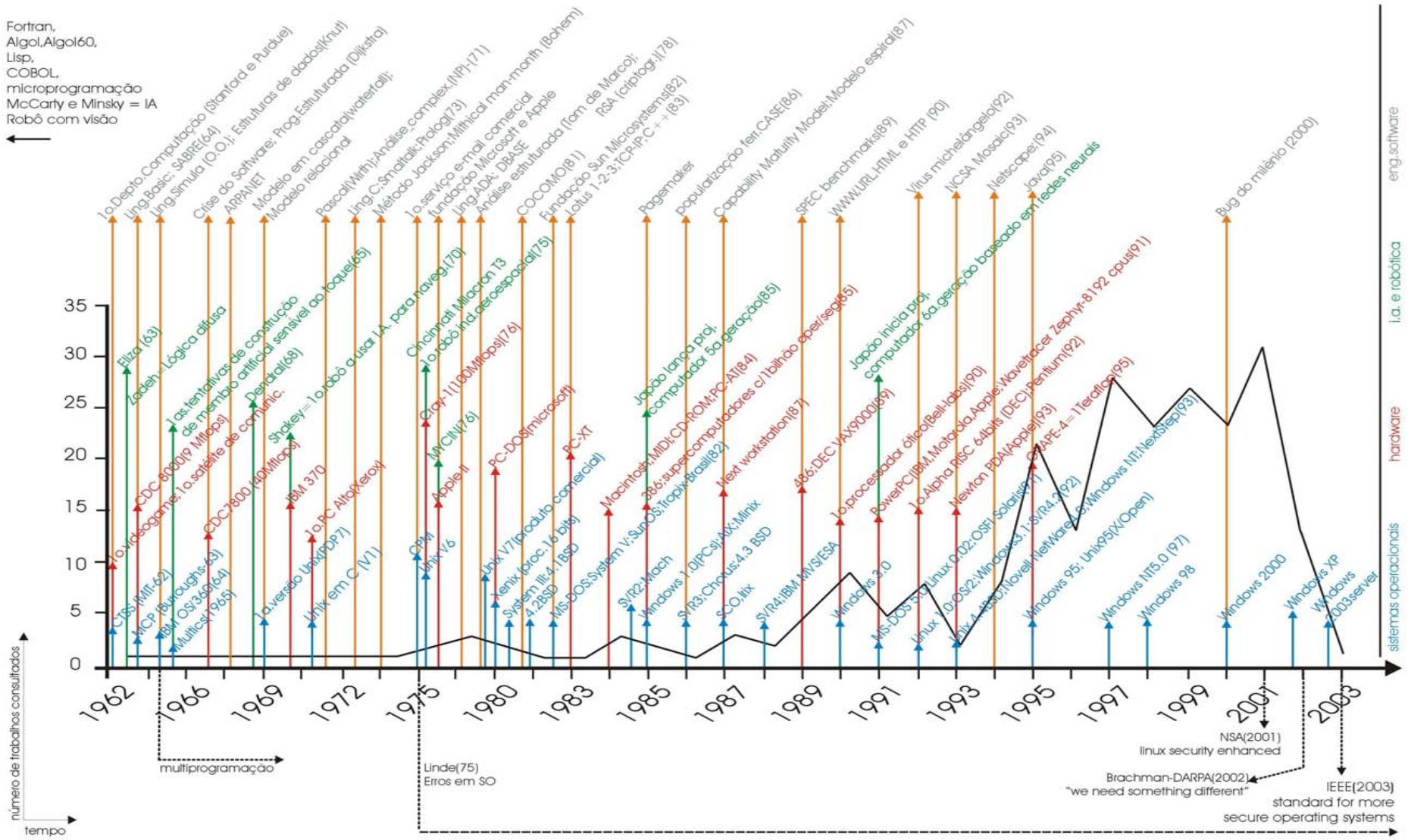


Figura 8 - Abrangência temporal x número de trabalhos consultados

Campbell e Stanley (1963) apud Hoppen, Lapointe e Moreau (1996) descrevem o principio do estudo longitudinal como aquele que permite medir as dimensões do objeto de um estudo antes e depois da intervenção de um fenômeno, para determinar se este fenômeno tem ou não efeitos. Os estudos longitudinais dão uma confiança maior ao nível de inferências causais, comparados com o princípio da pesquisa em corte transversal[12], porque estabelecem uma prioridade temporal (PINSONNEAULT e KRAEMER,1993 apud HOPPEN, LAPOINTE e MOREAU,1996).

A partir do cenário apresentado na introdução do capítulo, verificou-se a necessidade de uma revisão do atual modelo de construção de sistema operacional, visando identificar a relação entre a forma de construção, os problemas encontrados e a maneira de corrigi-los. Para tanto, realizou-se um estudo de abrangência com relação aos seguintes aspectos:

- Identificação das principais características de cada forma clássica de organização descritas na literatura;
- Identificação das principais demandas a serem suportadas pelos projetos de sistemas operacionais, bem como de suas características;
- Identificação dos principais problemas relatados na literatura, para os quais o atual modelo não apresenta soluções satisfatórias;
- Análise sobre as abordagens de tecnologia de informação utilizadas para superar as limitações que os projetos apresentam.

1.4.4 Classificação da Natureza da Pesquisa

A pesquisa foi desenvolvida no contexto do projeto de sistemas operacionais baseados em conhecimento. O projeto considera o estabelecimento de uma nova base conceitual como premissa básica para a solução de problemas recorrentes na área de Sistemas Operacionais.

Trata-se de uma pesquisa teórica para a construção dos fundamentos de um novo modelo de sistema operacional baseado na abstração de conhecimento e em comportamento inteligente. Segundo Oliveira (2002), a pesquisa aplicada requer determinadas teorias ou leis mais amplas como ponto de partida e tem por objetivo pesquisar, comprovar ou rejeitar modelos teóricos e aplicá-los às diferentes necessidades

[12] De acordo com Pinsonneault e Kraemer (1993) apud Hoppen, Lapointe e Moreau (1996), num estudo em corte transversal o pesquisador coleta dados em um momento preciso do tempo, junto à amostra selecionada para representar a população alvo. Mais tarde, o pesquisador pode generalizar a toda a população as descobertas feitas na amostra para o instante de tempo em que o estudo for feito.

humanas. Vale-se de contribuições de teorias e leis já existentes, tendo em vista uma grande gama de interesses, principalmente econômicos. Na maioria, é feita a partir de objetivos que visam sua utilização prática (PARRA e SANTOS, 1999 apud FERREIRA, 2002).

1.4.5 Classificação da Abordagem do Problema

A abordagem do problema é qualitativa por envolver um estudo teórico que não tem a preocupação de quantificar dados. Desta forma, não requer o uso de recursos e técnicas estatísticas. Para Silva e Menezes (2000,pág.20), "a pesquisa qualitativa considera que há uma relação dinâmica entre o mundo real e o sujeito, isto é, um vínculo indissociável entre o mundo objetivo e a subjetividade do sujeito que não pode ser traduzido em números". A interpretação dos fenômenos e a atribuição de significados são básicas no processo de pesquisa qualitativa. Não requer o uso de métodos e técnicas estatísticas. O ambiente natural é a fonte direta para coleta de dados e o pesquisador é o instrumento chave.

1.4.6 Classificação dos Objetivos

Conforme apresentado em Silva e Menezes (2000,pág.21), a pesquisa é classificada como exploratória, uma vez que visa "proporcionar maior familiaridade com o problema com vistas a torná-lo explícito ou a construir hipóteses. Envolve levantamento bibliográfico; entrevistas com pessoas que tiveram experiências práticas com o problema pesquisado e análise de exemplos que estimulem a compreensão."

Para Ruiz (2002), a pesquisa exploratória é usada quando um problema é pouco conhecido, ou seja, quando as hipóteses ainda não foram claramente definidas. Seu objetivo, portanto, consiste numa caracterização inicial do problema, sua classificação e reta definição. Constitui, pois, o primeiro estágio de toda a pesquisa científica; não tem por objetivo resolver de imediato um problema, mas tão somente apanhá-lo, caracterizá-lo.

Neste sentido, embora a problemática seja amplamente conhecida, como a abordagem proposta não foi implementada, o objetivo do trabalho é abrir uma nova linha de pesquisas para enfocar o problema sob outras perspectivas.

1.4.7 Classificação dos Procedimentos Técnicos

Segundo Silva e Menezes (2000,pág.21), a pesquisa bibliográfica é "elaborada a partir de material já publicado, constituído principalmente de livros, artigos periódicos e eventualmente com material publicado na Internet." Para Oliveira (2002), "a pesquisa bibliográfica tem por finalidade conhecer as diferentes formas de contribuição científica que se realizam sobre determinado assunto ou fenômeno".

Segundo Ruiz (2002,pág.52), experimentar, ou realizar experimentos, significa exercer positivo controle sobre as condições presumivelmente relevantes, relativamente a determinado evento; significa reproduzir, repetir o fenômeno dentro de um plano de modificações sistemáticas de suas variáveis independentes, com o objetivo de descobrir as condições antecedentes responsáveis pelo evento subseqüente, ou efeito, ou variável dependente assumida como objeto de pesquisa.

Os procedimentos técnicos utilizados foram a pesquisa bibliográfica e a pesquisa experimental, esta restrita à construção de protótipos de validação.

1.4.8 Procedimentos para a elaboração do trabalho

Segundo Ruiz (2002,pág.48), a pesquisa científica é a realização concreta de uma investigação planejada, desenvolvida e redigida de acordo com as normas consagradas pela ciência. Neste tópico apresentam-se as etapas da elaboração da pesquisa, bem como o seu detalhamento.

A pesquisa foi desenvolvida de acordo com as seguintes etapas:

- Planejamento de pesquisa;
- Levantamento bibliográfico, leitura e identificação de textos relevantes;
- Revisão bibliográfica;
- Proposição do modelo;
- Experimentação (construção de protótipos);
- Análise dos resultados.

Na etapa de planejamento foi elaborado o projeto de pesquisa, envolvendo a definição do tema, problemas e questões de pesquisa, objetivos e resultados esperados, estratégia de pesquisa, conteúdo para revisão bibliográfica e cronograma de execução.

Na etapa de levantamento bibliográfico, leitura e identificação de textos relevantes, foram pesquisados conceitos referentes aos vários enfoques de:

- Construção de núcleos de sistemas operacionais;
- Técnicas de obtenção de informações do núcleo;
- Modelo evolutivo de sistemas operacionais;
- Demandas de computação ubíqua, comércio eletrônico e teletrabalho;
- Técnicas de tecnologia de informações para instrumentação do sistema operacional;
- Pesquisas em robótica.

Figura 9 - Organização dos diretórios preservando a origem das informações coletadas.

Para tanto foram utilizados livros, revistas, apostilas, dissertações, Internet, entrevistas e materiais disponibilizados por vários órgãos e instituições. O material, em grande parte em formato digital, foi classificado tendo em vista preservar a data de obtenção e a origem. A Figura 9 ilustra a forma de armazenamento das informações coletadas. A metodologia de armazenamento das informações digitais permitiu recuperar não somente o *site* de origem (no caso **www.cin.mcgill.ca/~zelek**), mas também o nome do arquivo original (**TR-CIM-96-01.ps.gz**) e a data e hora do acesso (**29/03/2002 às 18:14h**). Além disso, utilizou-se um esquema de conversão de formato de arquivo que, além de manter o objetivo inicial de preservar sua origem, permitiu sua indexação e posterior consulta sem a necessidade de constantes conversões de formato. No exemplo, o nome do arquivo baixado era: **TR-CIM-96-01.ps.gz**, o que remetia à necessidade de

uma operação de descompactação (extensão **.gz**) e uma operação de conversão do arquivo **CIM-96-01.ps** (formato *postscript*) para o formato **pdf** (*portable document format,* compatível com o software *Acrobat Reader*), operação esta realizada utilizando-se o software *ghostview*. Assim, a regra adotada foi acrescentar ao nome original do arquivo a extensão.**pdf** transformando o nome do arquivo em: **TR-CIM-96-01.ps.gz.pdf**.

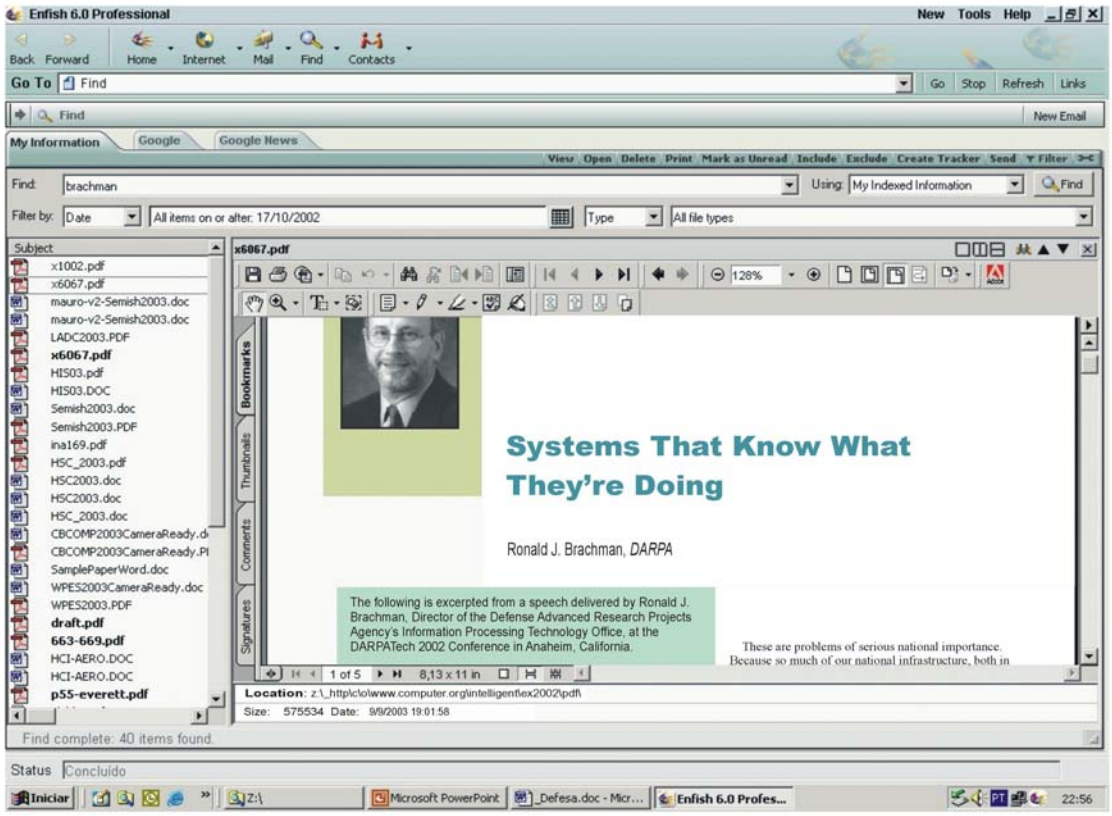


Figura 10 - Software utilizado para indexação da base digital

Para facilitar o acesso à base de informações digitais, utilizou-se o software Enfish Onespace (Figura 10), da empresa Enfish Corporation (www.enfish.com), como ferramenta de indexação das páginas html, arquivos de texto, planilhas e arquivos pdf coletados durante a fase de levantamento bibliográfico.

A etapa de revisão bibliográfica, incluída neste texto, apresenta de forma descritiva o embasamento científico da pesquisa, cujo conteúdo serviu de base para a proposição do enfoque de sistema operacional baseado em conhecimento. A construção do modelo foi realizada concomitantemente com o levantamento e revisão bibliográfica e envolveu um processo interativo de refinamentos sucessivos com o professor orientador, Dr. Roberto

Pacheco, e com os Professores Dr. Rômulo da Silveira e Pedro Bertolino[13]. Com o primeiro refletimos sobre a aplicabilidade do modelo proposto e com o segundo sobre seu enquadramento na área de sistemas operacionais. Já com o Prof. Bertolino mantivemos inestimáveis discussões sobre a elaboração do conceito de conhecimento em um sistema operacional baseado em conhecimento. Ainda nessa etapa mantivemos uma entrevista com o Prof. Dr. Bernard Ziegler, durante o evento "AI, Simulation and Planning in High Autonomy Systems. AIS'2002", ocorrido em Lisboa - Portugal em Abril de 2002, quando o modelo DEVS foi introduzido no modelo proposto.

Uma vez identificados os requisitos funcionais e não funcionais, foram desenvolvidos alguns protótipos para validar aspectos específicos do modelo proposto e, finalmente, foi realizada a redação do volume final do trabalho.

1.5 Estrutura do Trabalho.

Nesta seção apresenta-se a estrutura do trabalho em capítulos encadeados segundo a relação de dependência apresentada na Figura 11(pág.54). Assim, o trabalho está organizado como a seguir se descreve.

O capítulo 1 apresentou o tema de pesquisa, seus objetivos e caracterizou os procedimentos metodológicos empregados para sua execução. Considerando-se os objetivos deste trabalho, alguns conceitos fundamentais devem ser examinados antes de se abordar a solução proposta. Desta forma, o Capítulo 2(Pág.55) apresenta uma revisão sobre o estado da arte em construção de sistemas operacionais, de modo a permitir uma posterior análise sobre a forma como são concebidos e comparar seus requisitos com os que as novas demandas lhes estão impondo. O capítulo inicia discutindo o conceito de sistema operacional e alguns aspectos importantes que formam o modelo atual. A seguir, é descrito um modelo cíclico de evolução do hardware. A seção seguinte apresenta os modelos clássicos de construção de núcleo de sistemas operacionais, discutindo suas principais características, vantagens e desvantagens. Para facilitar o processo de análise dos dados levantados, as informações foram condensadas em tabelas. Contudo, uma descrição mais detalhada destes aspectos foi incluída como Apêndice 2 – Descrição detalhada das arquiteturas de núcleos de sistemas operacionais (Pág. 311). O capítulo encerra

[13] Filósofo e Mestre em Epistemologia da Ciência.

com uma análise sobre o processo evolutivo e apresenta uma analogia entre o modelo evolutivo dos sistemas operacionais e o modelo cíclico de evolução de hardware.

Uma vez que os vários enfoques de construção de núcleos de sistemas operacionais foram abordados, o passo seguinte foi identificar as demandas que deverão disparar uma nova fase no ciclo evolutivo apresentado na seção 2.6(Pág. 85).

O capitulo 3(Pág.94) apresenta uma revisão sobre as tendências das pesquisas em computação ubíqua, comércio eletrônico e teletrabalho e os requisitos que as aplicações deverão apresentar para operarem neste contexto. O capítulo encerra com um resumo (Tabela 19Pág.110) onde são relacionados os requisitos funcionais que as áreas estudadas deverão demandar.

O capitulo 4(Pág.111) analisa algumas das abordagens de tecnologias de informação cujo emprego está sendo considerado, atualmente, para permitir que o sistema operacional passe a atender algumas demandas ainda não satisfeitas pelo modelo atual. Neste contexto, a seção 4.2(Pág.111) apresenta algumas pesquisas que procuram instrumentalizar o sistema operacional através do emprego de técnicas da área de Inteligência Artificial. A seção 4.3(Pág.121) apresenta outras tecnologias que estão sendo empregadas com o mesmo propósito. A seção 4.4 (Pág.134) apresenta alguns conceitos fundamentais da área de Robótica que devem ser examinados antes de se abordar a solução proposta. Desse modo, este capítulo explora alguns conceitos importantes utilizados nesta área, os quais vão fundamentar a abordagem adotada neste trabalho. O capítulo conclui com uma análise que demonstra claramente a dissociação entre os objetivos dos projetos de sistemas operacionais, as demandas identificadas no capítulo 3 e a orientação dos projetos de Robótica. O capítulo encerra com uma tabela (Tabela 20Pág.140) sintetizando as características das estratégias apresentadas .

O capítulo 5(Pág.142) detalha a caracterização do problema de pesquisa, através do estabelecimento de seus conceitos básicos. Também examina a questão de seu domínio e campo de aplicação, gerando uma declaração formal, uma verdadeira definição do problema. Esta inicia a partir de algumas considerações sobre o estado da arte em projetos de sistemas operacionais, tendo em vista caracterizar que o modelo legado possui deficiências conceituais. A seguir, enfatiza a dissociação entre as novas demandas projetadas e a forma de concepção dos projetos baseados no atual modelo. Algumas

destas falhas e as tarefas cuja realização elas impedem, são discutidas sob os aspectos de confiabilidade, segurança e complexidade de utilização. O capítulo encerra analisando as projeções e tendências que ressaltam a necessidade de uma nova concepção de projeto de sistema operacional.

O capítulo 6(Pág.182) apresenta os fundamentos conceituais de um novo modelo de sistema operacional, baseado em abstração de conhecimento e comportamento inteligente, como uma possível solução para os problemas identificados nos capítulos anteriores, utilizando-se de técnicas também apresentadas naqueles capítulos.

O capítulo 7(pág 224) apresenta o modelo proposto. Ele inicia com a definição de sistema operacional baseado em conhecimento e, em seguida, aborda seus requisitos funcionais e não funcionais. Por último, apresenta uma descrição do modelo proposto e efetua uma análise de sua viabilidade, tanto em termos de construção como de aplicação.

O Capítulo 8(Pág 289) apresenta as conclusões e recomendações finais do trabalho. Como anexo inserimos um material complementar o qual é referido ao longo do texto.

Conforme referido anteriormente, Figura 11(pág.54) apresenta a relação de dependência entre os capítulos do trabalho.

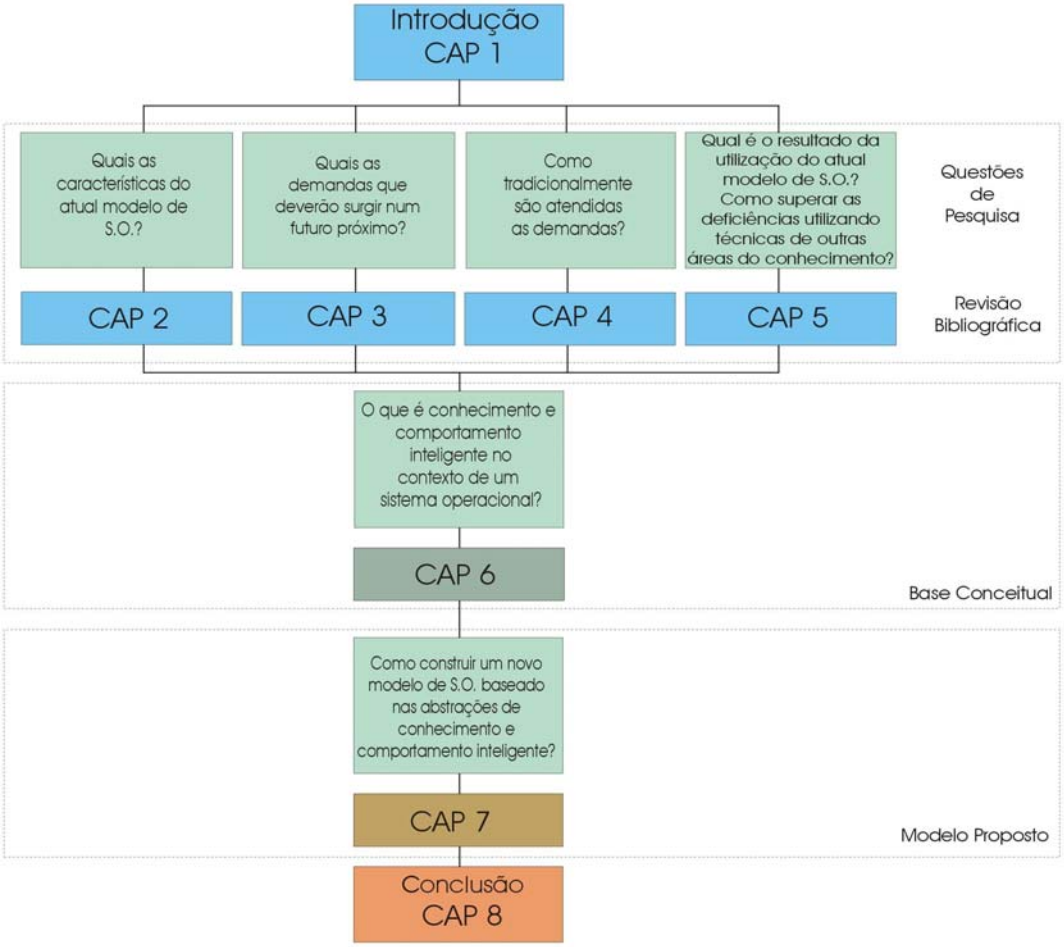


Figura 11 - Estrutura do trabalho.

2 O ESTADO DA ARTE NA TEORIA CLÁSSICA DE SISTEMAS OPERACIONAIS

"Era uma vez, há não muito tempo, e todo mundo sabia o que era um sistema operacional. Era um software complexo vendido pelo fabricante do computador, sem o qual nenhum outro programa poderia funcionar naquele computador. Ele fazia os discos girarem, iluminava os terminais, e geralmente mantinha registros do que o hardware estava fazendo e porque. Programas de aplicação (do usuário) faziam solicitações para o sistema operacional executar várias funções; os usuários raramente falavam diretamente com o Sistema Operacional. Hoje esses limites não estão muito claros. O surgimento das interfaces gráficas com o usuário (GUI – Graphical User Interface), linguagens de scripts e macros, conjuntos de aplicações que podem trocar informações de forma simplificada e transparente, e o aumento na popularidade das redes e dados distribuídos - todos estes fatores mesclaram as distinções tradicionais. Os ambientes de computação de hoje consistem em níveis de hardware e software que interagem para formar um todo quase orgânico" Burk e Horvath (1997).

2.1 Introdução

Considerando-se os objetivos deste trabalho, alguns conceitos fundamentais devem ser examinados antes de se abordar a solução proposta. Assim sendo, este capítulo explora conceitos básicos e faz algumas considerações parciais. A primeira questão a ser apresentada é o conceito propriamente dito, pois este é o fundamento das atuais abordagens.

2.2 O Conceito de Sistema Operacional

Segundo Tanenbaum (1987), um sistema operacional pode ser definido como :

"Uma camada de software que de forma segura, permite a abstração e a multiplexação dos recursos físicos".

Sendo uma camada de software interposta (Figura 12) entre as aplicações e o hardware propriamente dito, sua estrutura tem um impacto fundamental na performance e abrangência das aplicações que são sobre ele construídas.

Figura 12 - Estrutura funcional de um sistema operacional

O termo hardware designa em geral todo o aparato eletrônico e eletro mecânico fisicamente perceptível. No entanto, um componente de hardware em particular estabelece a forma como o software será implementado em última instância. Este componente denomina-se processador. Embora haja várias linhas arquitetônicas, via de regra o funcionamento de um processador pode, abstratamente, ser descrito como em Tanenbaum (1984,pág.22), através da implementação em linguagem pascal da Figura 13.

Imediatamente acima do hardware, está localizada a camada de software normalmente referida como camada de baixo nível, a qual concentra a porção de código que emite comandos específicos às controladoras dos periféricos conectados ao computador.Acima desta camada, reside o código do sistema operacional propriamente dito. Esta camada é responsável por exportar, através de uma *Application Program Interface* (API), um nível de abstração de hardware tal que permita aos programadores desenvolver a próxima camada de software (camada de aplicação) sem ter que se preocupar com detalhes de baixo nível relativos à gerência de recursos de hardware.

Sob este aspecto de separação entre hardware e software, Tanenbaum (1984, pág.117) afirma:

> *"... o limite entre hardware e software não é bem definido e, além disso, está constantemente deslocando-se. Os primeiros computadores tinham instruções para executar operações lógicas, aritméticas, de deslocamento, etc, as quais eram executadas diretamente pelo hardware. Para cada instrução havia um circuito de hardware especial responsável pela execução. Alguém poderia desparafusar o painel e indicar para o conjunto de componentes eletrônicos usados por uma instrução de divisão (pelo menos em princípio). Em computadores modernos, não é mais possível isolar o circuito de divisão porque o mesmo não existe mais. Todas as instruções disponíveis ao nível de máquina convencional (por exemplo, as instruções aritméticas, lógicas,...) são executadas um passo de cada vez, por um interpretador implementado ao nível de microprogramação. Como a microarquitetura é definida pelo hardware, ela é geralmente muito primitiva e de difícil programação".*

```
1    Type Word = ....;                    (*especificação do tipo word *)
2      Ender = ....;                      (* endereço*)
3      Mem = array[0..4095] of Word; (* alocação  da memória do "computador simulado" *)
4      (* ..... *)
5    procedure interpretador (memoria:Mem;EndInicio:Ender);
6    var
7      ProgramCounter,                    (* apontador da proxima instrução a ser executada *)
8      DataLocation      : Ender;         (* apontador para endereço dos operandos da instrução *)
9      RunBit            : 0..1;          (* flag indicando quando parar o processador *)
10     InstrType         : integer;       (* tipo da instrução a ser executada *)
11   begin
12     ProgramCounter := EndInicio;(*endereço da primeira instrução a ser executada quando a máquina é ligada *)
13     RunBit := 1;                       (* estabelece o padrão de repetição do processador *)
14     While RunBit = 1 do
15     begin
16       (* busca a próxima instrução e armazena no registrador de instruções do processador – FETCH *)
17       InstrRegister := memoria[ProgramCounter];
18       (* avança o ProgramCounter para apontar para a próxima instrução a ser executada *)
19       ProgramCounter := ProgramCounter + 1;
20       (* decodifica a instrução e classifica o seu tipo (aritmética,lógica, ...)  *)
21       DeterminaTipoInstrucao(InstrRegister,TipoInstr);      (* "chamada de procedimento" *)
22       (* identifica e localiza o endereço dos operandos da instrução- se houverem *)
23       IdentificaOperandos (TipoInstr,InstrRegister,DataLocation,NecessidaDados);
24       (* busca dados dos operandos na memória – se houverem *)
25       if NecessitaDados then  Operandos := memoria[ DataLocation];
26       (* avança o processo executando a instrução *)
27       Execute (TipoInstr,Operandos,memoria, ProgramCounter, RunBit);
28     end; (* while *)
29   end; (* Interpretador *)
```

Figura 13 - Pseudo-código que implementa um interpretador de máquina virtual.

Para um entendimento sobre as características do atual modelo, faz-se necessária a revisão de alguns conceitos importantes a ele relacionados. A seção seguinte apresenta esta revisão.

2.2.1 Outros Conceitos Que Formam o Modelo Atual

O ponto de partida desta análise é o significado da definição apresentada por Tanenbaum (1987), uma vez que a discussão conceitual que se segue está a ele associada. Para tanto, cada trecho da definição antes apresentada é discutido a seguir:

- **"Uma camada de software"**: o sistema operacional é um programa de computador que, como qualquer outro, pode apresentar os problemas de qualidade e produtividade tão amplamente discutidos pela área de Engenharia de Software[14]. Portanto, um projeto de

[14] Segundo Deitel (1990,pg 15), os problemas relacionados ao desenvolvimento de sistemas operacionais na década de 60 contribuíram significativamente para o surgimento da área de Engenharia de Software, uma vez que, até o início da década de 70, os fabricantes só vendiam hardware. O *software* e a documentação eram fornecidos como bônus, sem custos e, portanto, sem responsabilidade por parte dos fabricantes de

sistema operacional está sujeito às mesmas considerações relativas às atividades de criação, entrega e manutenção de sistemas de software.

- **"que permite a abstração":** Segundo Larman (2000) apud Rumbaugh (1997), um sistema, do mundo real ou de *software,* é, de modo geral, excessivamente complexo; portanto é necessário decompô-lo em pedaços para melhor compreensão e administração de sua complexidade. Estes pedaços podem ser representados como modelos que descrevem e abstraem aspectos essenciais do sistema. Desse modo, um passo útil na construção de um sistema de software é, primeiro, criar modelos que permitam a organização e comunicação dos detalhes do problema do mundo real com o qual o sistema está relacionado, bem como do sistema a ser construído (LARMAN,2000). Como as aplicações que serão executadas num sistema operacional só têm acesso às funcionalidades disponibilizadas pelas abstrações fornecidas pelo modelo, a estruturação do mesmo pode ser um fator determinante na qualidade, robustez e eficiência dos produtos finais**.**
- **"Que permite o compartilhamento de recursos":** Neste sentido, o sistema operacional é o principal controlador dos recursos físicos disponíveis, tais como: processador, memória e periféricos. A principal finalidade é homogeneizar e compatibilizar as velocidades e formas de operação, em alguns casos extremamente discrepantes. Segundo Deitel (1990), como gerente de recursos, o modelo de sistema operacional estabelece a forma de interação com o usuário (*user interface*) e gerencia o compartilhamento dos recursos físicos (hardware) e lógicos (dados) entre aplicações de um mesmo usuário ou de usuários diferentes.
- **"De forma segura"**: Compartilhamento e proteção são requisitos conflitantes. O compartilhamento de recursos entre usuários torna a utilização dos equipamentos mais eficiente e flexível. Contudo, este mesmo requisito torna mais vulnerável o acesso ao conteúdo de dados neles produzidos e armazenados. Em função disto, o aspecto segurança é um dos que atualmente demanda grandes investimentos em termos de pesquisa, no sentido de produzir soluções mais eficientes e eficazes.

Alguns outros conceitos também são importantes e precisam ser retomados[15]. O primeiro refere-se ao conceito de **núcleo**. Segundo Oliveira,Carissimi e Toscani (2000), a arquitetura de um sistema operacional corresponde à imagem que o usuário tem dele. Esta imagem é definida pela interface através da qual o usuário acessa os serviços proporcionados pelo sistema operacional. E esta interface é formada pelas chamadas de sistema e pelos programas de sistema. Os programas solicitam serviços ao sistema operacional através de chamadas de sistema. A parte do sistema operacional responsável por implementar o conjunto de chamadas de sistema, é normalmente referida como: **núcleo** ou ***kernel***. O tamanho desta porção de código é um dos fatores que determinam a classificação nas várias categorias a serem apresentadas na seção 2.4. Portanto, um núcleo de sistema operacional corresponde ao conjunto de procedimentos que

---

hardware. Estes forneciam listas de erros (*bugs*) conhecidos como "cortesia". A partir do momento em que a IBMpassou a vender software e hardware, é que surgiu a indústria de *software* independente e discussões mais amplas sobre questões relativas ao *software*.

[15] Não se pretende, aqui, detalhar os conceitos de sistemas operacionais, mas estabelecer uma base conceitual a partir da qual o entendimento do texto seja facilitado.

implementam os serviços que possibilitam a um programador desenvolver uma aplicação sem preocupar-se com os detalhes específicos de *hardware*. À lista dos serviços disponibilizados pelo sistema operacional, dá-se o nome de Interface com o programa de aplicação - comumente referida como API (OLIVEIRA,CARISSIMI e TOSCANI, 2000).

Um segundo conceito importante é relacionado à **gerência de memória**[16] e diz respeito ao escopo de memória visível por uma aplicação. Geralmente o núcleo do sistema pode acessar qualquer endereço de memória física. No entanto, seguindo a mesma filosofia adotada para proteger o código do núcleo, geralmente a memória física é virtualizada (quando há suporte de *hardware* adequado através de um componente denominado *Memory Management Unit* (MMU) para proteger as aplicações entre si e, assim, obter-se um maior grau de confiabilidade no sistema. O espaço de endereços de memória a que uma aplicação tem acesso denomina-se **espaço de endereçamento** (*address space*).

Um terceiro conceito importante relaciona-se à **diferenciação entre um programa** (seqüência de instruções) **e a sua efetiva execução**. Segundo Oliveira, Carissimi e Toscani (2000), "não há uma definição objetiva, aceita por todos, para a idéia de processo. Na maioria das vezes um **processo** é definido como um programa em execução". De forma geral, durante a execução de um programa ele passa por vários estados lógicos, tais como: executando (quando detém efetivamente o processador), pronto para executar (quando aguarda em uma das filas do sistema para receber o processador), esperando (quando aguarda em alguma das filas do sistema pela ocorrência de algum evento – ex: término de uma operação de entrada/saída). O núcleo do sistema é o módulo responsável pelo escalonamento dos vários processos a serem executados em determinado momento.

Segundo Machado e Maia (2002,pág.83), a partir do desenvolvimento do sistema **Toth,** em 1979, foi introduzido o conceito de processos leves (*lightweight*) em que o espaço de endereçamento de um processo era compartilhado por vários programas. Apesar de revolucionária, a idéia não foi utilizada comercialmente e somente em meados da década de 80, com o desenvolvimento do sistema **Mach** (RASHID et al., 1989), ficou clara a separação entre o conceito de processo e ***thread***. A partir do conceito de múltiplos

[16] Não é objetivo deste trabalho detalhar os aspectos relativos ao funcionamento dos esquemas de gerência de memória virtual. O leitor que pretender aprofundar-se no assunto pode consultar Deitel (1990) e Tannenbaum (1987).

threads, é possível projetar e implementar aplicações concorrentes de forma eficiente, pois um processo pode ter partes diferentes do seu código sendo executadas em paralelo, com um menor *overhead* do que usando vários processos. Como as *threads* de um mesmo processo compartilham o mesmo espaço de endereçamento, a comunicação entre *threads* não envolve mecanismos lentos de intercomunicação entre processos (*interprocess communication* - IPC), aumentando, conseqüentemente, o desempenho da aplicação (MACHADO e MAIA, 2002).

A Tabela 1 ilustra a diferença de performance entre a implementação do conceito de processo e do conceito de *thread,* em termos de tempo de execução.

Tabela 1 - Latência de processos e *threads*

Implementação	Tempo de criação (μs)	Tempo de sincronização (μs)[17]
Processo	1700	200
Thread	52	66

Fonte: Adaptado de Vahalia apud Machado e Maia (2002,pág.88)

Um quarto conceito refere-se aos **estados de execução do núcleo** do sistema operacional. Havendo suporte de *hardware* adequado (anéis de proteção), é possível proteger o código do núcleo de forma a evitar que erros em programas de aplicação possam corromper o conteúdo de estruturas de dados nele mantidas e, assim, prejudicar seu funcionamento. Quando isto é possível, o código do núcleo que executa sob proteção do *hardware* é dito como executando em **modo supervisor** (*kernel*); por conseguinte, o código que executa fora deste anel de proteção é dito executando em **modo usuário**. Este conceito é importante porque a transição do modo usuário para o modo supervisor demanda vários ciclos de cpu, o que pode comprometer a eficiência de um núcleo de sistema operacional. A linha que separa o código que executa em modo usuário, daquele que executa em modo supervisor, é denominada de linha vermelha (*red line*). Isto porque a passagem de um modo de execução para outro geralmente consome muitos ciclos de processador, o que pode comprometer a performance do sistema como um todo.

A seguir é apresentado o modelo evolutivo proposto por Siewiorek, Bell e Newell (1983) como forma de caracterizar o processo evolutivo de *hardware*. Esta análise é

[17] μs = microssegundos.

importante uma vez que, como o núcleo de um sistema operacional é o nível de software mais próximo do hardware, alterações na construção do hardware conduzem a alterações na forma como as abstrações de hardware são apresentadas aos usuários de níveis superiores ao sistema operacional.

2.3 O modelo evolutivo de hardware

Siewiorek, Bell e Newell (1983) apresentam um modelo de ciclo evolutivo de hardware que pode, resumidamente, ser caracterizado como envolvendo 7 fases. A primeira fase considera um modelo muito simples de arquitetura, onde o processador controla diretamente o acesso aos periféricos, emitindo sinais de controle em seqüências de tempo pré-estabelecidas. O processo evolutivo vai especializando os componentes envolvidos. E, no meio do processo, aparece o conceito de acesso direto à memória (*Direct Memory Access* -DMA), onde o periférico transfere dados diretamente para a memória. O processador envia o comando e recebe um sinal quando a operação é encerrada. O ciclo encerra com uma arquitetura que envolve um processador de entrada/saída cujos componentes individuais caracterizam-se como aqueles descritos na primeira fase. Assim, cada componente individual evolui a partir da primeira fase tornando o conjunto mais poderoso e complexo. A Figura 14$_{(pág.64)}$ caracteriza este modelo.

Um exemplo real de uma arquitetura de processador de Entrada e Saída (E/S) em fase 7 é apresentado por FiucZynski et al. (1998). Os autores descrevem um projeto de sistema operacional extensível denominado **SPINE**, o qual permite que as aplicações executem código diretamente sobre o adaptador de rede, obtendo ganhos significativos de performance através de transferências diretas entre dispositivos (sem a utilização da memória do computador hospedeiro) e da divisão na alocação de protocolos entre computador hospedeiro e placa de rede.

Kumar et al. (2001) afirmam que dispositivos como cartões de rede e discos rígidos, geralmente incluem um processador programável e memória. Isto permite aos dispositivos conter facilidades sofisticadas em *firmware*. Por exemplo, um controlador de disco rígido pode suportar algoritmos agressivos de escalonamento da cabeça de disco diretamente implementados no *firmware* através de programação concorrente utilizando

múltiplas *threads* de controle. Apesar disto, estes controladores geralmente possuem processadores e memória com capacidade limitada.

Um exemplo concreto refere-se a evolução de hardware baseado em lógica difusa. Segundo Patki (1996), três décadas de sistemas baseados em lógica difusa conduziram ao desenvolvimento da primeira geração de hardware baseado em lógica difusa, cujo principal objetivo foi obter soluções de controle difuso mais rápidas. Até recentemente, a maioria destas soluções eram implementadas como módulos de software, executando sobre microprocessadores convencionais, computadores pessoais ou estações de trabalho (*workstations*).

No entanto, a aplicação destas soluções em ambientes com requisitos de tempo-real orientou as pesquisas no sentido do desenvolvimento de hardware baseado em lógica difusa. Num primeiro momento, a solução emergiu em uma forma de implementação em chips (Figura 14$_{\text{pág.64}}$ - fase 2) de uma máquina de inferência difusa incorporando controladores difusos baseados em regras (PATKI, 1996) .

Assim, criou-se uma impressão de que a única aplicação desta tecnologia seria em aparelhos domésticos e produtos de consumo. No entanto, a necessidade de soluções para a área de processamento de informação, tais como: construção, simulação e modelagem de aplicações, recuperação de informações em bases de dados e outras aplicações semelhantes, estão conduzindo as pesquisas em direção ao desenvolvimento de microprocessadores com suporte a software e dispositivos periféricos (Figura 14$_{\text{pág.64}}$ - fase 7) (PATKI, 1996) .

Segundo Patki (1996), as primeiras tentativas em estudar sistemas de chaveamento em lógica difusa (MARINOS[18], 1969; SIY E CHEN[18], 1992), e sua implementação eletrônica, resultaram no desenvolvimento de um componente flip-flop (HIROTA e OZAWA[18], 1989). A partir daí, seguiram-se várias implementações de hardware de processamento de inferência difusa, tais como Costa et al. (1995), Eichfeld et al.[18](1995), Manzoul[18](1990), Watanabe,Dettloff e Yount[18] (1990), que demonstraram aplicações de controle com vários níveis de inferência difusa. Surgiu então a medida FLIPS (*Fuzzy Logic Inferences Per Second),* que é análoga àquela usada para os processadores tradicionais (MIPS– *Million Instructions Per Second*).

[18] apud PATKI (1996).

Como referido anteriormente, os esforços atualmente na área de hardware difuso (COSTA et al., 1995) estão mudando no sentido do desenvolvimento de um microprocessador com sua própria Unidade de Lógica e Aritmética Difusa (Figura 14$_{\text{pág.64}}$ - fase 8) para uso nas mais variadas aplicações. Este enfoque está abrindo um novo campo de pesquisas que deverá cobrir os seguintes aspectos (PATKI, 1996):

- Arquitetura de um conjunto de instruções (*instruction set*) para processamento de informações difusas;
- Construção de uma unidade lógica e aritmética difusa, como uma unidade funcional;
- Estudos de precisão relacionados à representação de conjuntos difusos e seus respectivos graus de pertinência (*membership*);
- Estudos no emprego de arquiteturas de memória associativa e seu impacto em microprocessadores difusos.

Em função disto, há duas correntes de desenvolvimento a saber (PATKI, 1996):

- Desenvolvimento de um conjunto de instruções (*chip set*) de lógica difusa: esta tendência visa construir pequenos módulos adaptáveis nos atuais PCs, cujo objetivo é atender a necessidades mais imediatas;
- Desenvolvimento de um microprocessador difuso e do software associado: que envolve o desenvolvimento de um processador difuso, nos moldes dos processadores RISC, juntamente com um co-processador neuronal. Este enfoque, segundo Patki (1993) apud Patki e Raghunathan (1996), procura agregar as vantagens que estas duas técnicas têm a oferecer, quais sejam: a capacidade de aprendizado do conhecimento do especialista, através de dados numéricos, e a capacidade de representação deste conhecimento através de uma relação de entrada/saída envolvendo raciocínio difuso.

Patki e Raghunathan (1996) apud Alag e Patki (1995) complementam que,

> *"...a maioria dos trabalhos baseadas em hardware difuso descritos na literatura descreve as soluções como módulos em hardware que suportam o mapeamento de conjuntos difusos em conjuntos difusos através de um módulo fuzificator, uma máquina de inferência e um módulo defuzificator. Estes módulos são integrados na forma de ASICs[19,20](Application Specific Integrated Circuit). Em função das limitações deste enfoque, as soluções em hardware para controle difusos passaram a ser abordadas através de um enfoque arquitetônico, o que originou os seguintes segmentos de pesquisa: co-processadores difusos dedicados[21]; ASICs de inferência difusa e a construção de conjuntos de instruções (chip set) difusas genéricos[22]".*

Este último enfoque tem a finalidade de preencher a lacuna entre as duas propostas anteriores.

---

[19]ASICs são chips projetados para aplicações específicas ao contrário dos circuítos integrados que controlam funções como a memória RAM de um PC.
[20]Figura 14 $_{\text{pág.64}}$ - fase 1.
[21] Figura 14$_{\text{pág.64}}$ - fase 2.
[22]Figura 14 $_{\text{pág.64}}$ - fase 7.

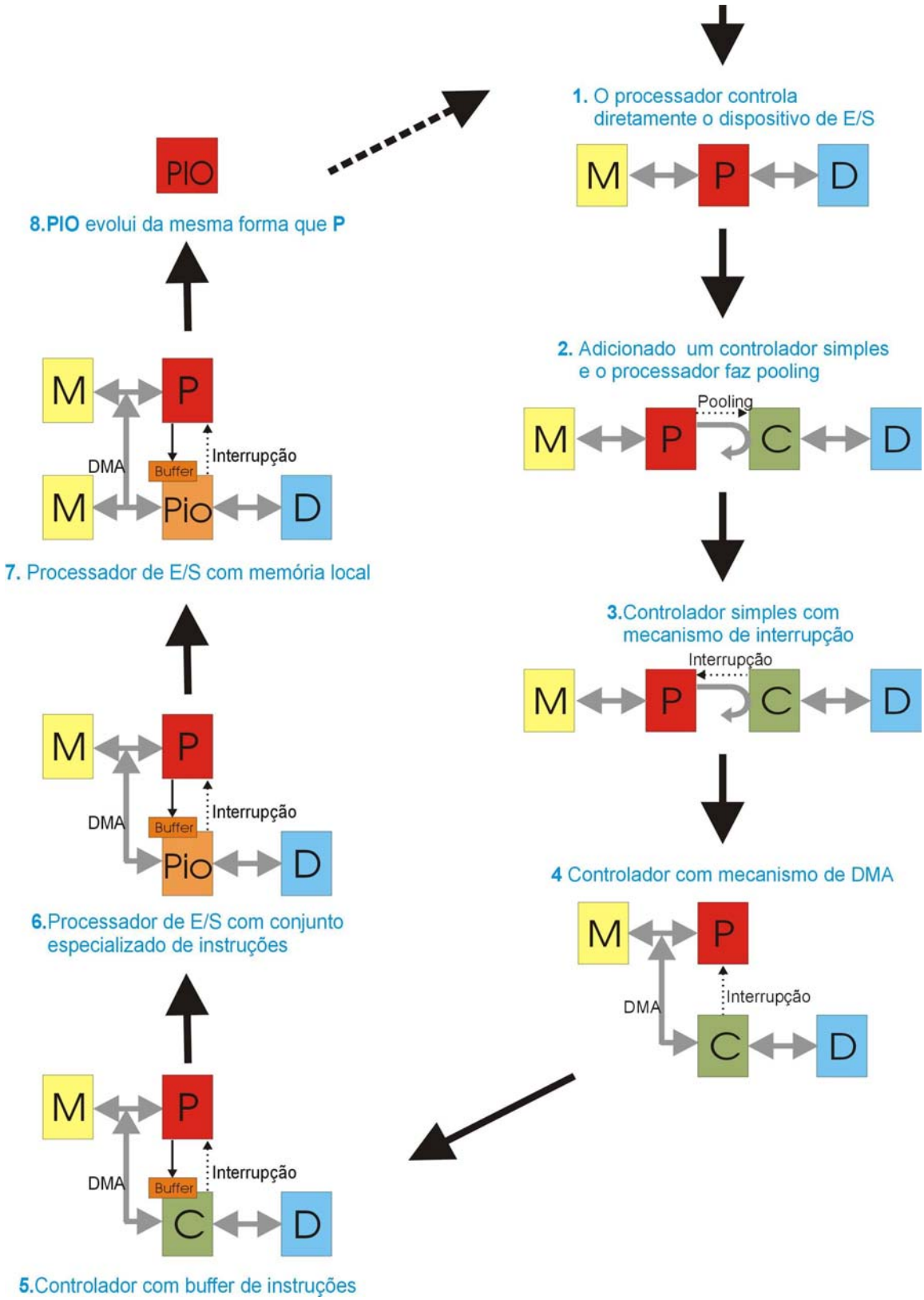


Figura 14 - Modelo Evolutivo de Hardware.

FONTE: Adaptado de Siewiorek, Bell e Newell (1983).

A partir desta revisão conceitual e tomando-se por base o mecanismo evolutivo de hardware, na próxima seção são analisadas as várias formas de estruturas organizacionais de núcleos de sistemas operacionais.

2.4 Estruturas organizacionais clássicas

Os primeiros sistemas operacionais começaram a surgir a partir da segunda geração de computadores, em meados da década de 50 e início da década de 60, mais como um produto de fatoração de rotinas de *run-time*[23] comuns do que efetivamente resultado de intenção em desenvolvê-los.

A seguir relacionam-se as principais abordagens estruturais dos sistemas operacionais descritas na literatura:

- Modelos monolíticos;
- Modelos em camadas (ou anéis)
- Modelos hierárquicos;
- Modelos baseados em micronúcleos (*microkernel*);
- Modelos cliente-servidor;
- Núcleos coletivos:
    - *Library operating systems;*
    - *Modelos exonúcleo (exokernel);*
    - *Modelos cache kernel;*
    - *Modelos nanonúcleo (nanokernel);*
- Modelos de máquinas virtuais;
- Modelo baseado em objetos;
- Modelos baseados em reflexão computacional;
- Modelos baseados em conhecimento;
- Modelos híbridos.

As próximas seções apresentam as principais características destas abordagens estruturais. Para uma análise mais detalhada, consultar o Apêndice 2 – Descrição detalhada das arquiteturas de núcleos de sistemas operacionais na página 311.

[23] Procedimentos necessários durante a execução de um programa, tais como: bibliotecas de manipulação de arquivos, de tratamento de números com ponto flutuante, etc.

2.4.1 Estruturas Monolíticas

Os antigos projetos de núcleos monolíticos foram escritos como uma agregação de toda a funcionalidade necessária para permitir a utilização do hardware, como processos, gerenciamento de memória, multiprogramação, comunicação de processos, acesso a dispositivos, sistema de arquivos e protocolos de rede. Projetos mais recentes (principalmente **Unix**) possuem uma arquitetura modular, o que oferece a possibilidade de adição/remoção de serviços em tempo de execução (*run-time*). Sistemas desta geração geralmente eram concebidos para atuarem como servidores em situações de execução contínua e não apresentavam muita flexibilidade para inserção/remoção de periféricos. Por outro lado, esta geração de sistemas apresentava um alto grau de estabilidade (DUMON, 1997).

A Tabela 2 apresenta um resumo das características desta abordagem para concepção de núcleo de sistemas operacionais. Uma descrição mais detalhada pode ser obtida na seção 10.2.1 (Pág 312).

Tabela 2 - Características de núcleos monolíticos

Características	
• Precursores das modernas arquiteturas de sistemas operacionais (DEARLE e HULSE, 2000); • Os antigos projetos foram escritos como uma agregação de toda a funcionalidade necessária para permitir a utilização (DUMON, 1997); • Projetos mais recentes possuem uma arquitetura modular que oferece a possibilidade de adição/remoção de serviços em tempo de execução (*run-time*) (DUMON, 1997); • Concebidos para atuarem como servidores em situações de execução contínua (DUMON, 1997).	
Vantagens	
• Modelos modulares permitem a carga de módulos em tempo de execução (MACHADO e MAIA, 2002); • Extensibilidade (DUMON, 1997); • Código não confiável pode ser isolado através de mecanismos de proteção de memória em tempo de execução (SELTZER et al, 1996); • Extensões ao sistema através de *device drivers* geralmente são confiáveis em sua totalidade (SELTZER et al, 1996).	
Desvantagens	
• Todos os módulos executam em modo supervisor, o que caracteriza um problema de segurança (MACHADO e MAIA, 2002); • O aspecto de maior impacto no projeto de núcleos monolíticos está relacionado à rigidez dos projetos, o que compromete operações de expansão (para fazer frente a novas demandas, correção e testes do sistema; • A evolução de núcleos monolíticos é extremamente difícil devido à fraca estruturação interior e à natureza monolítica (DEARLE e HULSE, 2000).	
Exemplos	
• Unix (AHL, 2001); • OS/360, VMS e VME (DEARLE e HULSE, 2000); • Linux (MACHADO e MAIA, 2002); • MS-DOS (AHL, 2001);	• Sprite (OUSTERHOUT[24] apud BALLESTEROS, 1997); • Plan9 e Inferno (OUSTERHOUT apud BALLESTEROS, 1997); • Solaris MC (OUSTERHOUT apud BALLESTEROS, 1997).

[24] John. K. Ousterhout et al. The Sprite network operating system. *Computer*, 21(2):23-36, February 1988.

2.4.2 Estruturas Baseadas em Camadas

Segundo Montez (1995), o primeiro sistema em camadas foi construído por Dijkstra e seus estudantes. A estrutura do sistema foi publicada no primeiro Simpósio em Princípios de Sistemas Operacionais da ACM, em 1967. O sistema **THE** era organizado em sete níveis. Cada nível consistia de uma coleção de objetos abstratos e um conjunto de regras controlando a interação entre eles. As regras eram codificadas em procedimentos do sistema operacional denominados "operações primitivas" (COMER, 1984).

A Tabela 3 apresenta um resumo das características desta abordagem para concepção de núcleo de sistemas operacionais. Uma descrição mais detalhada pode ser obtida na seção 10.2.2 (Pág.313).

Tabela 3 - Características de núcleos em camadas.

Características	
• Introduz o conceito de programação modular; • Pode ser considerado como uma generalização do modelo anterior; • É projetado como um conjunto de camadas sobrepostas; • Cada uma provê um conjunto de funções para a camada superior; • A camada superior é implementada em termos das funções providas pela camada inferior; • Não há como um nível inferior enviar comandos para um nível superior.	
Vantagens	
• Facilidade de manutenção uma vez que todo um nível pode ser substituído sem afetar o comportamento dos níveis inferiores e superiores; • Facilidade de construção do sistema operacional (AHL, 2001).	
Desvantagens	
• Projetos antigos baseados nesta estrutura apresentavam deficiências principalmente devido ao fato de que as camadas inferiores precisavam ser genéricas (NICOL, 1987); • Perda de performance (NICOL, 1987); • A modularidade do sistema permanece isolada dentro do contexto do software do sistema operacional não podendo ser usado pelos programadores de aplicação (SELTZER et al., 1996).;	
Exemplos	
• Unix e Windows 2000. (MACHADO e MAIA, 2002); • AIX (AHL, 2001);	• THE (DIJKSTRA, 1968 apud DEITEL, 1990).

2.4.3 Estruturas Hierárquicas

A estrutura hierárquica, embora semelhante à anterior, apresentava o diferencial de permitir a transposição de níveis da hierarquia funcional. Outro aspecto importante é que a estrutura hierárquica é baseada em funções e não em processos.

A Tabela 4 apresenta um resumo das características desta abordagem para concepção de núcleo de sistemas operacionais. Uma descrição mais detalhada pode ser obtida na seção 10.2.3 (Pág.315).

Tabela 4 - Características de núcleos hierárquicos.

Características
• A estrutura hierárquica, embora semelhante à anterior, apresentava o diferencial de permitir a transposição de níveis da hierarquia funcional; • Segundo Zancanella e Navaux (1993): o Baseado em funções e não em processo; o O conceito de módulo é usado para expandir vários níveis por meio de um único módulo; o Cada nível é caracterizado por um conjunto de funções estaticamente identificadas e conhecidas; o Entrelaçamento de módulos, como ocorre com os módulos do sistema de arquivos e memória virtual.
Vantagens
• Os módulos são agrupados em grandes componentes. Cada um deles é responsável por implementar algum subconjunto coerente da funcionalidade geral do sistema (REDELL et al, 1979); • Modularização, Reusabilidade e Facilidade para manutenção (ZANCANELLA e NAVAUX, 1993).
Desvantagens
• Dificuldade de adaptação e substituição de módulos para atender requisitos específicos das aplicações (SELTZER et al., 1996); • Os benefícios são exclusivamente internos ao sistema operacional (ZANCANELLA e NAVAUX, 1993); • Os mecanismos de proteção eram orientados para um ambiente monousuário, onde erros de programação são mais graves do que ataques externos (BACK, 2001).
Exemplos
• PILOT (DALAL et al.,1980).

2.4.4 Estruturas Baseadas em Micronúcleos

Segundo Liedtke (1995), sistemas baseados em micro-núcleos já eram construídos mesmo antes do próprio termo ter sido introduzido por Brinch-Hansen (1970) e Wulf et al. (1974). A proposta de micronúcleo está baseada na idéia de implementar fora do núcleo todos os serviços possíveis. Isto reduz o tamanho do código do núcleo e aumenta a flexibilidade do sistema como um todo. Serviços de mais alto nível, tais como: sistemas de arquivos, *device drivers*, gerência de processos e mesmo parte das funções de gerência de memória, podem ser construídos sob a forma de processos que executam sobre o micro-núcleo (MULLENDER et al., 1990).

A Tabela 5 apresenta um resumo das características desta abordagem para concepção de núcleo de sistemas operacionais. Uma descrição mais detalhada pode ser obtida na seção 10.2.4 (Pág.316).

Tabela 5 - Características de micronúcleos.

Características	
• Implementação fora do núcleo de todos os serviços possíveis; • Esta abordagem conduz a uma organização dos processos na forma de clientes e servidores; • O núcleo, executando em modo supervisor, encarrega-se de entregar as mensagens para o servidor responsável pelo atendimento; • Neste modelo, os componentes do sistema operacional podem ser pequenos e, como cada processo servidor executa como diferentes processos em modo-usuário, um determinado servidor pode falhar sem comprometer o restante do sistema operacional.	
Vantagens	
• Redução no tamanho do código do núcleo; • Aumento da flexibilidade do sistema como um todo; • Organização dos processos na forma de clientes e servidores; • Os componentes do sistema operacional podem ser pequenos; • Como cada processo servidor executa como em modo-usuário, um determinado servidor pode falhar sem comprometer o restante do S.O. • Estruturação mais modular do sistema; • Os servidores podem utilizar os mecanismos implementados no núcleo, da mesma forma que qualquer outro programa de aplicação; • A falha em um servidor pode ser isolada, não comprometendo os demais serviços do sistema, que se torna mais flexível e configurável; • É possível implementar diferentes estratégias e APIs em diferentes servidores e tolerar sua coexistência simultânea; • 2a geração: apresentava uma API ambígua (L4 e QNX).	
Desvantagens	
• A performance depende da eficiência da implementação de primitivas de comunicação entre processos.; • Apesar desta característica ser o ponto forte da abordagem, representa também o gargalo do sistema como um todo; • 1ª geração: apresentava uma API ambígua (Mach); • Como os serviços do sistema são disponibilizados via servidores, o processo de contabilização é mais complicado (BACK et al., 2001).	
Exemplos	
• Amoeba (BALLESTEROS, 1997); • Paramecium (BALLESTEROS, 1997); • Mach (RASHID et al., 1989); • L4 (DUMON, 1997); • BeOS (BEOS, 1998); • QNX (LEROUX, 2001; DUMON, 1997);	• Chorus (BALLESTEROS, 1997); • Spring (BALLESTEROS, 1997); • Fiasco (HOHMUTH, 1998); • Off (BALLESTEROS, 1997); • Off++ e vOff++ (BALLESTEROS, HESS, KON e CAMPBELL, 1999); • Per Brinch Hansen (1970) apud Engler,Kaashoek e O'Toole Jr. (1995).

2.4.5 Estruturas Configuráveis ou Adaptáveis

A abordagem de micronúcleo introduz o conceito de núcleos adaptáveis (*customizable kernels*), os quais podem ser dinamicamente especializados para atender a requisitos de performance e funcionalidade das aplicações. A motivação que conduziu a esta abordagem foi o fato de que as anteriores não permitiam a construção de sistemas que atendessem a todas as demandas de serviços e performance requisitados pelas aplicações.

A Tabela 6 apresenta um resumo das características desta abordagem para concepção de núcleo de sistemas operacionais. Uma descrição mais detalhada pode ser obtida na seção 10.2.5(Pág.319).

Tabela 6 - Características de núcleos configuráveis/adaptáveis.

Características	
• Aplicações podem alterar o comportamento do núcleo, permitindo que código da aplicação execute no espaço de endereços do núcleo. (SELTZER et al., 1996).	
Vantagens	
• Permite ajustar a performance e flexibilidade da aplicação (SELTZER et al., 1996); • O *overhead* das extensões fica restrito à complexidade e funcionalidade das extensões (Seltzer,Small e Smith, 1995); • O grau de granularidade dos filtros (*hooks*) facilita a instalação de código do usuário em substituição a código do núcleo (SELTZER,SMALL e SMITH,1995); • O grau de granularidade determina o grau de flexibilidade do sistema (SELTZER,SMALL e SMITH, 1995).	
Desvantagens	
• Código errado ou mal intencionado pode comprometer a integridade do núcleo (SELTZER et al., 1996); • Requer grande conhecimento sobre as demandas da aplicação ao sistema operacional e sobre a forma como seus módulos serão substituídos para adequar-se às necessidades da aplicação; • Possui um determinado número de interfaces não confiáveis para instalação das extensões (SELTZER et al., 1996); • Requer um sistema de *run-time* para suporte a proteção, isolamento, gerência de recursos e intercomunicação de processos (BACK et al., 2001); • Dificuldade de garantia contra mau uso de recursos (CPU, memória) (SELTZER et al., 1996); • Serviços para solução de problemas de segurança impõem complexidade ao projeto e compromete a performance (SELTZER et al.,1996).	
Exemplos	
• VINO (SELTZER et al.,1994; SELTZER et al., 1996)	• SPIN (BERSHAD et al., 1994)

2.4.6 Estruturas Baseadas em Núcleos Coletivos

Segundo Montz (1995), a estrutura de núcleos coletivos é uma variação do modelo de micronúcleo. Neste modelo, o núcleo é construído como uma coleção de processos independentes. A proposta deste tipo de estrutura é eliminar a noção de que um sistema operacional deve:

- Prover abstrações sobre as quais as aplicações são construídas;
- Suportar a multiplexação segura do hardware; e;
- Suportar as interações entre processos que implementam os serviços do sistema operacional.

A Tabela 7 apresenta um resumo das características desta abordagem para concepção de núcleo de sistemas operacionais. Uma descrição mais detalhada pode ser obtida na seção 10.2.6(Pág.323).

Tabela 7 - Características de núcleos coletivos.

Características	
• É uma variação do modelo de micronúcleo (MONTZ et al., 1995); • O núcleo deve conter somente os mecanismos que devem ser implementados com segurança; • A maior parte da funcionalidade do sistema é implementada ao nível do usuário e pode ser ajustada independentemente para cada aplicação; • Este modelo de estrutura "é o atual estado da arte para o projeto de sistemas operacionais, explorando de alguma forma as vantagens da técnica estruturada." (ZANCANELLA e NAVAUX,1993).	
Vantagens	
• Todas as decisões sobre políticas são tomadas em modo usuário. • Aplicações que não apresentam requisitos particulares podem ser ligadas às bibliotecas padrão do sistema (MONTZ et al., 1995); • Aplicações que apresentam requisitos diferentes podem implementar políticas específicas para adequar-se a estes requisitos (MONTZ et al., 1995); • Separação do núcleo em duas partes: mecanismos que manipulam recursos do sistema (tais como memória física, espaços de endereçamento), etc) e o modelo de programação que auxilia os programadores a usar efetivamente o sistema (ZANCANELLA e NAVAUX,1993).	
Desvantagens	
• Não localizadas na bibliografia consultada.	
Exemplos	
• Chorus (HERRMANN et al., 1988; BANINO, 1985); • V-Kernel (CHERITON et al., 1984);	• Mach (BLACK et al., 1986); • Amoeba (MULLENDER et al., 1990).

2.4.7 Estruturas Baseadas em Library Operating Systems

A linha dos *library operating systems* (DEARLE,2000) caracteriza-se como uma variação do modelo de micronúcleo onde grande parte da funcionalidade do sistema operacional é assentada em bibliotecas (LibOS) implementadas em modo usuário e específicas para cada aplicação.

Há duas correntes nesta linha, quais sejam:

- Exonúcleos *(exokernels)* e
- *Cache kernels*.

Segundo Engler e Kaashoek (1995), a diferença entre *Cache Kernel* e *Exokernel* está centrada em aspectos filosóficos. O *cache kernel* concentra-se primariamente em confiabilidade (*reliability),* ao invés de preocupar-se em exportar com segurança os recursos de hardware para as aplicações. Em função disto, esta é uma proposta limitada a estruturas baseadas em servidores.

A Tabela 8 apresenta um resumo das características desta abordagem para concepção de núcleo de sistemas operacionais. Uma descrição mais detalhada pode ser obtida na seção 10.2.7(Pág.324).

Tabela 8 - Características de núcleos baseados em library OS.

Características
• Variação do modelo de micronúcleo; • Duas correntes: exonúcleos e *cache kernels* (DEARLE et al. 1994; DEARLE e HULSE, 2000).
Vantagens
• Estrutura que permite extensibilidade com uma granularidade bastante fina (KAASHOEK, 1995).
Desvantagens
• Uma interface de tão baixo nível implica em que a freqüência das chamadas de sistema podem aumentar, comprometendo a performance (DEARLE et al. 1994); • Este comprometimento não é tão visível em implementações de núcleos monolíticos ou micronúcleos. (DEARLE et al. 1994).
Exemplos
• Grasshopper (DEARLE e HULSE, 2000).

A seguir são detalhadas as duas abordagens.

2.4.7.1 Estrutura em Exonúcleo

Os projetos de exonúcleo partem do princípio de que a criação de abstrações sobre o hardware é um enfoque errôneo para a construção de sistemas operacionais. Isto porque, independentemente do nível de abstração fornecido por um sistema operacional, sempre haverá alguma classe de aplicação para a qual as abstrações serão inadequadas. Em função desta reflexão, ao invés de prover todo um conjunto de abstrações, um exonúcleo exporta diretamente os recursos de hardware. A única responsabilidade do núcleo é prover acesso multiplexado e seguro aos recursos de hardware (KAASHOEK,1995).

A Tabela 9 apresenta um resumo das características desta abordagem para concepção de núcleo de sistemas operacionais. Uma descrição mais detalhada pode ser obtida na seção 10.2.7.1(Pág.325).

Tabela 9 - Características de exonúcleos.

Características	
• Ao invés de prover todo um conjunto de abstrações, um exonúcleo exporta diretamente os recursos de hardware; • A única responsabilidade do núcleo é prover acesso multiplexado e seguro aos recursos de hardware (KAASHOEK,1995); • Exonúcleo permite que programas do usuário sobrescrevam o código exportado pelo sistema e pelo próprio núcleo (DEARLE e HULSE, 2000); • A principal diferença entre este enfoque e aquele proposto pelos núcleos monolíticos, é que o limite da portabilidade não mais coincide com a interface entre as aplicações e o núcleo (KAASHOEK, 1995); • Segundo Engler e Kaashoek (1995) Máquinas virtuais e exonúcleos diferem sob os seguintes aspectos: o Máquinas virtuais emulam o hardware, enquanto os exonúcleos exportam funções primitivas de baixo nível; o O processo de emulação esconde informação, o que pode contribuir para o uso ineficiente dos recursos de hardware; o Um exonúcleo tenta expor toda a informação sobre o sistema e, explicitamente, comunica-se com um *library operating system*, ao invés de apresentar uma visão virtual; o Máquinas virtuais confinam sistemas operacionais especializados e processos associados em máquinas virtuais isoladas.	
Vantagens	
• Permitem aos usuários obter controle da funcionalidade do núcleo; • Uma interface exonúcleo garante mais poder para as aplicações e também promove um melhor compartilhamento de recursos (ENGLER e KAASHOEK, 1995); • Aplicações especializadas como SGBD módulos de *run-time*[25] de linguagens e *garbage collectors*[26], podem substituir a interface padrão e acessar o hardware diretamente para obter ganhos de performance (KAASHOEK, 1995).	
Desvantagens	
• A conseqüência imediata é a falta de segurança – como restringir o que os programas do usuário podem fazer. Em função disto o projeto, que pretende ser pequeno e simples, começa a tornar-se mais complexo com a adição de algoritmos de proteção no núcleo (DEARLE et al. 1994); • Dificuldade no projeto da exposição dos recursos de hardware de modo seguro e eficiente, tendo em vista atingir a alto grau de flexibilidade (DEARLE et al. 1994); • O limite da portabilidade não mais coincide com a interface entre as aplicações e o núcleo (KAASHOEK,1995); • Mesmo movendo a responsabilidade sobre gerência de recursos para o nível de usuário, o núcleo é responsável por fornecer mecanismos que garantam (*enforce*) determinada política (BACK et al., 2001); • Para evitar que uma aplicação não libere os recursos, deliberadamente ou por erro, o sistema *Aegis* emprega também um protocolo de cancelamento (*abort*) em que os recursos são recuperados à força. (MULLENDER et al., 1990); • Um dos problemas de disponibilizar-se uma interface de tão baixo nível é que a freqüência das chamadas de sistema pode aumentar demasiadamente (DEARLE et al. 1994).	
Exemplos	
• Aegis (KAASHOEK, 1995); • SPIN (BERSHAD et al., 1994);	• Nemesis (LESLIE et al., 1997; REED, DONNELLY e FAIRBARINS, 1997; e STEVEN, 1999).

2.4.7.2 Estruturas em *cache kernel*

Segundo Cheriton e Duda (1994) os projetos de *cache kernel* adotam uma filosofia diferente. Ao invés de exportar uma interface com o hardware, como fazem os projetos de exonúcleo, o enfoque *cache kernel* suporta um pequeno conjunto de abstrações primitivas para as quais todas as decisões sobre política são implementadas por bibliotecas em nível de usuário. Um *cache kernel* somente armazena *threads*, espaços de memória, comunicação entre processos e núcleos. O *cache kernel* executa em modo supervisor. Para cada objeto armazenado, é associada uma aplicação do núcleo. Esta aplicação efetivamente executa em modo usuário e implementa funções de gerência de espaço de endereçamento e escalonamento de suas próprias *threads*.

[25] Módulos (programas, bibliotecas) complementares necessários durante a execução de um programa.
[26] São módulos responsáveis pela desfragmentação e reorganização do espaço da memória dinâmica.

A Tabela 10 apresenta um resumo das características desta abordagem para concepção de núcleo de sistemas operacionais. Uma descrição mais detalhada pode ser obtida na seção10.2.7.2(Pág.328).

Tabela 10 - Características de cache kernel.

Características
• Os projetos de *cache kernel,* ao invés de exportar uma interface com o hardware, suportam um pequeno conjunto de abstrações primitivas para as quais todas as decisões sobre política são implementadas por bibliotecas em nível de usuário (CHERITON et al., 1984); • Um *cache kernel* somente armazena *threads*, espaços de memória, comunicação entre processos e núcleos (CHERITON et al., 1984); • O *cache kernel* executa em modo supervisor (CHERITON et al., 1984); • Explora troca de mensagens em memória e memória física controlada pela aplicação para permitir a construção de aplicações sofisticadas que desejam construir seu próprio núcleo de sistema operacional (CHERITON e DUDA,1994); • Permite customização específica por aplicação de modo semelhante à proposta microkernel, através de uma interface mínima com o hardware(CHERITON e DUDA,1994).
Vantagens
• A interface de código de baixo nível permite às aplicações controlar o gerenciamento de recursos de acordo com qualquer política desejada. (CHERITON et al., 1984); • O enfoque de *caching* temporário previne as aplicações de exaurir a fonte de descritores de objetos como pode ocorrer em sistemas convencionais. (CHERITON et al., 1984); • O modelo de *cache* reduz a complexidade do núcleo, se comparado com os modelos de micronúcleos. (CHERITON et al., 1984); • Significativa redução em tamanho e complexidade do código – exemplo o código de gerência de memória virtual em V++ é de 1500 linhas em C++ enquanto é de 14400 para SunOS 4.1.2 (Sparc) e quase 20.000 linhas para o *kernel* Mach (MIPS) (CHERITON e DUDA, 1994).
Desvantagens
• O *cache kernel* não tenta gerenciar todos os objetos do sistema, somente os objetos ativos (CHERITON et al., 1984); • Como serve apenas como um *cache* de objetos ativos, fica por conta dos *kernels* das aplicações em modo usuário gerenciar os demais objetos (CHERITON et al., 1984).
Exemplos
• V++ (CHERITON et al., 1984; CHERITON e DUDA,1994).

2.4.8 Estruturas Baseadas em Nanonúcleos

Segundo Dearle e Hulse (2000), as abstrações fornecidas pelos sistemas operacionais convencionais suportam processos que acessam dados que estão em memória real ou virtual. Dados armazenados não podem ser acessados da mesma forma. Isto leva à necessidade de uma interface (API) diferente para cada classe de dados. Por exemplo: o sistema **Unix** possui chamadas de sistema para alocação de memória virtual e outras para acesso a arquivos.

Segundo Hulse e Dearle (1998), em função das questões identificadas relativas à construção de aplicações persistentes sobre os núcleos de sistemas operacionais existentes, tornou-se necessário o desenvolvimento de um novo enfoque, o qual foi denominado nanonúcleo[27]. Esta camada implementa toda a porção de código dependente de hardware. Isto permite a construção de um sistema operacional independente de

[27] Também referida na literatura como *Hardware Abstraction Level -HAL.*

hardware. Este nível de código contém primitivas para alocar memória física, mapear a tradução de endereços, mascarar interrupções, refletir exceções e interrupções e troca de contexto.

A Tabela 11 apresenta um resumo das características desta abordagem para concepção de núcleo de sistemas operacionais. Uma descrição mais detalhada pode ser obtida na seção 10.2.8(Pág.329).

Tabela 11 - Características de nanonúcleos

Características	
• Abordagem para suporte à aplicações persistentes (DEARLE e HULSE, 2000); • Esta camada implementa toda a porção de código dependente de hardware (HULSE e DEARLE,1998); • Código contém primitivas para alocar memória física, mapear a tradução de endereços, mascarar interrupções, refletir exceções e interrupções e troca de contexto (HULSE e DEARLE,1998); • Para Bomberger (1992) : o Núcleo *stateless*; o Armazenamento persistente; o *Capabilities*.	
Vantagens	
• Suporte à persistência ortogonal[28] (HULSE e DEARLE,1998); • Tratam todos os dados de forma idêntica (como objetos persistentes) e usam uma interface (API) única para manipular tais objetos; • Permite a construção de um sistema operacional independente de hardware (HULSE e DEARLE,1998); • Vantagens em relação à implementação de persistência sobre núcleos monolíticos (DEARLE e HULSE, 2000): o As abstrações de alto nível são inadequadas para a construção de sistemas persistentes; o A evolução de núcleos monolíticos é extremamente difícil devido à fraca estruturação interior e à natureza monolítica e, o Como todos os módulos do núcleo residem no mesmo espaço de endereçamento, um erro em um módulo pode comprometer o sistema. • Para Bomberger (1992) : o O nanonúcleo inclui todo o código do sistema em modo supervisor; o O nanonúcleo em execução ocupa 100Kb de memória principal; o No caso do KeyKOS, o núcleo é implementado em aproximadamente 20.000 linhas em C e 2000 de código assembly (destas, 1000 linhas são usadas para implementar a troca de contexto); o Persistência de um nível (single-level store); o Adota um espaço de endereços virtuais para cada processo; o Um mecanismo de checkpoint global (system-wide); o Acesso primitivo e limitado a dispositivos de E/S; o Um nanonúcleo baseado em capabilities é a única parte do sistema que interpreta as chaves. Nenhum outro programa tem acesso direto aos bits contidos nas chaves, o que evita a questão de *key forgery*; o O nanonúcleo inclui código que define os objetos primitivos do sistema como suporte à multiprogramação, escalonador primitivo e filtros (*hooks*) para escalonadores mais sofisticados que executam como aplicações; o Interpretação das chaves que esconde a localização do objeto no disco ou em memória principal.	
Desvantagens	
• Este modelo tem como conseqüência uma dificuldade adicional na determinação do limite entre modo supervisor e modo usuário (HULSE e DEARLE, 1998).	
Exemplos	
• Grasshopper (DEARLE e HULSE, 2000; HULSE e DEARLE, 1998);	• KeyKOS (BOMBERGER ,1992).

[28] Segundo DIXON et al. (1989), persistência ortogonal implica que a propriedade de persistência é independente tanto da classe de um objeto como da maneira como ele á usado.

2.4.9 Estruturas Baseadas em Máquinas Virtuais

Uma máquina virtual é uma ilusão de uma máquina real. Ela é criada por um sistema operacional de máquina virtual, a qual faz com que uma máquina real pareça ser várias máquinas reais. Do ponto de vista do usuário, as máquinas virtuais podem parecer semelhantes ao hardware real, ou serem muito diferentes dele. Este conceito foi utilizado na série de sistemas VM/370, da IBM (DEITEL, 1990).

A Tabela 12 apresenta um resumo das características desta abordagem para concepção de núcleo de sistemas operacionais. Uma descrição mais detalhada pode ser obtida na seção 10.2.9(Pág.333).

Tabela 12 - Características de núcleos baseados em máquinas virtuais

Características	
• Implementa em software uma máquina, a qual possui os mesmos recursos e funcionalidades do hardware real (inclusive as instruções privilegiadas) (FOLLIOT, 2000).	
Vantagens	
• Permite que vários sistemas sejam executados sobre um mesmo hardware (CREASY, 1981) ; • Suporta a execução de múltiplos sistemas operacionais simultaneamente sobre o mesmo hardware; • Independência de plataforma; • Pode ser utilizada para análise e depuração de sistemas operacionais e/ou aplicativos (ROSENBLUM et al., 1995, ROSENBLUM et al. 1997); • Pode ser utilizada como ferramenta de simulação de hardware real para fins didáticos (MATTOS e TAVARES, 1999); • Permite compatibilidade com versões antigas de sistemas operacionais comerciais (HAZZAH, 1997); • Sansonnet et al. (1982) relatam a implementação de um processador LISP em hardware (utilizando microprogramação) e destacam os ganhos de performance que esta estratégia permite em relação às técnicas tradicionais de interpretação de máquinas abstratas.	
Desvantagens	
• Tendência em degradar a performance da máquina real - VM/370 (CREASY, 1981); • Latência no atendimento a interrupções pode ser significativa (HAZZAH, 1997, pág 107); • A comunicação entre sistemas operacionais de máquinas virtuais diferentes ocorre somente através de protocolos de comunicação, comprometendo a performance.	
Exemplos	
• VM/370 (DEITEL, 1990; CREASY, 1981); • VmWare (SUGERMAN, VENKITACHALAM e LIM, 2001); • VirtualPC (CONNECTIX, 2001); • Bochs (BOCHS, 2003); • Java Virtual Machine (VENNERS, 1999); • Jalapeño (ALPERN et al., 2000);	• Máquina Pascal Concorrente (BRINCH HANSEN, 1970, BRINCH HANSEN, 1975); • Inferno/Limbo (DORWARD et al, 1997); • XSLTVM (NOVOSELSKY e KARUN, 2000); • Windows 2000 (HAZZAH, 1997); • HEART (KAMIBAYASHI et al., 1982); • Sansonnet et al. (1982).

2.4.10 Estruturas Baseadas no Conceito de Objetos

A estrutura baseada em objetos tem a construção do sistema operacional com base no conceito de objetos, onde os serviços do sistema são implementados como uma coleção de objetos definidos como segmentos protegidos, através dos quais os estados internos são acessados e alterados (ZANCANELLA e NAVAUX, 1993).

A Tabela 13 apresenta um resumo das características desta abordagem para concepção de núcleo de sistemas operacionais. Uma descrição mais detalhada pode ser obtida na seção 10.2.10(Pág.340).

Tabela 13 - Características de núcleos baseados no conceito de objetos

Características	
• Segundo Zancanella e Navaux (1993): o Os serviços do sistema são implementados como uma coleção de objetos, os quais são definidos como segmentos protegidos, através dos quais os estados internos são acessados e alterados; o Geralmente, o núcleo do sistema tem a função de gerenciamento dos objetos, gerenciamento de interações entre objetos e a responsabilidade de proteger os objetos contra acessos indevidos; o Requer uma disciplina para organizar os serviços do sistema operacional como uma coleção de objetos; o Existência de uma thread de controle, de modo que objetos coletivamente possam ser tratados como uma unidade única. • Para Gutiérrez et al. (1997): o Objeto como única abstração; o Objetos autocontidos e com semântica sustentada no modelo de objetos; o Identidade de objetos – cada objeto possui um identificador único.	
Vantagens	
• Segundo Zancanella e Navaux (1993): o Em Muse, cada objeto consiste de: memória local, métodos que acessam a memória local e processadores virtuais que executam os métodos; o O sistema Aurora suporta o conceito de herança dinâmica e trata todos os objetos de maneira uniforme e independente de localização. • Modularidade na representação dos objetos do sistema (BEUCHE et al.,1997); • O conceito de migração de objetos é utilizado para implementar os serviços do sistema operacional e distribuir a carga do sistema entre os processadores onde todos os recursos são representados como abstração do modelo de objetos, incluindo o núcleo do sistema (Aurora - ZANCANELLA e NAVAUX, 1993; Cosmos - NICOL et al., 1989); • Em Cosmos, a imutabilidade dos objetos (objetos assim quecriados não podem ser alterados) permite a produção de um grafo de versões com a história do objeto (NICOL et al., 1989); • Facilidade de emprego da técnica de programação orientada a aspectos - AOP (KICZALES et al., 1997; BEUCHE et al., 1999); • Portabilidade, independência da linguagem e facilidade de implementação (GUTIÉRREZ et al. , 1997).	
Desvantagens	
• Segundo BEUCHE (1998): o Baixa eficiência; o À medida que a complexidade do projeto aumenta, começa a se tornar difícil o processo de seleção dos componentes apropriados; o Este sistema possui 170 classes, as quais podem ser combinadas para produzir 20 membros diferentes de uma família, permitindo a geração de código para quatro tipos diferentes de processadores. Contudo, esta flexibilidade produz mais de 80 conjuntos diferentes de código; o Em determinadas situações, decisões sobre configuração global podem causar efeitos secundários tanto globais como parciais no sistema gerado.	
Exemplos	
• Cosmos (NICOL et al., 1989); • Hydra (WULF et al., 1974); • Chorus (BANINO, 1985); • Amoeba (MULLENDER et al., 1990); • Aurora (ZANCANELLA e NAVAUX, 1993);	• PURE (BEUCHE et al., 1999); • Oviedo3 (GUTIÉRREZ et al., 1997); • Muse (FUJINAMI, 1991 apud ZANCANELLA e NAVAUX, 1993); • Tunes (GUTIÉRREZ et al. ,1997); • Merlin (GUTIÉRREZ et al., 1997).

2.4.11 Estruturas Baseadas em Reflexão Computacional

Separação de interesses é uma estratégia que procura separar os requisitos funcionais dos requisitos operacionais de uma aplicação. A estratégia de reflexão computacional é uma forma de separação de interesses onde os aspectos funcionais e não funcionais são separados através do uso de objetos base e meta-objetos.

Segundo Sztajnberg (1999), os objetos–base implementam em seus métodos a funcionalidade da aplicação, enquanto os meta-objetos implementam os procedimentos de controle que determinam o comportamento dos objetos base a eles associados. Os objetos base podem não estar cientes da existência dos meta-objetos, não sendo obrigatório que cada objeto possua um meta-objeto associado. As chamadas aos métodos do objeto base são desviadas de modo a ativar métodos na meta-classe, permitindo, dessa forma, a supervisão e a modificação do comportamento do código funcional residente no objeto base. Além disso, também é possível adicionar funcionalidades ao objeto base. A execução do objeto base é suspensa até que o meta-objeto termine a sua execução, o que permite ao meta-objeto ter completo controle das atividades do objeto base.

Tabela 14 - Características de núcleos baseados em reflexão computacional

Características	
• Sistemas capazes de acessar sua própria descrição e alterá-la de modo a alterar seu próprio comportamento (ZANCANELLA e NAVAUX, 1993; CAZZOLA, 1998); • Os objetos–base implementam em seus métodos a funcionalidade da aplicação, enquanto os meta-objetos implementam os procedimentos de controle que determinam o comportamento dos objetos base a eles associados (SZTAJNBERG, 1999); • Informações reflexivas constituem-se em meta dados sobre o sistema. Estes meta-dados podem ser sobre as aplicações, bem como sobre a performance e outras propriedades do sistema (STANKOVIC, 1998); • O sistema AURORA utiliza o modelo estrutural de Apertos, o qual implementa a idéia de que o nível de objetos e o nível de abstração das linguagens possam ser separadamente descritos e implementados dentro da mesma estrutura (ZANCANELLA e NAVAUX, 1993); • Aurora utiliza o conceito de separação entre objetos e meta-objetos, onde o objeto representa um repositório de dados, enquanto o meta-objeto define não somente a semântica de seu comportamento, mas a abstração da classe (ZANCANELLA e NAVAUX, 1993); • Oviedo3 utiliza uma arquitetura reflexiva como um mecanismo de colaboração entre a máquina e o sistema operacional (GUTIÉRREZ et al., 1997); • Em MUSE um objeto é uma simples abstração de um recurso computacional do sistema e pertence a um nível de abstração mais elevado, denominado de meta-espaço. O meta-espaço é que provê um ambiente de execução (FUJINAMI, 1991 apud ZANCANELLA e NAVAUX, 1993, pág 31); • Schubert (1997) descreve o projeto CHEOPS onde são aplicados conceitos de orientação a objetos e reflexão computacional para suportar adaptabilidade dinâmica com uma granularidade bastante fina.	
Vantagens	
• Vários modelos de reflexão computacional (meta-classes, meta-objetos, reificação de mensagens e reificação de canal), têm sido desenvolvidos e cada modelo tem diferentes facilidades ausentes dos demais modelos (CAZZOLA, 1998); • Esta estratégia facilita a construção de facilidades básicas de um sistema operacional, como persistência ortogonal, distribuição de objetos, concorrência e segurança baseada em capabilities (GUTIÉRREZ et al., 1997; • O conceito de processadores virtuais distingue os objetos MUSE de outras estruturas baseadas em objetos, pois permite que cada objeto tenha seu próprio ambiente de execução e permite a introdução do conceito de computação reflexiva (ZANCANELLA e NAVAUX, 1993).	
Desvantagens	
• O tipo de reflexão suportada pelo modelo reflexivo especifica que aspecto do sistema pode ser monitorado e alterado pelas meta-entidades. (CAZZOLA, 1998) .	
Exemplos	
• Apertos (YOKOTE, 1992); • Gowing e Cahill (1995); • Kon (2000); • Aurora (ZANCANELLA e NAVAUX,1993);	• Oviedo3 (GUTIÉRREZ et al., 1997); • MUSE (FUJINAMI, 1991 apud ZANCANELLA e NAVAUX, 1993,pág 31); • Mitchell et al. (1997); • CHEOPS (SCHUBERT, 1997).

A Tabela 14 apresenta um resumo das características desta abordagem para concepção de núcleo de sistemas operacionais. Uma descrição mais detalhada pode ser obtida na seção 10.2.11(Pág.343).

2.4.12 Estruturas Baseadas em Conhecimento

As estruturas baseadas em conhecimento têm como característica a existência de uma base de conhecimento, a partir da qual se propõe a criação de um suporte mais inteligente para as aplicações e de melhores ambientes para o usuário.

Segundo Li et al. (1995) a idéia de um sistema operacional baseado em conhecimento não é nova.

> *"Fleish e Blair já haviam reconhecido que a solução para os problemas detectados em sistemas operacionais (os quais serão discutidos nos capítulos seguintes) poderia ser viabilizada através do que eles chamaram, respectivamente, de: Sistema Operacional Inteligente e Sistema Operacional Baseado em Conhecimento." Li et al. (1995).*

Além disso, complementam Li et al. (1995), outros sistemas experimentais também seguiram esta direção, como: **UC** (VILENSKY,ARENS e CHIN, 1984) , que propôs melhorar a interface de interação com o sistema operacional; **PIMOS** (CHIKAYAMA, 1988), que propôs uma máquina de inferência paralela, e as propostas de Pasquale (1988) e Korner (1990) no sentido de implementar mecanismos de gerenciamento inteligente de sistemas distribuídos.

Segundo Zancanella e Navaux (1993), o modelo de objetos é utilizado como ponto de partida para o desenvolvimento da estrutura baseada em conhecimento, de modo que as informações semânticas mantidas na base de dados, tidas como a chave para prover a inteligência e a integração requerida pelo sistema operacional, dizem respeito à caracterização dos objetos e ao inter-relacionamento entre os mesmos.

A Tabela 15 apresenta um resumo das características desta abordagem para concepção de núcleo de sistemas operacionais. Uma descrição mais detalhada pode ser obtida na seção 10.2.12(Pág.347).

Tabela 15 - Características de núcleos baseados em conhecimento

Características
• Estruturas baseadas em conhecimento têm como característica a existência de uma base de conhecimento, a partir da qual propõe-se a criação de um suporte mais inteligente para as aplicações e de melhores ambientes para o usuário (LI et al., 1995); • O modelo proposto por Nicol (1987), assemelha-se ao modelo de redes semânticas de objetos, onde o conhecimento é representado de forma explícita e concisa. Novos fatos sobre objetos podem facilmente ser adicionados à rede semântica e todo o relacionamento para um dado objeto pode ser determinado sem uma pesquisa extensiva. • Em Genera (1990) são relacionadas as seguintes características: o Todas as atividades são constituídas por funções Lisp, baseadas em uma biblioteca genérica; o Abaixo deste nível de software há o hardware, de tal forma que o usuário (ou programador) não tem acesso ao mesmo; o Genera possui uma forma de registro histórico de quase tudo o que acontece no sistema (histórico de comandos, resultados, janelas); o Em Genera, o ambiente todo é chamado de *world* e é salvo em disco de tal forma a estar sempre pronto para ser inicializado. • O sistema KZ2 apresenta as seguintes características principais (Li et al., 1995) : o Multi-computadores: ele suporta computação distribuída e processamento paralelo baseado em um sistema com massivo número de computadores conectados (fracamente ou fortemente acoplados) através de redes que cooperativamente suportam a execução de aplicações; o Processamento de conhecimento: utiliza conhecimento para gerenciar os recursos do sistema e para controlar a comunicação homem-máquina. Além disso, possui um mecanismo para processamento eficiente de conhecimento disponível tanto para o sistema como para as aplicações; o Inteligência: utiliza técnicas de IA para resolver os problemas (particularmente as indeterminações encontradas em gerenciamento de recursos de sistemas computacionais maciçamente paralelos) e apresentar um comportamento mais inteligente. A capacidade de aprendizagem do sistema pode também melhorar suas funções e a própria performance; o Orientado a novatos: o processamento paralelo/distribuído que, em algum grau, é difícil tanto para usuários novatos como para especialistas, pode ser transparente. Além disso, uma interface em linguagem natural permite que o usuário não necessite estar familiarizado com as linguagens de comando dos sistemas operacionais tradicionais; o O mecanismo de inferência paralelo adota Grafos de Expressões de Execução e Árvores de Execução Paralela para decompor um esquema de controle. Além disso, utiliza um mecanismo de sincronização baseado em produtor/consumidor para controlar a utilização de variáveis compartilhadas no sentido de evitar a produção de resultados inconsistentes durante a execução de esquemas de controle.
Vantagens
• Conforme citado em Genera (1990): o O sistema possui uma biblioteca de funções genéricas, algumas das quais são implementadas em componentes de hardware para garantir a eficiência do sistema; o Através da utilização de linguagens orientadas a objetos (Flavors, Clos) o sistema permite a utilização dos conceitos de encapsulamento, herança, etc., o que permite a criação de estruturas abstratas de forma relativamente simples; o Gerenciamento automático de memória; o O garbage collector é implementado através da utilização de hardware dedicado; o Ligação dinâmica: Em Genera, é necessário somente o passo de compilação. Uma vez que o código é verificado, ele é automaticamente incluído na biblioteca do sistema; o Integração ao nível de dados: todas as funções e dados existem em memória virtual, a qual é compartilhada por todos os processos; o Arquitetura aberta: todas as partes do sistema são disponíveis para inspeção, reuso, extensão ou substitUição; o O sistema é construído de forma a permitir adequações dos usuários em função de preferências e características individuais; o Auto-revelação: as informações sobre o funcionamento interno do sistema estão sempre disponíveis. • Segundo Englemore e Morgan (1988), arquiteturas do tipo *blackboard* apresentam as seguintes características: o Modularidade na representação de conhecimento e técnicas de inferência; o Liberdade em raciocinar sobre suas próprias estruturas de controle; o Fornecem mecanismos para implementar uma variedade de estratégias de raciocínio e representação de conhecimento; • As abstrações fornecidas pela abordagem *blackboard* (DOSBART) são adequadas para tratar com muitas das questões relativas ao controle em tempo real (Larner, 1990): o Controle em tempo real significa que o estado do sistema é ativamente controlado de modo a proceder de acordo com uma seqüência de valores desejados (ou permitidos) durante o tempo em que o próprio sistema está executando; o Além disso, as ações de controle são tomadas para garantir que estas seqüências sejam decididas no mesmo período de execução. Ou seja, o sistema adapta-se dinamicamente às alterações percebidas no ambiente controlado; o Um aspecto relevante nesta proposta é o conceito de *knowledge activity (KA)*. Uma *KA* utiliza um módulo provador de teoremas para criar novos objetos, ou para implementar qualquer tipo de estratégia de solução ou controle desejados; o Este enfoque de um lado aproveita as vantagens da técnica de *blackboard* e, de outro, os benefícios da distribuição de capacidade de processamento. • O sistema KZ2 caracteriza-se como a nova geração de sistemas operacionais capazes de gerenciar recursos de sistemas maciçamente paralelos, apresentar uma interface amigável de comunicação homem-máquina e controlar a execução de programas baseando-se em processamento de conhecimento e processamento paralelo (LI et al.,1995).

Desvantagens	
• Conforme citado em Genera (1990): o Como não há uma distinção formal entre o sistema e o usuário, o segundo pode sentir-se tentado a resolver um problema utilizando primitivas ao invés de utilizar as facilidades mais apropriadas ao problema; o O fato de que o usuário pode usar qualquer subconjunto de facilidades faz com que aumente o tamanho e a complexidade do conjunto de documentação disponível. Um outro aspecto importante é que a documentação é sobre funções Lisp, o que demanda um conhecimento mais aprofundado do usuário; o À medida que o sistema evolui, surgem questões de compatibilidade entre suas novas versões e o código gerado pelo usuário, baseado em bibliotecas prescritas. • No enfoque *blackboard* devem ser considerados aspectos relativos à sincronização, replicação de dados, consistência, protocolos de comunicação e algoritmos distribuídos (LARNER, 1990).	
Exemplos	
• Genera (GENERA, 1990); • UC (VILENSKY, ARENS e CHIN, 1984 apud LI et al., 1995); • PIMOS (CHIKAYAMA, 1988 apud LI et al., 1995); • DOSBART (LARNER, 1990);	• KZ2 (LI et al., 1995); • KZ10 (LI , 2003); • KZ1 (LI , 2003).

2.4.13 Estruturas Baseadas em Agentes

Segundo Kalakota e Whinston (1996), "a metáfora de agentes tem suas raízes em trabalhos realizados no início dos anos 60. A área de Inteligência Artificial utilizou o termo para referir-se a programas que atuam em favor de seus usuários".

Booker et al. (1999) complementa que:

> *"Em 1995, Wooldridge e Jennings publicaram o livro: "Agentes Inteligentes: Teoria e Pratica" (1995), o qual provavelmente contém a primeira definição compreensível de agentes inteligentes, suas propriedades e comportamentos. Ao mesmo tempo, outros modelos não diretamente relacionados à computação estavam sendo desenvolvidos, tais como o projeto Ubiquitous Computing do Palo Alto Research Center".*

O conceito mais comumente adotado define agente como sendo um sistema que percebe e age em ambientes dinâmicos e imprevisíveis, que coordena capacidades de interoperação entre componentes complementares e que executa serviços úteis com um alto grau de autonomia. (BÁRBARA HAYES-ROTH apud BOOKER et al.,1999).

A Tabela 16 apresenta um resumo das características desta abordagem para concepção de núcleo de sistemas operacionais. Uma descrição mais detalhada pode ser obtida na seção 10.2.13(Pág.356).

Tabela 16 - Características de núcleos baseados em agentes

Características	
• O núcleo do sistema operacional é composto de um conjunto mínimo de recursos que permite a gerência e a intercomunicação entre pequenos módulos autônomos denominados agentes; • Diferentemente dos núcleos tradicionais onde todos os módulos compartilham o mesmo espaço de memória, os agentes em ABOS executam como processos separados. Isto permite modificar a estrutura do sistema sem a necessidade de recompilar ou mesmo reinicializar o sistema operacional (SVAHNBERG, 1998); • ABOS é organizado como um conjunto de níveis ao redor de um núcleo (SVAHNBERG, 1998); • AGENTOS (CHEN, 2000) caracteriza-se como um sistema operacional virtual que suporta o desenvolvimento de aplicações orientadas a objetos e distribuídas através da utilização de agentes; • Agentos é um agente-hospedeiro (*agent-host*) que provê suporte para a execução de agentes implementados em Java. O sistema é composto por um pequeno núcleo que estabelece um modelo de execução para agentes, um mecanismo de comunicação baseado em eventos com transparência de rede e um modelo para acesso aos serviços disponíveis (CHEN, 2000).	
Vantagens	
• Segundo Svahnberg (1998): o Em ABOS os agentes executam em espaços de memória independentes; o Com isto, pretende-se atingir um nível de extensibilidade e flexibilidade bastante razoável; o Como os agentes são autônomos e podem adaptar-se com o tempo, é possível a criação de um sistema operacional mais flexível utilizando esta tecnologia; o Em ABOS é possível modificar a estrutura do sistema sem a necessidade de recompilar ou mesmo reinicializar o sistema operacional; o Em ABOS o nível dos serviços aumenta na medida em que há uma transposição de nível mais distante do núcleo; o O núcleo ABOS suporta as funções básicas de gerenciamento de processos, memória e comunicação entre processos; • Seguindo a filosofia adotada em linguagens funcionais, uma vez informada uma meta pelo usuário o sistema constrói uma seqüência de ativação de primitivas a serem usadas e executa-as tomando decisões intermediárias baseando-se no contexto e nas condições existentes (ETZIONI et al., 1995).	
Desvantagens	
• Não localizadas na literatura consultada.	
Exemplos	
• ABOS (SVAHNBERG, 1998);	• AGENTOS (CHEN, 2000).

2.4.14 Estruturas Híbridas

A estratégia adotada pela Microsoft no projeto do sistema operacional Windows 2000 envolve o conceito de arquitetura híbrida (AHL, 2001). Esta arquitetura utiliza conceitos do modelo cliente-servidor, em camadas, orientada a objetos e suporta multiprocessamento simétrico (Figura 91Pág360).

A Tabela 17 apresenta um resumo das características desta abordagem para concepção de núcleo de sistemas operacionais. Uma descrição mais detalhada pode ser obtida na seção 10.2.14(Pág.359).

Tabela 17- Características de núcleos híbridos

Características
• A arquitetura do sistema operacional Windows 2000 envolve estruturas cliente-servidor, em camadas, orientada a objetos e suporta multiprocessamento simétrico (AHL, 2001); • Segundo Oney (1999), todo o conhecimento do sistema operacional sobre como o computador está conectado aos vários dispositivos físicos repousa sobre o HAL o qual detém informações sobre como as interrupções funcionam em uma plataforma específica, como são implementados os *spin-locks* e como deve ser endereçado o espaço de entrada/saída; • Ao invés de acessar o hardware diretamente, os *drivers WDM* fazem chamadas a primitivas do HAL para solicitar os serviços necessários para efetivamente realizar o acesso aos periféricos (ONEY, 1999).
Vantagens
• Usa um HAL para virtualizar o hardware e facilitar o porte para outras plataformas (AHL, 2001); • Ao invés de acessar o hardware diretamente, os *drivers* WDM fazem chamadas a primitivas do HAL para solicitar os serviços necessários para efetivamente realizar o acesso aos periféricos. Esta solução permite o desenvolvimento de *device drivers* independentes de plataforma e de barramento (*bus*) (ONEY, 1999).
Desvantagens
• Latência de interrupção decorrente da virtualização do hardware (AHL, 2001); • Overhead para comunicação entre camadas complexa (AHL, 2001; DUMON, 1997).
Exemplos
• Windows 2000 (AHL, 2001; DUMON, 1997).

A partir da revisão sobre as diferentes estruturas organizacionais dos núcleos de sistemas operacionais, a próxima seção apresenta as técnicas geralmente utilizadas para obtenção de dados do núcleo de um sistema operacional.

2.5 Técnicas para obtenção de dados do núcleo

Segundo Skjellum et al. (2000), existem várias técnicas sendo utilizadas pelos desenvolvedores para obter informações sobre o estado dos sistemas operacionais. Estas técnicas podem ser resumidamente descritas da seguinte forma (Figura 15,pág.85), conforme apresentado em Skjellum et al. (2000):

- **Através do acesso direto a estruturas de dados do núcleo:** Neste enfoque, os dados são obtidos através de operações de leitura direta em endereços conhecidos (usando uma interface como **/dev/kmem**). Este enfoque torna a tarefa de obter dados do núcleo dependente de versões do núcleo, o que implica em que, a cada alteração no mesmo, podem ocorrer falhas no sistema de aquisição de dados. Para evitar este problema, os projetos de núcleos mais modernos disponibilizam uma interface (API) para acesso aos serviços de informações sobre performance do sistema. Uma característica deste enfoque é permitir alta velocidade na aquisição de informações.
- **Através de *device drivers* específicos:** algumas ferramentas recentes (ASTALOS e HLUCHY, 2000) tentam viabilizar o acesso rápido às informações do núcleo valendo-se de um *device driver* especial que acessa diretamente as informações, porém mantém uma API fixa de acesso. A vantagem desta estratégia, em relação à anterior, é a independência do núcleo. Ou seja, alterações nas estruturas de dados do núcleo não alteram as aplicações, visto que o acesso dá-se via API do *device driver*. Contudo, a cada alteração do núcleo o *device driver* também precisa ser atualizado. Tomando-se como exemplo o ambiente de desenvolvimento **Linux**, onde a taxa de atualizações do núcleo é elevada, pode-se considerar que a taxa de

atualizações dos *device drivers* especializados acarreta um esforço extra, em se adotando esta solução. O apêndice 5 (seção 10.4,pág.365) apresenta um extrato dos resultados de testes realizados utilizando-se esta técnica para obtenção de dados do núcleo do sistema operacional Windows 98 durante o processo de estabelecimento de conexão com a internet através do serviço de dial-up daquele sistema.

- **Através de uma interface com o sistema de arquivos:** muitos sistemas **Unix**, incluindo o **Linux**, possuem uma interface para acesso às informações do núcleo na forma de um sistema de arquivos virtual (*Virtual File System* - VFS). As informações do núcleo, como estatísticas de CPU, processos e rede, aparecem para os usuários como arquivos num diretório especial denominado **/proc**. Pode-se citar como vantagens desta abordagem a portabilidade e independência do núcleo. O local das informações encontradas neste diretório tende a ser o mesmo para todas as versões do núcleo. Além disso, a velocidade de acesso é comparável àquela obtida pelo acesso direto às estruturas do núcleo (1ª abordagem), dado que o diretório **/proc** é mapeado em memória.
- **Através de APIs específicas:** certos sistemas operacionais, como o Microsoft Windows, disponibilizam APIs na forma de um WMI (*Windows Management and Instrumentation*) para fornecer informações sobre o sistema. A vantagem de utilizar tais APIsé a facilidade de programação e a independência do núcleo. A API WMI é implementada como objetos COM, que podem ser chamados a partir de praticamente todas as linguagens de scripts e de programação utilizadas no ambiente Windows. Há APIs similares disponíveis para o ambiente **Unix**, mas nenhuma implementada de forma tão consistente como a WMI.
- **Através de interface com o sistema operacional:** Algumas ferramentas baseadas em **Unix** usam o resultado gerado pela execução de comandos tradicionais do **Unix**, como ***uptime*** ou os, para obter informações sobre performance. Isto pode garantir portabilidade entre famílias de sistemas **Unix**. Contudo, este é um enfoque ineficiente, uma vez que compromete a performance do sistema cada vez que os comandos são executados. O apêndice 6 (seção 10.5,pág.368) apresenta um extrato dos resultados de testes realizados utilizando o comando sar para obtenção de dados do núcleo do sistema operacional **AIX**.

Ainda segundo Skjellum et al. (2000), o monitoramento de performance deve ser uma atividade que cause a menor intrusão possível ao sistema. Podem ser consideradas como fontes de intrusão a leitura de informações do sistema e a execução de notificação de eventos. A leitura de informações utilizando o diretório **/proc**, geralmente causa baixo impacto no sistema, uma vez que esta atividade requer somente acessos à memória. Contudo, em Astalos e Hluchy (2000), o nível de intrusão foi reduzido ainda mais através de um *driver* especial que coleta informações do sistema e as envia diretamente para o módulo de consulta (*probing*). O balanceamento entre baixa intrusão e portabilidade é a próxima questão a ser considerada. Por exemplo: para a execução de um mecanismo de notificação de eventos, o nível de intrusão depende da complexidade das regras, numero de eventos, freqüência de consultas (*checking*) e eficiência na implementação. Neste caso, a única otimização possível é selecionar um número apropriado de eventos e o melhor intervalo de verificação.

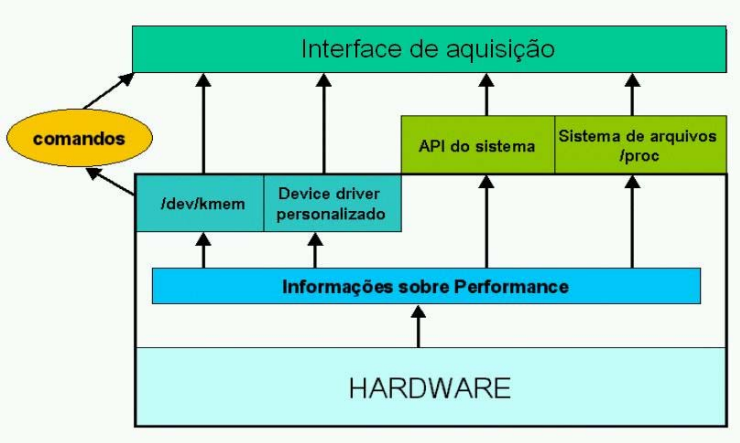


Figura 15 - Formas de obtenção de informações do sistema operacional

FONTE: Adaptado de Skjellum et al. (2000).

A próxima seção sintetiza o modelo evolutivo dos sistemas operacionais segundo a visão do autor do presente trabalho.

2.6 O modelo evolutivo dos sistemas operacionais

Até muito pouco tempo, a grande preocupação dos projetistas de sistemas operacionais estava centrada no conceito de interligação entre equipamentos, com uma grande parcela dos investimentos sendo direcionada para a área hoje denominada de "redes locais". Segundo Deitel (1990), a evolução deste tipo de aplicação passou, logo a seguir, pela introdução do conceito de processamento em lotes (*batch*) e, após, pelo conceito de tempo compartilhado ou *time-sharing* (CTSS) (CORBATÓ, MERWIN-DAGGETT e DALEY, 1962), que culminou com o surgimento do sistema **Multics** (DALEY e DENNIS, 1968, BENSOUSSAN,CLIGEN e DALEY, 1972), que pretendia suportar um grande número de usuários simultâneo. Apesar de não ter sido um sucesso comercial, esse sistema influenciou o desenvolvimento de vários sistemas posteriores.

Analisando-se o modelo evolutivo dos sistemas operacionais a partir de uma análise das seções anteriores, ou seja, sob a ótica de requisitos funcionais, também é possível identificar um movimento cíclico semelhante àquele discutido anteriormente na seção 2.3. Esta análise é apresentada a seguir.

Segundo Deitel (1990), nos primórdios da computação, logo após o advento dos montadores e linguagens simbólicas, os programadores tinham completo acesso ao

hardware. A complexidade de controle nas operações de entrada/saída conduziu ao desenvolvimento das bibliotecas de E/S (Figura 16).

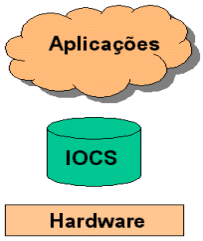


Figura 16 - Fase 0: primeiro estágio evolutivo dos sistemas operacionais.

A partir deste pequeno, mas importante nível de abstração, os programadores passaram a ser mais produtivos. Afinal, não era mais necessário preocupar-se com os detalhes de mais baixo nível. Mais tarde surge o conceito de sistema operacional e aplicações mais complexas começam a ser construídas, demandando novas necessidades, como a interligação entre computadores.

Nas primeiras implementações de redes locais, o suporte era fornecido por aplicações que rodavam acima do sistema operacional, o que tornava o sistema ineficiente e instável. Como essa facilidade não era suportada pelos sistemas, tornou-se necessária a construção de bibliotecas específicas para implementar os mais variados protocolos de comunicação.

Surgiram então vários fornecedores de protocolos de comunicação no mercado e, novamente, aplicações mais complexas foram sendo construídas, gerando novas demandas (Figura 17). Vários fornecedores de soluções surgiram no mercado nesta época.

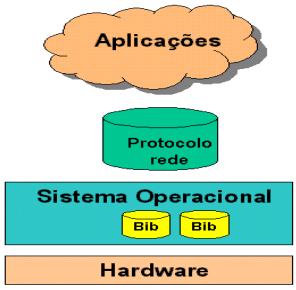


Figura 17 - Fase 1: Protocolos implementados como bibliotecas externas ao sistema

Entretanto, à medida que o tempo foi passando, os projetistas de sistemas operacionais agregaram ao núcleo do sistema as tarefas de suporte à rede, tornando-as uma facilidade "natural" do sistema.

Na fase seguinte, os protocolos, antes executados como extensões, passaram a fazer parte dos sistemas (Figura 18). Isso fez com que os programadores não precisassem mais se preocupar com detalhes de protocolos, visto que a API do sistema operacional passou a prover abstrações que permitiam a eles acessar tanto os recursos locais como os recursos remotos, através de operações tradicionais de leitura/escrita.

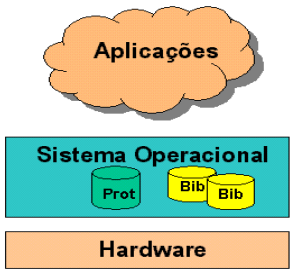


Figura 18 - Fase 2: protocolos incorporados ao sistema operacional

Esta realidade verificou-se também com as interfaces de interação entre hardware e o usuário. Nos últimos 10 anos, observou-se um direcionamento dos investimentos no sentido de se abandonar as antigas interfaces de comunicação orientadas a caractere, para a adoção cada vez maior de suporte a interfaces gráficas. Em paralelo ao desenvolvimento destas interfaces gráficas, um novo periférico passou a compor "naturalmente" o contexto de um ambiente de computação: o mouse. Da mesma forma, as primeiras soluções incorporavam uma aplicação que executava sobre o sistema operacional e que fornecia o suporte necessário para a utilização deste recurso de hardware. Não se admite, atualmente, que um sistema operacional não suporte de forma natural (ou nativa) equipamentos deste tipo.

A partir do momento em que se disponibilizaram para o usuário interfaces que permitiam a utilização de gráficos, novas necessidades e produtos surgiram no sentido de agregar facilidades de sons e manipulação de imagens. Novamente seguiram-se as antigas práticas de se agregar aplicações para suportar estas facilidades e, num momento seguinte, os sistemas operacionais passaram a suportar originalmente estes recursos. Mais uma vez, aplicações novas e mais complexas começaram a ser construídas. Então surgiu a Internet e, com ela, novas demandas.

Num primeiro momento, foram construídas aplicações que permitiam a navegação pela rede através de aplicações chamadas navegadores ou *browsers* (Figura 19).

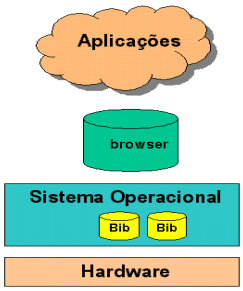


Figura 19 - Fase 1: Navegadores desenvolvidos como bibliotecas externas

O movimento seguinte foi o de incorporar os recursos de navegação *(browsing)* no sistema operacional para criar um tal nível de abstração que, tanto os recursos da máquina local, como os recursos remotos, passam a ser tratados da mesma forma (exceto pelo atraso imposto pelos canais de comunicação). Tomando-se por base o modelo acima, observa-se um movimento cíclico no processo evolutivo. Em princípio, os recursos disponibilizados pelo sistema operacional permitem a construção de aplicações capazes de gerar novas demandas, que são atendidas através de soluções proprietárias. Numa geração seguinte, essas soluções são incorporadas ao sistema, gerando novo nível de abstrações (Figura 20). E o ciclo se repete.

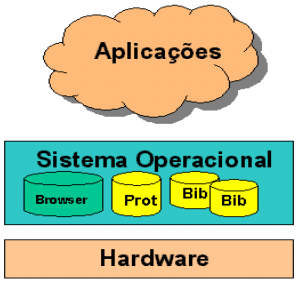


Figura 20 - Fase 2: Navegadores incorporados ao sistema operacional

Seguindo-se este caminho evolutivo, as pesquisas indicam que, possivelmente, o próximo modelo a ser agregado ao contexto dos computadores está associado à utilização de soluções de interface de voz, em substituição dos recursos de hardware, teclado e mouse. As aplicações atualmente existentes ainda são incipientes e não têm suporte de sistema operacional. Na verdade, os sistemas operacionais, em sua grande maioria, ainda

tratam esta facilidade como um recurso multimídia e, assim, classificam-na como dado, fornecendo somente suporte para os aspectos de armazenamento e transferência de/para os periféricos associados (placas de som e sintetizadores). Assim, projeta-se um movimento imediato no sentido de construirem-se aplicações específicas para reconhecimento de voz (como o ViaVoice da IBM), e uma posterior absorção desta facilidade pelo sistema operacional.

Com base nas tendências de pesquisa, descritas anteriormente, e na observação de que há uma demanda crescente por soluções de extração e manipulação de conhecimento (HAN e KANEBER, 2000; ZAIANE e JIAWEI, 1995), estima-se que outro possível movimento de absorção de soluções proprietárias ocorrerá sob a forma da incorporação, pelo sistema operacional, das funções de gerência de conhecimento. O uso cada vez maior destas técnicas nas aplicações tem ocorrido em função de novas demandas que a atual plataforma computacional disponibiliza.

Zaiane e Jiawei (1995) afirmam que:

> *"(...) estamos vivendo num período era denominado 'Era da Informação'. Isso porque acreditamos que a informação conduz ao poder e ao sucesso e graças aos avanços tecnológicos, como computadores ou satélites, estamos coletando uma tremenda quantidade de informação. Confrontados com essa enorme quantidade de dados, nós estamos criando novas necessidades para auxiliar-nos a tomar melhores decisões gerenciais. (...)".*

Dieng et al. (1998), ao pesquisarem técnicas, métodos e ferramentas que auxiliem a gerenciar o conhecimento corporativo, afirmam:

> *"(...) há um interesse crescente na capitalização do conhecimento (tanto teórico quanto prático) de grupos de pessoas na organização. Grupos, esses, possivelmente dispersos geograficamente. Portanto, há a necessidade de ferramentas que possam viabilizar a integração de recursos e conhecimento (know-how) corporativo. (...)".*

Este fato tem desencadeado um processo de produção de bibliotecas proprietárias para suporte a algoritmos de redes neurais, lógica difusa, raciocínio baseados em casos, sistemas especialistas e algoritmos genéticos, os quais são embutidos nas aplicações de forma transparente. Portanto, pode-se antecipar que um possível próximo movimento deverá ser o da absorção desta tecnologia pelos sistemas operacionais (Figura 21).

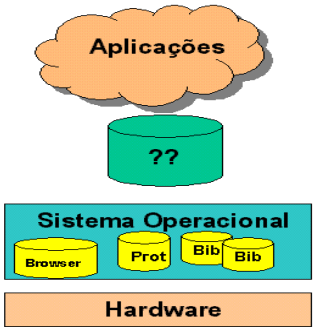


Figura 21 - Próximas demandas

É importante destacar, ainda, que o ciclo evolutivo acima apresentado não descarta as tecnologias que a fase anterior produziu, mas procura agregar num todo uniforme, e pelo menos teoricamente consistente, o conjunto de soluções que estavam anteriormente dispersos. Dessa forma, é criado um nível mais amplo de abstração, que remete ao ciclo evolutivo através da geração de novas aplicações e demandas. Exemplos de *frameworks* híbridos são descritos em Bridges e Vaughn (2000) e Leckie (1995).

2.7 Considerações Finais

Algumas conclusões parciais podem ser inferidas a esta altura. Elas ajudarão na realização dos objetivos apresentados na introdução do capítulo.

Este capítulo apresentou uma revisão das diversas abordagens de construção de núcleos de sistemas operacionais. O levantamento realizado permitiu a identificação dos principais modelos, a partir dos quais é possível classificar a arquitetura de um núcleo de sistema operacional. Como visto, cada abordagem possui seus pontos positivos e negativos.Estas informações foram obtidas a partir do *survey* apresentado na seção 10.2(Pág.311). A Tabela 18 apresenta um resumo das principais características identificadas na forma de requisitos funcionais e não funcionais de projetos de sistemas operacionais.

Tabela 18 - Requisitos funcionais e não funcionais de projetos de sistemas operacionais

Requisitos Não Funcionais de Projetos de Sistemas Operacionais

• Estrutura organizacional (monolítico, camadas, etc);
• Performance do núcleo/serviços do sistema;
• Modelo de aplicação com código compilado estático com suporte a:
  o *Late binding;*
  o Código interpretado*;*
  o Pesquisas em paralelismo de instruções (*pre-fetch*).
• Políticas de segurança através de permissões, restrição de acesso, *capabilities*;
• Suporte a primitivas de sincronização e intercomunicação de processos (*threads*);
• Suporte a uma ou mais políticas de escalonamento;
• Suporte a um ou mais estratégias de gerenciamento de memória;
• Suporte a tratamento de exceções;
• Suporte a gerência de meios de armazenamento;
• Interface do sistema com as aplicações através de APIs.

Requisitos Funcionais de Projetos de Sistemas Operacionais
• Facilidade de extensão dos serviços do sistema para permitir a adaptação por parte das aplicações; • Suporte à multitarefa; • Compatibilidade (ou não) com versões anteriores; • Facilidades de intercomunicação e interligação com outros equipamentos; • Interfaces de interação com o usuário através de interfaces gráficas (e mais recentemente através de interação por voz); • Política de segurança; • Controle de recursos; • Robustez; • Administração do sistema.

A partir desta análise verificou-se que a tendência atual segue na direção da construção de pequenos núcleos (Figura 22), onde os programadores de aplicações são "privilegiados" com níveis de flexibilidade cada vez maiores.

Segundo Dearle e Hulse (2000),

> *"Há uma clara separação entre mecanismos e políticas e, sempre que possível, as abstrações são mínimas. Além disso, elas procuram imitar a funcionalidade do hardware, contrariando o esforço do passado orientado por altos níveis de abstrações".*

Contudo, a construção de núcleos menores, ao mesmo tempo que minimiza a responsabilidade dos projetistas de núcleos de sistema operacional e, conseqüentemente, potencializa os aspectos de qualidade, robustez e eficiência (discutidos na seção 2.2,Pág.55), maximiza os problemas dos desenvolvedores das camadas superiores, uma vez que exige dos mesmos um grau de conhecimento maior sobre as abstrações de baixo nível disponibilizadas. Assim sendo, neste novo nível exportado, ressurgem os problemas de qualidade, robustez e eficiência que, no passado, deram origem a novos projetos de núcleos de sistemas operacionais.O capítulo encerra com uma discussão sobre o modelo cíclico de evolução que os sistemas operacionais têm apresentado, o qual assemelha-se ao modelo de evolução de hardware discutido na seção 2.3(Pág.61).

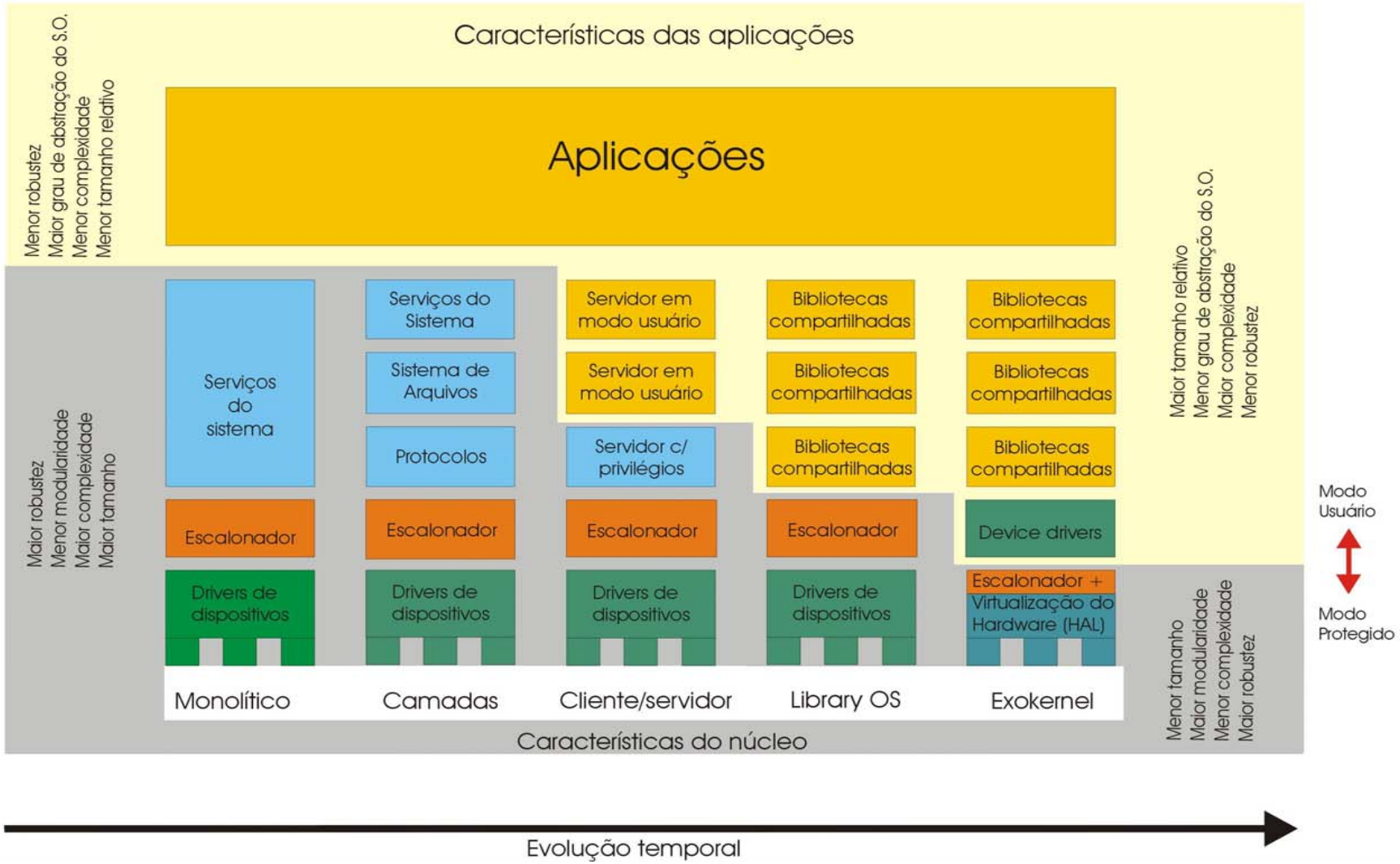


Figura 22 - Evolução dos modelos de núcleos de sistemas operacionais

O próximo capítulo identifica os requisitos que as novas demandas introduzidas com as pesquisas em computação ubíqua, teletrabalho e comércio eletrônico, deverão impor aos projetos de sistemas operacionais. E o capítulo se lhe segue analisa as abordagens de tecnologia de informação que estão sendo consideradas para fazer frente àquelas demandas.

3 NOVAS DEMANDAS: COMPUTAÇÃO UBÍQUA, TELETRABALHO e COMÉRCIO ELETRÔNICO

3.1 Introdução

Uma vez que os vários enfoques de construção de núcleos de sistemas operacionais foram abordados, o próximo passo foi identificar as demandas que deverão disparar uma nova fase no ciclo evolutivo apresentado na seção 2.6(Pág.85).

O objetivo deste capítulo é apresentar uma revisão sobre as tendências das pesquisas em computação ubíqua e os requisitos que as aplicações deverão apresentar para operar neste contexto.

3.2 Tendências em Computação

A computação ubíqua caracteriza-se pela ampliação da capacidade computacional dos ambientes físicos, permitindo que os usuários utilizem os recursos disponíveis de forma transparente e amigável. Estes recursos envolvem os processadores propriamente ditos, mas também incluem uma série de outros equipamentos que hoje requerem atenção e configuração manuais, como projetores e controle de iluminação ambiental.

Esta proposta evoluiu a partir de pesquisas em computação móvel (*mobile computing*) ou computação nômade (*nomadic computing)* (GRIMM et al., 2001) e também tem sido referida na literatura como *ubiquitous computing* (ABOWD,1999; WEISER, 1991; WEISER e BROWN, 1998), espaços ativos *(*WANG e GARLAN, 2000) ou, ainda, espaços inteligentes (NEWMAN, INGRAM e HOPPER, 2001).

Segundo Elliott (2003), o conceito vem sendo desenvolvido lentamente desde a metade da década de 80. Os detalhes do conceito variam entre diferentes pesquisadores, mas, em geral, a idéia principal é que os computadores tornar-se-ão parte da sociedade, assim como os motores elétricos ou a eletricidade.

> *"Mark Weiser, o pai da computação ubíqua, sugere que esta é a terceira vez que há uma quebra de modelo na computação: (i) os mainframes estão associados a muitos usuários por computador; (ii) os PCs associados a um usuário por computador e, (iii) a computação ubíqua está associada a vários computadores por usuário." Elliott (2003).*

Neste sentido afirma Brooks (2002) que:

> *"... há duas classes de computadores nas casas das pessoas atualmente: (i) uma é caracterizada por processadores embutidos em aparelhos de televisão, cafeteiras e praticamente qualquer outro equipamento elétrico – estes aparelhos são relativamente triviais para interagir e não requerem grande carga cognitiva do usuário humano; (ii) a outra categoria é composta pelos PCs que possuem centenas de opções que podem ser difíceis de entender – estes requerem uma alta carga cognitiva de seus usuários. Seria desejável se os robôs seguissem o caminho dos processadores embutidos ao invés dos PCs e, portanto viessem a requerer uma pequena carga cognitiva para serem utilizados".*

As previsões feitas para os espaços inteligentes (Figura 23), que se utilizam da computação invisível (DERTOUZOS, 1999 NELSON,T., 1998 e ROMÁN e CAMPBELL, 2000), descrevem cenários onde qualquer ambiente previamente preparado, seja uma casa, um escritório ou um espaço público, poderá disponibilizar aos usuários a possibilidade de compartilhar dados (DABEK et al., 2001), preferências e contextos de trabalho ou recreativo.

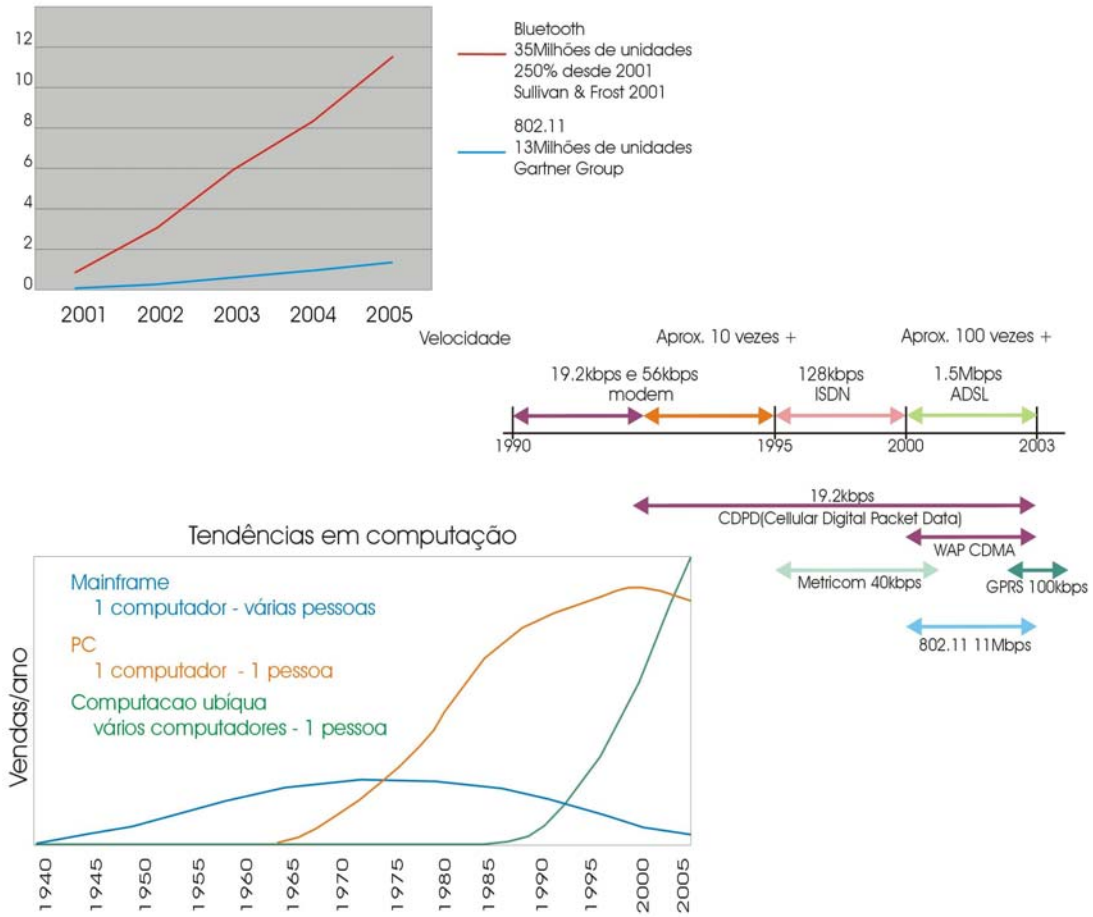


Figura 23 - Tendências em Computação.

FONTE: Adaptado de Weiser (1991) e Want (2003).

Na medida em que esses usuários passam a se movimentar em tais ambientes e contextos diferentes, automaticamente ocorrerá a reconfiguração dinâmica para adequação à nova realidade, que acaba sendo modificada a cada momento. Portanto, os novos modelos de desenvolvimento de software deverão ser independentes de arquitetura de hardware e deverão suportar qualidade de serviço adaptável, segurança dinâmica (ZDANCEWIC et al., 2001) e autoconfiguração.

3.3 Pesquisas em computação ubíqua

As pesquisas sobre computação ubíqua, apesar do envolvimento de um grande número de pesquisadores, ainda estão em fase experimental. Os projetos na área são validados através de aplicações específicas em ambientes controlados (ABOWD, 1999). A seguir são descritos alguns resultados importantes conquistados até o momento.

O projeto **Aura** (WANG e GARLAN, 2000) está centrado na idéia de que o recurso mais precioso num ambiente computacional é a atenção do usuário. O objetivo do projeto é construir uma espécie de aura invisível de recursos computacionais e serviços de informações, a qual persiste independentemente da localização do usuário. Os desafios para atingir este objetivo exigem esforços desde o nível de *hardware* e rede, passando pelo sistema operacional e *middleware*, até as aplicações e, finalmente, a interface com o usuário.

O projeto **Globus** (FOSTER, 1996) investiga um modelo que permite a criação de meta-computadores, ou seja, supercomputadores virtuais, construídos dinamicamente a partir de recursos distribuídos geograficamente, ligados por meio de redes de alta velocidade.

O projeto ***Sentient Computing*** (NEWMAN, 2001), estabelece a seguinte questão: "O que nós poderíamos fazer se programas de computador pudessem ver o modelo do ambiente (*world*) onde estamos? Atuando neste ambiente poderíamos interagir com programas através do modelo". Essa tecnologia está baseada na presença de identificadores (*badges*) que cada pessoa carrega consigo. Através de uma rede de sensores é possível identificar onde as pessoas estão, com quem estão conversando, e se alguém está atendendo ao telefone ou qual ramal está usando. Nesse ambiente é possível utilizar recursos como videoconferência, denominada siga-me (*follow me*). O usuário

seleciona uma câmera de vídeo e, a partir deste momento, ela o acompanha em seus movimentos pela sala. Segundo os autores, o sistema funciona praticamente 24 horas por dia. Ele só não funciona por poucos minutos, durante as operações de backup noturnas.

O projeto do MIT, denominado ***Oxygen*** (DERTOUZOS,1999), vislumbra uma sociedade onde computadores embutidos serão tão avançados que poderão se tornar parte do dia-a-dia das pessoas (Figura 24).

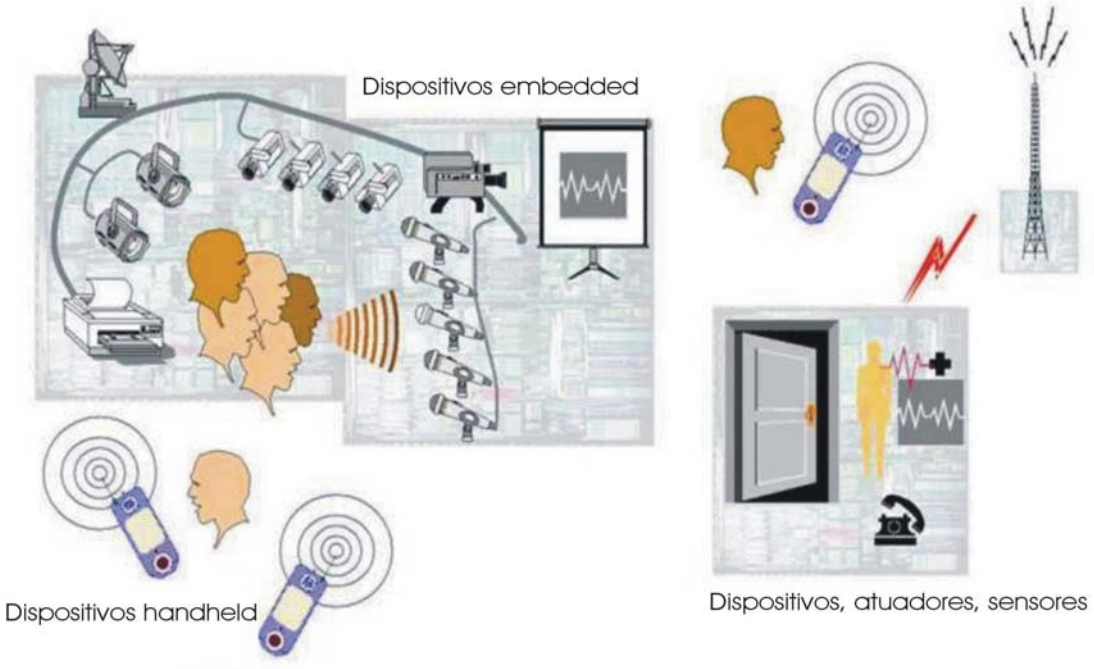


Figura 24 - Visão geral do projeto *Oxygen* do MIT.

O projeto estabelece como meta de pesquisa a criação de um sistema que disponibilize aos usuários uma abundância de recursos computacionais e de comunicação através da interação por voz e interfaces visuais, facilitando as atividades de colaboração, acesso ao conhecimento e automatização de tarefas repetitivas. As premissas desse projeto são:

- O sistema deve estar disponível em qualquer lugar e relacionado a uma mesma base de informações;
- Deve estar disponível no dia-a-dia das pessoas e ser capaz de perceber a rotina e interferir sobre ela;
- Deve ter capacidade de mobilidade conforme as necessidades dos usuários;
- Deve estar sempre disponível, sem que sejam necessários processos de desligamento (*shut-down*) ou recarga (*reboot*). Os componentes devem fluir em resposta às demandas, aos erros e às atualizações e não devem afetar o comportamento exterior do sistema inteligente.

O projeto ***One.world*** (GRIMM et al., 2001) descreve um *framework* para construção de aplicações de espaços inteligentes, através da separação entre dados e funcionalidade. A arquitetura proposta não introduz mudanças fundamentais nas tecnologias de sistemas operacionais. Em seu lugar, propõe um conjunto de serviços de suporte a *check-point*, migração, descoberta (*discovery*) e passagem remota de eventos e replicação.

Nessa mesma linha de pesquisa, que trata da separação de interesses (*concerns*), há o projeto **GaiaOS** (ROMÁN e CAMPBELL, 2000). O principal objetivo desse projeto é construir um *framework* capaz de prover suporte inteligente para o uso de múltiplos canais de comunicação, a partir do conhecimento das necessidades das aplicações, recursos de comunicação e de energia disponíveis. O sistema exporta um modelo que provê a mesma interface e os mesmos serviços básicos para diversos espaços físicos. Este modelo descreve o espaço físico, incluindo as entidades contidas (dispositivos de serviços e pessoas) e suas propriedades (interfaces, características de qualidade de serviço, questões de segurança e localização). Ele suporta aplicações sofisticadas, que exploram os atributos dos recursos computacionais disponíveis em favor dos usuários.

O projeto **JX** (GOLM e KLEINÖDER, 2001) é caracterizado por ser um sistema operacional *single-address-space*, quase totalmente escrito em Java (com uma pequena porção do código escrito em C e *assembly*), que pretende atender aos requisitos acima apresentados. Por ser escrito em Java, apresenta a característica de independência de arquitetura. Além disso, possui uma facilidade de compilação automática, o que garante flexibilidade ao ambiente.

O Projeto **Classroom2000** (ABOWD, 1999) é um experimento destinado a determinar o impacto da aplicação da tecnologia de computação ubíqua na educação, por meio da instrumentação de salas de aula. Abowd relata ainda a tentativa de replicação de um serviço de guia turístico através do uso de tecnologia de *handhelds*, sistemas de posicionamento e serviços de comunicação.

CoolBase é um projeto da HP (CHAN, 2001) que permite a criação de cenários de computação ubíqua chamados de presença *web*, onde espaços físicos são acessíveis através da *web*. Nesse caso, *web sites* deixam de ser entidades virtuais e passam a ser espaços físicos. A plataforma utiliza vários componentes de *hardware* e *software*, que são padrões de mercado, para implementar uma arquitetura que permite a conexão entre

web sites e pessoas, locais e dispositivos. A proposta de uso de padrões abertos (*open standards*), tem por objetivo diminuir os desafios inerentes ao projeto e a criação de aplicações de computação ubíqua.

Scott et al. (2000) descrevem as características e implicações do projeto ***Networked Surfaces*** da AT&T Labs. Neste trabalho é apresentada uma visão da tecnologia usada através da descrição de um cenário formado por superfícies que possuem conectividade à rede, com o objetivo específico de potencializar os objetos que são fisicamente colocados sobre elas. Quando um objeto (por exemplo, um notebook) conecta-se, um protocolo de reconhecimento (*handshaking*) associa funções, tais como transferência de dados ou energia, através dos vários condutores instalados.

O projeto ***Easy Living*** (KRUMM et al., 1999), da Microsoft, concentra-se em ambientes comerciais e domésticos. Preferências são associadas aos usuários para configurar os espaços físicos. Um modelo geométrico é usado para acompanhar objetos no espaço e um sistema de localização é desenvolvido através de uma rede de câmeras, ligadas a um conjunto de software de detecção de objetos.

Outro projeto da Microsoft, denominado projeto ***Millennium*** (BOLOSKY et al., 2001), propõe que a base para uma nova arquitetura de componentes está centrada em um modelo de serviços orientados a usuários e aplicações, onde o ambiente distribuído ajusta-se aos novos requisitos de mudança de usuário e aplicação. Utilizando reflexão computacional, o sistema pode alterar os atributos de invocação de objetos para, por exemplo, atender restrições de baixa energia ou de mobilidade. Assim sendo, o sistema configura-se automaticamente, carregando um conjunto mínimo de componentes para atender às necessidades do usuário, usando o modelo de gerência de recursos que pode ser resumido à máxima: *what you need is what you get*[29].

Segundo Muthitacharoen, Chen e Mazières (2001), a heterogeneidade de dispositivos exige um *framework* de comunicação transparente e integrado, dentro do qual a meta deve ser a de fornecer o melhor nível possível de qualidade de serviço fim-a-fim. Esse aspecto é especialmente desafiador porque os diferentes tipos de dispositivos de comunicação podem diferir em capacidade de comunicação, banda de transmissão e consumo de energia.

[29] Aquilo que você necessita é o que você obterá.

Devem ser considerados os seguintes aspectos:

- Quando o usuário comunica-se dentro de um espaço inteligente;
- Quando o usuário se comunica através de diferentes espaços inteligentes (YU e VAHDAT, 2001); ou
- Quando se comunica durante a transição entre dois diferentes espaços (MUTHITACHAROEN,CHEN e MAZIÈRES, 2001).

A partir de um levantamento sobre as características dos projetos acima referidos, foi possível a identificação dos principais requisitos de computação ubíqua, os quais são apresentados na próxima seção.

3.3.1 Principais requisitos de computação ubíqua

Analisando-se o perfil das propostas acima, obtém-se um conjunto de requisitos que uma aplicação invisível deve apresentar. Em ambientes dominados pela computação ubíqua é explícita a necessidade de que as aplicações percebam simultaneamente as mudanças que ocorrem no espaço operacional. Isto permite que elas se adaptem às necessidades do usuário, alterando sua capacidade de computação, comunicação e de interação com o usuário. Compilando-se as propostas das várias linhas de pesquisa, chega-se ao seguinte conjunto de requisitos fundamentais (não exclusivo):

- Mobilidade de código estático e dinâmico;
- Controle de recursos com qualidade de serviço;
- Segurança;
- Robustez;
- Suporte a diferentes formas e estilos de interação com o usuário;
- Localização e;
- Aspectos sociais.

A seguir são apresentados os argumentos para cada um dos requisitos acima relacionados.

3.3.1.1 Mobilidade de código estático e dinâmico

O código móvel deve ser independente de plataforma para poder ser executado em diferentes equipamentos. Atualmente, a maior parte dos sistemas de código móvel utiliza um interpretador, que roda como um processo do sistema operacional nativo. Quando um

agente necessita acessar primitivas deste sistema nativo, deve atravessar uma camada significativa de proteção, o que implica em perda de performance. No momento em que esse tipo de código migra para outra máquina, um novo ambiente de execução deve ser criado. Em sistemas tradicionais, isso significa criar um novo processo para o interpretador. Essa característica de migração de código em execução é referida freqüentemente como *strong migration* (GHEZZI e VIGNA, 1997). Para que isso ocorra, é necessária a transferência do estado de execução da aplicação, incluindo a porção do estado do sistema operacional no exato momento em que a aplicação está sendo migrada.

Deve-se considerar, também, que tanto as questões de mobilidade quanto aquelas relativas à heterogeneidade dos equipamentos envolvidos, possuem requisitos diferentes nos contextos de utilização. É possível, por exemplo, que, num ambiente restrito, haja largura de banda suficiente para rápida transferência de um grande volume de dados, enquanto que, num ambiente aberto (*wide area*), as restrições de largura de banda e consumo de energia são determinantes no aspecto de performance. Contudo, um espaço inteligente deve explorar os recursos disponíveis em determinado contexto e se adequar da melhor forma às necessidades do usuário (STEVEN,1999; WELSH, CULLER e BREWER,2001).

3.3.1.2 Controle de recursos com qualidade de serviço

Uma das principais funções de um sistema operacional é a gerência de recursos. São amplamente discutidos na literatura (DEITEL, 1990; TANENBAUM, 1987) os aspectos desse ponto, principalmente quando se trata dos sistemas operacionais tradicionais. Além deles, no entanto, é preciso adicionar o fato de que o sistema deve gerenciar também o consumo de recursos de código baixado (*downloaded*) dinamicamente.

Segundo Golm e Kleinöder (2001), os sistemas tradicionais enfrentam problemas para controlar o consumo de recursos porque empregam estratégias de melhor esforço (*best-effort*), o que apresenta uma granularidade muito alta, pouco adequada quando os recursos são compartilhados por muitos processos. É necessário, portanto, que as informações sobre o perfil de consumo de recursos da aplicação que está sendo baixada (*downloaded*), sejam recebidas pelo sistema operacional hospedeiro (*host*) antecipadamente. Isso faz com que esse sistema tenha condições de se adequar à nova

demanda. Dessa forma, é possível evitar sobressaltos e permitir que o sistema operacional se ajuste mais suavemente às flutuações nas demandas advindas da rede.

Para atender à demanda de qualidade de serviço, ainda é necessário que o modelo suporte o controle de serviços e recursos fim-a-fim, utilizando camadas capazes de sustentar funções de reserva de recursos e controle adaptativo, de tal modo que se possa disponibilizar um alto grau de qualidade e adaptabilidade de serviços. Também é imprescindível um nível de controle para prover facilidades de reserva de recursos, que sustente um conjunto mínimo de requisitos de qualidade. Um outro nível deve ser responsável pelo controle de disponibilidade de recursos frente a picos de demanda. Este tipo de flutuação ocorre quando as requisições superam as reservas de recursos solicitadas. Facilidades de controle adaptativo fornecem estabilidade, agilidade e reagem a pequenas mudanças na alocação de recursos, acima do nível reservado. E, finalmente, é preciso garantir um nível de controle que aceite comandos e parâmetros difusos, com o objetivo de disponibilizar facilidades de adaptação funcional e reconfiguração dinâmica para atender aos requisitos de qualidade de serviço.

O objetivo de um sistema de configuração automática é facilitar a configuração do sistema e das aplicações através do gerenciamento de dois tipos de dependências entre componentes: pré-requisitos, que especificam os requisitos para carga de um componente no sistema, e dependências dinâmicas, que se referem aos relacionamentos entre componentes de software durante a execução dos mesmos. Na medida em que um sistema tem acesso a essas informações, o procedimento de instalação e configuração pode ser automatizado. Como subproduto desse conhecimento, a performance dos componentes pode ser melhorada, analisando-se o estado dinâmico dos recursos do sistema, as características de cada componente e configurando-os adequadamente.

3.3.1.3 Segurança

Um espaço inteligente é composto por muitos dispositivos embutidos e móveis e um determinado número de usuários móveis. É necessário, portanto, o desenvolvimento de novos enfoques para garantir a segurança. O controle de acesso ao ambiente deve compreender, entre outras coisas, o suporte a senhas, crachás e *smart cards* e o reconhecimento de funções biométricas. Toda uma API deve ser desenvolvida como suporte para essas funções.

Outro aspecto importante a destacar é o fato de que, num ambiente tão heterogêneo, onde há dispositivos com limitações na capacidade de processamento, no consumo de energia e na disponibilidade de memória, os mecanismos de segurança devem utilizar diferentes padrões de qualidade de proteção (*Quality of Protection*) (CHANDAK, 1999).

3.3.1.4 Robustez

Falhas são comuns em sistemas computacionais. A proposta de espaços inteligentes consiste em tornar um sistema computacional invisível aos usuários. Para atingir esse nível de abstração, as falhas em tempo de execução devem ser reduzidas ao mínimo e a forma de tratamento daquelas que ocorrerem deve ser menos complexa. A confiabilidade deve ser garantida para todos os níveis do sistema, dos *device drivers* ao código móvel das aplicações. Segundo Szyperske e Gough (1995), somente as linguagens estaticamente seguras, como Java, podem ser capazes de atingir esse nível de confiabilidade. Nos sistemas atuais, a proteção é obtida usando isolamento de falhas de software (WAHBE et al., 1993) ou executando-se um interpretador para linguagens ditas seguras, tais como **Tcl**, **Java**, **DIS** (DORWARD et al.,1997), **Omniware** (COLUSA,1995).

Nos últimos anos, as discussões sobre formas adequadas de tratamento de erros têm recebido maior atenção, principalmente com a popularização de estruturas do tipo *try-catch* (**Java**, **C++**, **Object Pascal**) nas linguagens de programação usadas comercialmente. Estas estruturas permitem uma forma mais organizada de controle de exceções ao nível de programação. Contudo, afirmam Robillard e Murphy (1999): "da mesma forma que a estrutura das operações normais de um sistema tendem a degradar a medida em que o sistema evolui, a estrutura de tratamento de exceções também degrada."

Em função disto, a forma de comunicação das falhas permanece um tanto quanto rude e inadequada ao usuário final, o que implica diretamente na necessidade de mais investimentos na fase de desenvolvimento das aplicações. Geralmente os sistemas tradicionais têm dificuldades em localizar precisamente o erro e reportá-lo ao usuário de forma amigável e de modo compreensível. Assim sendo, a comunicação do erro é direcionada para o lugar errado.

O usuário final não quer e não deveria saber que uma aplicação provocou uma "falha de proteção geral". Quem deveria ser informado sobre o problema é o quem

desenvolveu o sistema. No entanto, as mensagens geradas são incompreensíveis para os usuários finais e insuficientes para quem deve realmente se interessar por elas.

Cabe salientar que, hoje, soluções para esse problema envolvem basicamente a automatização do processo de comunicação, através da coleta e do envio dos registros (*logs*) de erro, normalmente via Internet, para as equipes de desenvolvimento e suporte. Durante o prazo de análise e solução do problema, o usuário fica impossibilitado de usar o sistema, o que não pode ocorrer num espaço inteligente. Nessa situação, os ambientes inteligentes deverão tratar o problema de uma forma que permita o isolamento dos módulos com falhas, sem que a operação na parte do sistema não atingida pela falha seja interrompida. Tanenbaum (1987) define esta característica como capacidade de degradação suave do sistema.

3.3.1.5 Interação com o usuário

Segundo Tandler (2001), a infra-estrutura de software necessária para o suporte do desenvolvimento de aplicações de um espaço inteligente deve, além de incluir os requisitos acima mencionados, considerar aspectos relativos às diferentes formas de interação, decorrentes da heterogeneidade dos equipamentos (ABOWD,1999), e à característica de colaboração intrínseca dos espaços inteligentes (MYERS,HUDSON e PAUSCH, 2000).

Em primeiro lugar deve-se levar em conta que a variedade de dispositivos e as diferentes formas de interação entre eles é muito grande, se comparadas aos tradicionais ambientes *desktops* equipados com monitor, *mouse* e teclado. Segundo Dertouzos (1999),

> *"(...)não será necessário teclar ou clicar ou aprender qualquer jargão de informática para mover-se nos espaços inteligentes Ao invés disso, nós iremos nos comunicar naturalmente, usando a voz, a visão e frases que descrevam nossas intenções, deixando que o ambiente aloque os recursos apropriados e execute nossas intenções".*

Essa diversidade apresenta características e limitações específicas que devem ser consideradas por quem desenvolve as aplicações para ambientes dinâmicos. Já o aspecto de colaboração entre usuários e dispositivos conduz a vários cenários de uso. Um usuário pode utilizar vários dispositivos simultaneamente. Vários usuários podem compartilhar informações a partir de periféricos com diferentes características de interação. Nessa situação, a característica da informação apresentada a um usuário, possivelmente, não é a mesma apresentada a outro. Também é possível prever um cenário em que vários

usuários compartilham o mesmo dispositivo ou mesmo um dispositivo virtual, construído a partir da junção temporária ou permanente de vários dispositivos.

3.3.1.6 Localização

A questão da localização é um aspecto tão importante para a viabilização da computação ubíqua que eventos dedicados à discussão de aspectos específicos desta tecnologia começam a surgir, como é o caso da edição da *Personal and Ubiquitous Computing (Springer-Verlag),* que publica os artigos selecionados do primeiro workshop em Modelagem de Localização ocorrido no evento *UbiComp Conference*, em Atlanta, em 2001.

Segundo Antifakos e Schiele (2002), no contexto de computação ubíqua a localização é geralmente usada como um sinônimo para informação posicional. Este tipo de informação é uma rica fonte de recursos para aplicações que são dependentes de contexto. Provendo os computadores com estas informações será possível executar ações como abertura automática de portas para certas pessoas, carga correta de informações na tela para quem está sentado em frente a um terminal, ou encaminhamento de mensagens eletrônicas ou de voz diretamente para o aparelho telefônico mais próximo do usuário.

> *"Os aspectos técnicos de informação de localização incluem tanto sistemas de posicionamento internos (indoor) como externos (outdoor). As implementações atuais utilizam o sistema de posicionamento global (GPS) para posicionamento externo e os active badges desenvolvidos pela Olivetti Research - Cambridge."* Elliott *(*2003*).*

O uso de medidas de proximidade é de grande importância em computação ubíqua. Antifakos e Schiele (2002), propõem uma hierarquia de proximidade semântica baseada em uma rede de sensores *wireless* tendo em vista complementar as informações de um sistema puramente posicional.

Segundo Kirsh (1995) apud Beigl, Zimmer e Decker (2002),

> *"... por termos corpo, nós somos criaturas espacialmente localizadas: nós estamos sempre voltados para alguma direção, tendo somente certos objetos no campo de visão e podendo alcançar certos outros objetos. Como nós controlamos a organização espacial de itens a nossa volta não é um afterthought – é uma parte integral da forma como nós pensamos, planejamos e nos comportamos."*

Em computação ubíqua, os modelos e aplicações que são projetadas para apresentar informações ou reagir de uma forma compreensível por pessoas, devem levar em conta a localização das coisas envolvidas no contexto da interação.

Beigl, Zimmer e Decker (2002), complementam que:

> *"Embora seja possível projetar aplicações de um ponto de vista a partir do qual a experiência humana não seja importante, nós acreditamos que na prática projetistas e programadores tendem a ignorar ou deixar para depois (shift) tais pontos de vista".*

3.3.1.7 Aspectos sociais

Além dos aspectos técnicos discutidos anteriormente, deve-se destacar alguns outros relacionados a mudanças culturais e sociais decorrentes do uso desta tecnologia.

A meta principal da computação ubíqua é criar um ambiente onde o poder computacional envolve (*surrounds*) o usuário, sem que este seja solicitado a operar o computador para executar alguma tarefa. Segundo Elliott (2003), um dos maiores argumentos contra o emprego de computadores em todos os lugares é o risco de perda de privacidade pessoal e segurança. Os defensores desta preocupação utilizam a metáfora apresentada no livro de George Orwell, **1984**, relativamente ao "*Big Brother*", ou seja, uma superentidade que controla e vigia todos os passos do indivíduo em suas relações sociais.

Para caracterizar melhor esta questão, estudantes de Stanford, apresentam a seguinte consideração em um estudo sobre Privacidade de Computadores e Redes conduzido pela Prof. Bárbara Simons (AGRAWAL et al., 2002):

> *"Os eventos ocorridos em meses recentes podem ter um impacto direto sobre a privacidade dos registros de estudantes atuais e futuros. O jornal "*The San Jose Mercury News*" recentemente publicou: "Colégios em todo país... estão fornecendo ao FBI e outras agências federais registros de estudantes alinhando-se a investigação sobre os ataques terroristas de 11 de setembro"*[30]

Uma outra questão importante refere-se à dependência criada em torno da tecnologia. Se a computação ubíqua tornar-se aceita pela sociedade como um todo, ela deverá ser tão robusta como outras tecnologias ubíquas que já estão presentes nas residências. Contudo, os computadores atuais requerem um grande investimento em suporte técnico e manutenção. Portanto, para que as pessoas sintam-se confortáveis com as facilidades disponibilizadas pela computação ubíqua, estes aspectos da tecnologia atual devem ser "remediados" (ELLIOTT, 2003).

---

[30] The events of recent months may have a direct impact on the current and future privacy of student records. The San Jose Mercury News recently reported: "Colleges across the country … are providing the FBI and other federal agencies with student records in connection with the investigation of the Sept. 11 terrorist attacks."

Espaços inteligentes devem suportar transferência de grandes volumes de dados entre dispositivos e serviços. O caráter heterogêneo do ambiente faz com que alguns dispositivos não sejam capazes de interpretar os dados em seu formato original. Por isso é necessária uma camada de software que permita a adaptação do conteúdo às características de um determinado dispositivo, principalmente em função dos requisitos de qualidade da aplicação.

3.3.1.8 Considerações finais

Apesar de uma série de avanços nas pesquisas em sistemas operacionais não estarem diretamente ligadas às pesquisas em computação ubíqua, eles podem contribuir apresentando soluções que eventualmente serão utilizadas no desenvolvimento de espaços inteligentes.

A seguir são apresentados os requisitos funcionais que esta nova geração de sistemas operacionais deverá contemplar para fazer frente às demandas identificadas nesta seção.

3.4 Demandas em Teletrabalho

A era que ora se consolida tem sido referida freqüentemente como "Era da Informação" (ZAIANE e JIAWEI, 1995; DRUKER, 1999 apud MACHADO, 2002). Isto porque a evolução tecnológica possibilitou o acesso, aquisição e tratamento de grandes volumes de dados que outrora era impossível conceber. Este fato promoveu profundas mudanças estruturais na sociedade de hoje.

Na ponta deste movimento estão as organizações que necessitam lidar não mais com a previsibilidade, continuidade e estabilidade, mas sim com seus contrários, ou seja, com a incerteza. Esta incerteza é provocada pelas rápidas e bruscas mudanças que a sociedade experimenta a partir da forte competição entre as empresas e o fenômeno da globalização da economia. Isto porque, para sobreviver num ambiente turbulento e de crescente competitividade, as organizações estão buscando alternativas viáveis para seus negócios, assim como estruturas organizacionais e novas formas de trabalho (MACHADO, 2002).

Segundo Steil e Barcia (2001) apud Machado (2002), em função destes fatores, a flexibilidade passou a ser uma necessidade para as organizações. A diminuição das

fronteiras intra e interorganizacionais passou a gerar novas formas de organizações caracterizadas pela dispersão temporal e espacial. Neste contexto surgiu e está se desenvolvendo a proposta do teletrabalho.

Segundo Nilles (1997) apud Machado (2002), "a proposta do teletrabalho era que os trabalhadores poderiam exercer suas atividades em seus domicílios, o que em si não era uma novidade, mas empregando recursos computacionais e de telecomunicações, o que consistia em um novo modelo nas relações de trabalho: o trabalho a domicílio (VERAS OLIVEIRA,1996 apud MACHADO,2002) onde o contato entre as pessoas ocorre também por meios eletrônicos de comunicação" (MACHADO,2002).

Bell (2000) apud Machado (2002) sintetiza em um quadro a evolução do espaço de teletrabalho que começa com organizações tradicionais e centralizadas, passa pela adoção de tecnologia de informação, dividida em duas fases – a informatização e a interligação em redes, até chegar na fase da virtualização das empresas, nos dias atuais e num futuro próximo (Figura 25).

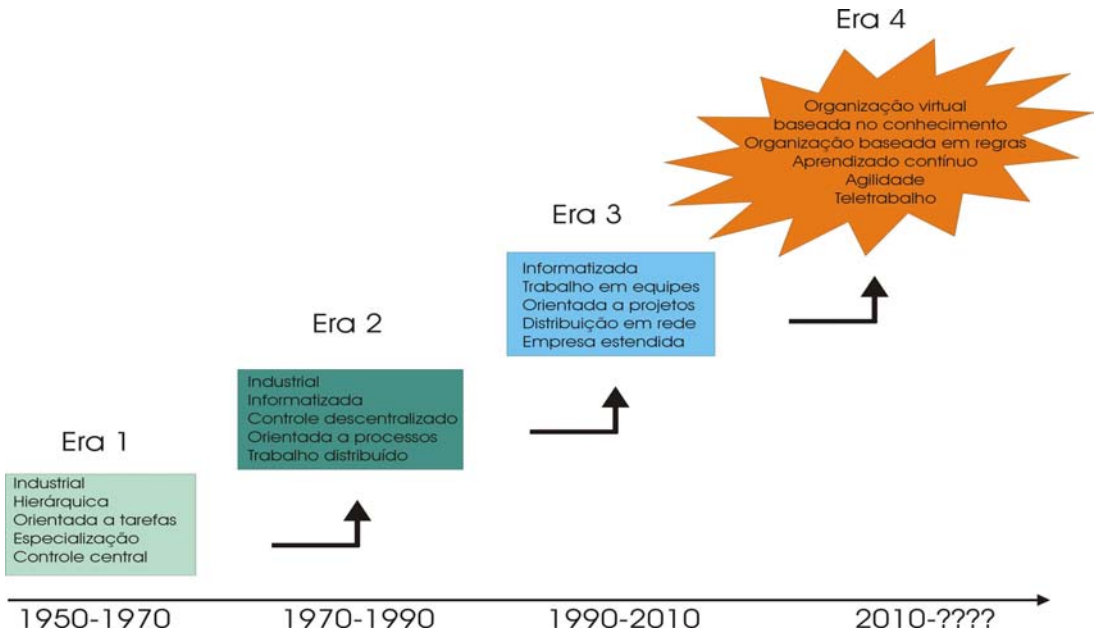


Figura 25 - As quatro fases da evolução do espaço de trabalho.

FONTE: Adaptado de Machado (2002).

As demandas de teletrabalho à área de sistemas operacionais estão relacionadas, principalmente, à disponibilidade e facilidade de acesso remoto a servidores das empresas, bem como a aspectos relativos à segurança das informações, tanto aquelas

acessadas pelo usuário na empresa, como aquelas produzidas ou copiadas para a máquina do teletrabalhador. Estes aspectos são discutidos na seção 5.4.4(Pág.158) no contexto das deficiências que o atual modelo de sistemas operacionais apresenta.

3.5 Demandas em Comércio Eletrônico

Uma outra linha de demandas aos sistemas operacionais decorrente do movimento de globalização é o comércio eletrônico – uma metodologia de negócio que atende as necessidades das organizações, *merchants* e consumidores através do corte de custos e melhoria da qualidade de bens e serviços e da rapidez na entrega (*delivery*) (KALAKOTA e WHINSTON, 1996).

O comércio eletrônico caracteriza-se, portanto, como uma nova forma de conduzir, gerenciar e executar transações comerciais usando recursos computacionais e de telecomunicações. O termo comércio eletrônico (*e-commerce*) está associado à idéia de convergência de tecnologias como produtos eletrônicos, televisão, *publishing*, telecomunicações e computadores com a finalidade de facilitar novas formas de comércio baseado em informações.

Estas demandas do comércio eletrônico, de forma semelhante àquelas apresentadas pelo teletrabalho à área de sistemas operacionais, estão relacionadas, principalmente, à disponibilidade e facilidade de acesso remoto a servidores das empresas, bem como a aspectos relativos à segurança e privacidade das informações sobre transações comerciais, como dados de pedidos, documentos (CPF, Número de cartão de crédito), identificação do cliente, entre outras. Estes aspectos são discutidos na seção 5.4.5(Pág.162) no contexto das deficiências que o atual modelo de sistemas operacionais apresenta.

3.6 Considerações finais

A Tabela 19 apresenta os requisitos funcionais compilados a partir das informações apresentadas neste capítulo.

Tabela 19 - Requisitos funcionais de computação ubíqua, teletrabalho e comércio eletrônico.

Requisitos Funcionais
• Mobilidade de código estático e dinâmico. • Controle de recursos com qualidade de serviço. • Segurança. • Robustez. • Interação com o usuário. • Localização. • Capacidade de tratar informações parciais e/ou imprecisas. • Capacidade de extração de conceitos. • Capacidade de raciocínio aproximado (difuso) e de aprendizado. • Facilidades para usuários especializados aumentarem as bibliotecas do sistema operacional sem comprometimento dos aspectos de segurança e confiabilidade. • Sistema de arquivos com maior nível de abstração. • Identificação de perfil de usuário e adequação do ambiente a este perfil. • Privacidade das informações sobre transações. • Privacidade sobre dados pessoais do usuário. • Confidencialidade das informações recuperadas/enviadas pela rede. • Integridade das informações transitadas/produzidas pelo usuário. • Disponibilidade das informações. • Criptografia das informações.

Depois do que vimos neste capítulo, é possível concluir e afirmar que os sistemas operacionais, atualmente, não contemplam de forma adequada as demandas antes referidas. Este fato este deve disparar o início de uma nova fase no ciclo evolutivo dos sistemas operacionais, o que ocorrerá a partir do desenvolvimento de bibliotecas especializadas em atender a demandas específicas destas aplicações, conforme apresentado na seção 2.6(Pág.85).

O próximo capítulo apresenta as abordagens de tecnologias de informações que são adotadas no projeto dos sistemas operacionais no sentido de estender (ou incorporar) facilidades que venham a contribuir para a melhoria da qualidade dos serviços por eles prestados.

4 ABORDAGENS DE INSTRUMENTALIZAÇÃO DE SISTEMAS OPERACIONAIS

"Os últimos anos tem estabelecido o início de uma nova era para os sistemas operacionais. A computação distribuída é certamente uma dimensão deste contexto. Nós afirmamos, contudo, que aspectos qualitativos serão uma segunda dimensão maior ainda. Estas mudanças causarão profundos efeitos no projeto de sistemas s operacionais." Nicol et al.(1989).

4.1 Introdução

O presente capítulo tem por objetivo analisar algumas das abordagens de tecnologia de informação que estão sendo consideradas para instrumentalizar o sistema operacional a fim de que ele possa atender algumas demandas ainda não satisfeitas pelo atual modelo, como gerência de grande volume de dados, entre outras. Também são analisadas outras abordagens que já consideram os requisitos que as pesquisas em computação ubíqua começam a demandar. Neste contexto, a seção 4.2 apresenta algumas pesquisas que empregam técnicas da área de Inteligência Artificial. Apresenta, também, outras tecnologias que estão sendo empregadas com o mesmo propósito, assim como a abordagem adotada em projetos de robôs autônomos.

4.2 Emprego de técnicas de Inteligência Artificial em Sistemas Operacionais

Esta seção apresenta e analisa algumas propostas existentes na literatura e que empregam técnicas de inteligência artificial em sistemas operacionais, identificando as vantagens e limitações que existem em seu emprego no contexto. Em sua grande maioria, estas propostas têm por objetivo instrumentalizar um sistema operacional para atender os *aspectos qualitativos* referidos por Nicol et al. (1989).

Conforme Gürer et al. (2001), verifica-se claramente que o uso das técnicas de inteligência artificial tem sido cada vez mais permeado entre as soluções produzidas nas mais diversas áreas de aplicação.

Segundo Seltzer e Small (1997), determinar que partes do núcleo são críticas para a performance de uma aplicação, requer um profundo conhecimento sobre as demandas que ela solicita ao sistema operacional. Em áreas como sistemas gerenciadores de banco de dados, estas demandas são bem conhecidas. Contudo, em outros domínios de

aplicação, onde as demandas não são completamente conhecidas, este não é um método conveniente para obter tais informações. E Seltzer et al. (1996) continuam: "...uma das lições aprendidas pelos grandes desenvolvedores de projetos de software é que nunca há dados suficientes sobre como o sistema operacional está operando e o que está ocorrendo de errado".

Saltzer e Gintell [31] (1970) apud Seltzer e Small (1997) afirmam:

> *"... lembra os projetistas de que medições são mais confiáveis que a intuição e podem revelar problemas que os usuários ainda não haviam detectado. Ele encoraja os projetistas a concentrarem-se no entendimento do funcionamento interno do sistema, ao invés de concentrarem-se nos tempos de resposta baseando-se nas alterações de carga de trabalho (workloads)".*

Enquanto em muitos sistemas da década de 70 foram construídas ferramentas significativas para avaliação de performance, a década de 80 caracterizou-se por apresentar uma surpreendente ausência delas. Atualmente sistemas como o **Unix** apresentam um conjunto de utilitários que permite seu constante monitoramento, tais como **netstat**, **iostat** e **vmstat** No entanto, qualquer sistema que faça uso desta coleção de ferramentas de monitoramento de performance tem que fazer analisar os dados capturados *off-line* (SELTZER e SMALL, 1997).

Uma outra fonte de dados importantes advém do próprio hardware. Atualmente, vários microprocessadores contêm um ou mais contadores de instrumentação que fornecem dados para análise de performance. Conforme Seltzer e Small (1997), a família de processadores Pentium contém um contador de 64 bits e dois contadores de 40 bits que podem ser configurados para contar eventos de hardware, tais como *TLB misses*, *cache misses* e carga de registradores de segmento. Os microprocessadores Sparc e Alpha também possuem contadores de performance, embora nenhum deles possua um conjunto como aquele disponível nos processadores Pentium. Monitoramento e coleta regular destes contadores de hardware constitui-se em uma importante fonte de informação e insight sobre o comportamento do sistema (SELTZER e SMALL,1997).

A atividade de administração de um sistema operacional é uma tarefa árdua e complexa. Muitas vezes chega a ser uma questão "vaga" e que depende da experiência de alguns chamados "gurus". Apesar da grande quantidade de manuais e livros

[31] SALTZER, J.H.,GINTELL,J.W. The Instrumentation of Multics. Communications of the ACM, Vol.13, No.8,August 1970,pp.495-500.

disponíveis sobre o assunto, esta atividade requer um nível de especialização bastante elevado. Exige, também, constantes atualizações, tendo em vista que a cada momento são disponibilizadas novas versões incorporando facilidades recém surgidas e/ou corrigindo problemas, as quais, por sua vez, incorporam novo comportamento ao sistema. Isto conduz a um círculo vicioso e desgastante.

Não obstante a documentação descrever o que significam os parâmetros dos comandos e os resultados esperados, há poucas referências a respeito da análise que deve ser feita sobre os dados resultantes. E, na verdade, esta informação não é disponibilizada porque depende de vários fatores inter-relacionados, tais como:

- Configuração do hardware;
- Configuração atual do sistema operacional para operar com o hardware;
- Carga do sistema em função do número de usuários ativos;
- Característica das aplicações que estão em execução.

A seguir são apresentados trabalhos relacionados ao emprego das seguintes técnicas de Inteligência Artificial em soluções para instrumentalizar sistemas operacionais:

- Sistemas Especialistas.
- Redes neurais.
- Lógica difusa.
- Redes Probabilísticas.
- Mapas Cognitivos Difusos (Fuzzy *Cognitive Maps)*

4.2.1 Sistemas Especialistas

Cockcroft (1995) descreve uma iniciativa que utiliza conceitos de Sistemas Especialistas sobre a arquitetura de um sistema operacional tradicional (**Unix**). A estratégia adotada, apesar de não incluir regras difusas, apresenta os resultados do estado dos vários componentes do sistema, através de uma notação de cores que poderia ser considerada como uma forma de representação gráfica de variáveis lingüísticas. A notação é a seguinte:

- **Estado branco:** indica que o recurso não está sendo utilizado;
- **Estado azul:** indica que o recurso está desbalanceado;

- **Estado verde**: indica estado de operação normal;
- **Estado âmbar:** indica condição de atenção;
- **Estado vermelho:** indica sobrecarga ou detecção de algum problema;
- **Estado preto:** indica problema.

Através desta interface, o usuário pode mais facilmente mapear intuitivamente o estado do sistema operacional, sem ter que se aprofundar em estudos de valores numéricos. Esta solução está disponível somente para sistemas Solaris, e vale-se da experiência de Cockcroft, como administrador de sistemas **Unix**, na criação da base de regras. Apesar da base de regras estar disponível para *download*, há pouca documentação auxiliar que facilite os ajustes dos percentuais adotados por Cockcroft quando de sua criação, o que dificulta o seu emprego em outros sistemas operacionais. No entanto, como solução geral, é uma clara indicação de que, mesmo em sistemas não escritos especificamente para suportar tais aplicações, é possível a adoção desta técnica como ferramenta de suporte ao usuário.

Hernández,Vivancos e Botti (1998) descrevem um enfoque para incorporar sistemas de produção baseados em regras, em uma arquitetura de tempo-real denominada **ARTIS** (*Architecture for Real-Time Intelligent Systems*). A correção de um sistema a tempo-real depende não somente dos resultados lógicos da computação, mas também do tempo em que estes resultados são produzidos. Por outro lado, problemas que são abordados utilizando-se técnicas de Inteligência Artificial, geralmente requerem um processo de busca como componente principal. A busca pode ser executada por sistemas de produção que navegam através do conhecimento disponível, em busca da solução apropriada (HERNÁNDEZ,VIVANCOS e BOTTI**,**1998).

Contudo, esta dependência dos algoritmos de busca geralmente conduz a tempos de resposta imprevisíveis. Há duas dificuldades principais em limitar o tempo de execução de sistemas de produção (BARACHINI[32],1994 apud HERNÁNDEZ,VIVANCOS e BOTTI,1998):

- O conteúdo da memória de trabalho do sistema muda continuamente e o número de elementos da memória varia cada vez que uma regra é disparada;
- Diferentemente das linguagens procedimentais, cujo controle depende das sentenças, o fluxo de controle em sistemas de produção é embutido nos dados e não pode ser facilmente derivado.

[32] BARACHINI,F. Frontier in Run-time Prediction for the Production-System Paradigm. AI Magazine, 15(3). Fall 1994.

Musliner[33] et al. (1993) apud Hernández,Vivancos e Botti (1998), afirmam que, em função das questões acima relacionadas, alguns enfoques têm sido empregados para incorporar técnicas de Inteligência Artificial em ambientes de tempo real:

- Modificação das técnicas de IA tradicionais, incluindo sistemas de produção, tendo em vista melhorar a sua previsibilidade;
- Arquiteturas de software adaptadas para combinar técnicas de IA sem restrições com a possibilidade de implementar garantias de performance para aplicações *hard real time*;
- Modificação das políticas de escalonamento tradicionais para acomodar o uso de algoritmos não previsíveis;
- Mescla das duas últimas estratégias através do desenvolvimento de uma extensão do S*hell* de sistema especialista Clips 6.0 (MultiClips) para permitir a execução concorrente de múltiplos sistemas de produção.

A estratégia adotada em **ARTIS** utiliza um módulo *hard real-time* como base da arquitetura e um escalonador inteligente para escalonar todas as partes não previsíveis do sistema. O escalonador inteligente e a extensão *MultiClips* são executados em modo de "tempo frouxo" (*slack time*), de forma que o *timing* dos sistemas de produção não previsível não pode interferir com as garantias de prazo das operações do módulo previsível.

4.2.2 Redes neurais

Zomaya, Clements e Olariu (1998) apresentam uma solução para o problema de escalonamento dinâmico de ambientes de multiprocessadores baseada na técnica de redes neurais e *reinforcement learning*. A categoria de escalonadores adaptativos engloba aqueles que se ajustam à realidade de acordo com a história recente e/ou ao comportamento do sistema. Para tanto, o uso de redes neurais com aprendizado adaptativo apresentou resultados interessantes em termos de simulação.

A indicação de Zomaya, Clements e Olariu (1998) é de que sua pesquisa continuará no sentido de explorar o paralelismo inerente à estrutura do sistema de aprendizagem, bem como no uso de outras técnicas não ortodoxas que também podem apresentar bons resultados, tais como: algoritmos genéticos e lógica difusa.

Como se pode observar, apesar de ser um projeto de pesquisa, ele demonstra que, a partir do momento em que houver suporte de co-processamento (difuso, neural ou

[33] MUSLINER,D.J.,HENDLER,J.A.,DURFEE,E.H.,STROSNIDER,J.K.,PAUL,C.J. The Challenges of Real-Time Artificial Intelligence. Computer IEEE,, 28(1):58-66. 1995.

genético), a aplicação da técnica passa a ser um recurso em potencial a ser utilizado na construção de sistemas operacionais.

Madhyastha e Reed (1996) descrevem um trabalho (*Portable Parallel File System* - PPFS) que objetiva aumentar a performance de operações de entrada/saída de uma aplicação através de um sistema de entrada/saída de baixo *overhead* que reconhece, de forma automática, padrões de acesso a arquivos e adapta as políticas do sistema para adequá-lo às necessidades de entrada/saída da aplicação.

Este enfoque reduz os esforços do programador de aplicação, isolando decisões de otimização de operações de entrada/saída dentro da infra-estrutura do sistema de arquivos. Para classificar automaticamente os padrões de acesso a arquivos, foi desenvolvida uma rede neural *feed forward* capaz de produzir uma descrição qualitativa dos padrões de acesso. Segundo a autora, "a imprecisão inerente das redes neurais torna possível classificar padrões que são muito próximos a um padrão bem conhecido".

4.2.3 Lógica difusa

A linguagem humana pode ser caracterizada pelo uso corriqueiro de termos cuja tradução em termos precisos não ocorre sem alguma perda semântica. Na área de sistemas operacionais, em particular, pode-se constatar esta afirmação, através da avaliação que normalmente se realiza sobre o nível de performance dos equipamentos em determinado momento - "a máquina está lenta" ou "o sistema operacional está pesado". Embora seja muito difícil traduzir estas afirmações em termos percentuais, as pessoas entendem muito bem o seu significado.

Lógica difusa, por definição, é um ramo da lógica que utiliza graus de pertinência em conjuntos, ao invés de utilizar uma referência de pertinência estritamente verdadeira ou falsa. Ou seja, a idéia básica é que os valores que representam verdades são representados dentro do intervalo entre [0.0, 1.0], sendo que o valor 0.0 representa o valor lógico absolutamente falso e o valor 1.0 representa o valor lógico absolutamente verdadeiro. Dentro deste contexto, pode-se verificar que a afirmação "o sistema está muito pesado", poderia ser traduzida para um valor de pertinência de 0.8, o que denotaria uma condição de baixa performance. Neste exemplo, "muito pesado" caracteriza um valor típico de uma variável lingüística, ou seja, em lógica difusa (ZADEH e YAGER,1994), permite-se que os valores dos atributos que descrevem uma determinada

situação sejam “vagos”. Neste caso, valores típicos para uma variável lingüística sobre performance poderiam ser: leve, moderado, pesado, muito pesado, afundando.

Segundo Reed (2001), aplicações que estão surgindo atualmente apresentam um padrão muito irregular de chamadas de serviços de entrada/saída. Como as interações entre estas aplicações e o software do sistema de arquivos mudam durante e entre várias execuções da mesma aplicação, é difícil ou impossível determinar uma configuração ótima de entrada/saída. Também é difícil estaticamente configurar sistemas de *run-time* e políticas de gerenciamento de recursos.

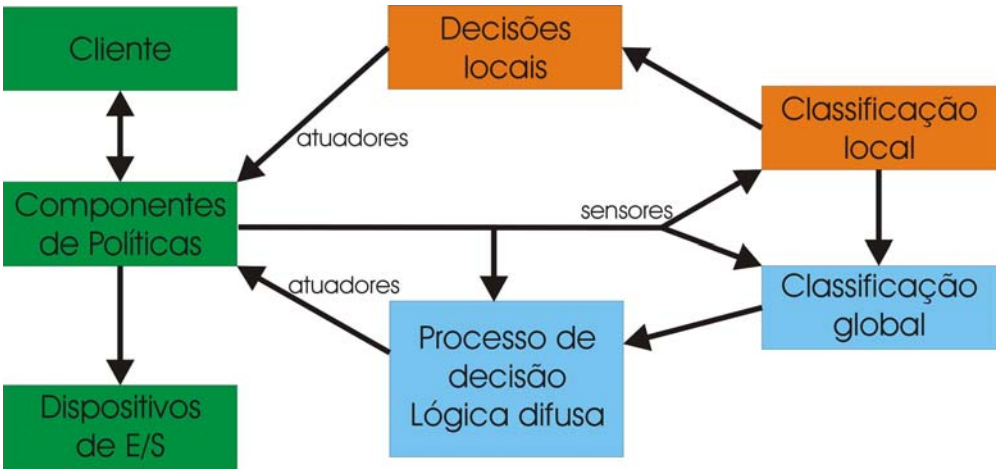


Figura 26 - Classificador de padrões de acesso de E/S em PPFSII.

FONTE: Adaptado de Reed (2001).

A Figura 26, apresenta o fluxo de informações através do mecanismo de decisão utilizando lógica difusa. Um controlador baseado em conjuntos difusos representa as propriedades semânticas de cada entrada (sensor) e saída (atuador). As variáveis são processadas através de um conjunto de regras de produção difusas que mapeiam os valores de entrada para o espaço de saída através de uma coleção de conjuntos difusos.

Além desta técnica, para suportar a tarefa de classificação automática e qualitativa do uso de recursos foi desenvolvido um conjunto de redes neurais artificiais e modelos escondidos de Markov.

Simitci (2000) descreve um sistema baseado em regras difusas para *striping* adaptativo de arquivos através de múltiplos discos. *Disk striping* é uma técnica que combina o espaço disponível de vários discos e divide os dados de forma a otimizar o acesso paralelo aos mesmos. A base de regras assenta-se em um modelo de filas de

contenção de disco que inclui padrões de acesso de E/S das aplicações e parâmetros de hardware de disco e de interconexão dos mesmos. Em situações de baixa demanda, a base de regras atua menos agressivamente para maximizar a eficiência total. Esta solução foi incorporada na segunda geração do sistema **PFSII** (*Portable Parallel File System*). Ele emprega sensores em software para capturar dados sobe atividades de E/S em tempo real. Embasado nesses sensores, utiliza procedimentos de decisão para selecionar dinamicamente políticas de prefetching, caching e striping. Por último, lança mão de atuadores para implementar as políticas de atuação.

Kandel, Zhang e Henne (1998), por outro lado, sugerem que a técnica de inferência difusa pode ser aplicada nas seguintes situações: seleção de um algoritmo de escalonamento útil, avaliação da performance geral ou avaliação de performance de algum módulo em particular . Neste trabalho é descrita mais detalhadamente a aplicação dos conceitos difusos em gerenciamento de processos, gerenciamento de estratégias de armazenamento e gerenciamento de arquivos.

Segundo Kandel,Zhang e Henne (1998), a solução apresentada poderia atender à seguinte suposição:

> *"Procura-se um algoritmo de escalonamento relativamente eficiente para tomada de decisões com múltiplos objetos e múltiplos fatores. Poder-se-ia tomar como um dos fatores envolvidos, a taxa de troca de contexto em determinado período. Outros fatores poderiam ser: tempo de execução, quantidade de recursos alocados, etc."*

Sendo este um problema de difícil solução, uma vez que não pode ser precisamente descrito através de funções de mapeamento matemáticas, funções de mapeamento difusas poderiam ser usadas para encontrar soluções relativamente satisfatórias.

A questão relativa à performance de soluções utilizando lógica difusa é abordada por Patki (1996), o qual afirma que, a partir do desenvolvimento de co-processadores difusos em hardware, o processo de *fuzification*, inferência e *defuzification,* passa a ser realizado de forma muito rápida. Portanto, a visão inicial, segundo a qual sistemas baseados em lógica difusa eram soluções especializadas em software, está mudando. Ainda segundo Patki e Raghunathan (1996), espera-se que todo um novo espectro de aplicações venha a emergir. Até o momento isso era impraticável, dadas as limitações das técnicas convencionais baseadas em lógica tradicional. Também se espera que novos

chip sets venham a ser projetados especificamente adaptados para suportar operações envolvendo lógica difusa.

Segundo Patki e Raghunathan (1996),

> *"O enfoque convencional de arquitetura de hardware (Von Neumann/ Processamento paralelo) bem como dos sistemas operacionais e softwares aplicativos não serão suficientes para atender as necessidades de tecnologia de informações no contexto de prover informações para as grandes massas".*

Um outro aspecto que está começando a ser abordado com maior ênfase na literatura, e que certamente apresentará um grande impacto na forma como os computadores são usados, refere-se à aplicação de inferência difusa em interfaces com o usuário. Isso ocorrerá não só em termos de gerenciamento de arquivos, como referido por Dalal et al. (1980), mas também no sentido de incorporar facilidades de respostas do sistema para o usuário em termos de variáveis lingüísticas e não sob a forma de mensagens curtas e códigos de erros, como ocorre hoje.

4.2.4 Redes Probabilísticas

Segundo Horvitz et al. (1998) o laboratório da Microsoft Research vem pesquisando métodos de raciocínio em situações de incerteza sobre as intenções de usuários de software. No centro das pesquisas está o projeto **Lumière**, que consiste no desenvolvimento de protótipos de modelos que objetivam capturar os relacionamentos entre metas e necessidades de um usuário, além de representar observações sobre o estado dos programas, seqüências de ações ao longo to tempo e palavras utilizadas por usuários em consultas.

O projeto **Lumière** utiliza a tecnologia de redes de Bayes e diagramas de influência com os seguintes objetivos:

- A construção, a partir de ações observadas, de modelos probabilísticos para raciocínio sobre metas de usuários (variáveis no tempo);
- Obter acesso ao fluxo de eventos de aplicações de software;
- Desenvolver uma linguagem para transformar eventos do sistema em variáveis observáveis, representadas em modelos probabilísticos de usuários;
- Desenvolver perfis persistentes para detectar alterações na experiência do usuário e;
- Desenvolvimento de uma arquitetura para que permita a construção de interfaces inteligentes com o usuário.

Com base na observação dos costumes dos usuários são calculadas as distribuições de probabilidades de comportamentos, tendo em vista alimentar o projeto de serviços automáticos e/ou assistentes e orientar a alocação de recursos do sistema operacional de forma adequada (HORVITZ et al.,1998).

Segundo Horvitz et al. (1998), resultados das pesquisas do projeto **Lumière** serviram como base para a construção do assistente (*Office Assistant*) presente no pacote Microsoft Office'97.

4.2.5 Mapas Cognitivos Difusos *(Fuzzy Cognitive Maps)*

Segundo Mohr (1997), os *fuzzy cognitive maps* (FCM) são ferramentas que combinam elementos de lógica difusa e redes neurais. Geralmente são implementados como dígrafos onde os nodos representam conceitos qualitativos e as conexões entre eles são representadas pelas influências causais. Um nodo conceitual possui um estado numérico, o qual denota a medida qualitativa de sua presença no domínio conceitual. Um valor alto indica que o conceito está presente. Um valor negativo ou, em alguns casos, um zero, indica que o conceito não está ativo ou relevante no domínio conceitual.

A técnica de FCM geralmente é utilizada em domínios envolvendo redes complexas de relacionamento (particularmente de *feedback*) e onde as medidas de influência entre os nodos não estão disponíveis. A técnica é de implementação simples e, por possuir elementos de lógica difusa, permite a adoção em domínios onde há incerteza (MOHR,1997).

Siraj, Bridges e Vaughn (2001) descrevem o uso de FCM e bases de regras difusas para aquisição de conhecimento causal (e para suporte ao processo de raciocínio sobre estas bases) no desenvolvimento de um sistema de detecção de intrusões em um ambiente de rede local. A função do sistema denominado *Intelligent Intrusion Detection System* – **IIDS,** é detectar e diagnosticar tentativas de afetar a integridade do sistema (sistema operacional e aplicações). Para tanto, são usados vários tipos de sensores, cada um dos quais provendo diferentes tipos de informações sobre algum aspecto do sistema que está sendo monitorado. Os dados dos sensores podem ser analisados sob óticas diferentes. Por exemplo, módulos de detecção de mau uso (*misuse*) procuram por padrões de ataques, enquanto módulos de detecção de anomalias procuram por desvios nos padrões normais de comportamento.

Na arquitetura IIDS, módulos de detecção de anomalias monitoram o trafego de rede para medir o grau de normalidade de transmissão dos dados. Múltiplos sensores de detecção investigam os dados auditados. Estes módulos de auditoria empregam técnicas de *data mining* com enfoques diferentes, como regras de associação difusas e episódios de freqüência difusos. Estes módulos apresentam, como saída, medidas difusas de normalidade sobre os dados auditados. Uma das principais funções da máquina (*engine*) de decisão é a de analisar o efeito combinado das saídas dos módulos de detecção de anomalias para determinar o nível de alerta da rede. Com este propósito, são usados FCM para verificar o relacionamento causal entre as medidas difusas que influenciam o nível de alerta da rede. Para detecção de mau uso (*misuse*), o IIDS usa o mecanismo de detecção baseado em regras que executam em cada *host* da rede. Segundo os autores, a técnica de FCM auxilia a prevenir alguns tipos de problemas de extração de conhecimento freqüentemente encontrados em tradicionais sistemas baseados em regras (SIRAJ, BRIDGES e VAUGHN ,2001).

Segundo Gürer et al. (2001), o uso combinado de diferentes técnicas de Inteligência Artificial para a solução de problemas relacionados ao gerenciamento de falhas de rede, pode produzir melhores resultados do que o emprego de uma técnica em particular. Por exemplo, métodos probabilísticos, tais como redes neurais ou redes probabilísticas, são apropriados para correlação, enquanto métodos simbólicos, como Raciocínio Baseado em Casos (RBC) ou Sistemas Especialistas (SE), são apropriados para identificação de falhas. Na área de correção de falhas, o uso de técnicas como RBC, SE ou Sistemas Inteligentes de Planejamento, permite desenvolver planos de ação para a correção de falhas identificadas e verificadas previamente.

4.3 Outras técnicas utilizadas

Além do emprego de técnicas de inteligência artificial na instrumentalização de sistemas operacionais, outras técnicas têm sido utilizadas com a finalidade de melhorar os níveis de qualidade dos mesmos.

Nesta seção são considerados os seguintes exemplos:

- Agentes.
- Visualização de informações.
- Análise de perfil.

Finalmente são apresentadas outras pesquisas que também possuem o mesmo propósito.

4.3.1 Agentes

Segundo Kalakota e Whinston (1996),

> *"Em 1988 a Apple Computer apresentou um vídeo sobre sua visão de um Knowledge Navigator*[34] *no qual um professor vivendo no ano de 2010 e utilizando um laptop o qual apresentava um agente de software (uma face masculina adornada com uma gravata) denominado Phil. Entre outras coisas, o agente auxiliava o professor em atividades de resposta a e-mails e interação com colegas em locais distantes. Além disso, Phil era capaz de manter a agenda do professor, além de coletar e analisar dados de pesquisa. A forma de comunicação com o agente dava-se através de comandos de voz (utilizando tecnologia de reconhecimento de voz) e uma interface de tela sensível ao toque (touch screen) a partir da qual o professor selecionava opções de menu".*

Em 1993, a Apple criou o primeiro produto comercial utilizando interface com agentes (*on-screen agents*) gráficos. Através da tecnologia de reconhecimento de voz, os agentes eram programados para entender um conjunto de comandos de computador como "*print*" ou "*restart*". Para criar uma ilusão de que o agente "entendia" o usuário, era gerado um som semelhante ao estalo da língua e o agente piscava para o usuário. Se a mensagem por algum motivo não podia ser decodificada, o agente "falava" para o usuário: "Pardon me?"(KALAKOTA e WHINSTON,1996).

Outras companhias também contribuíram para a evolução da tecnologia de agentes. Em 1991 a **HP (***NewWave***)** incorporou suporte a tecnologia de agentes para automatização de tarefas de *work-flow,* tais como roteamento de formulários. Esta tecnologia foi utilizada no desenvolvimento de um sistema de automação de escritórios desenvolvida pela NCR, denominada **Cooperation**. Mais recentemente, a IBM desenvolveu **Charlie**, um sistema tri-dimensional de agentes gráficos com alto grau de realismo. A Microsoft também desenvolveu a sua versão de agente de interface, denominada **Bob** (KALAKOTA e WHINSTON,1996).

A proposta dos ***soft bots,*** agentes que interagem com ambientes de software e executam tarefas de alto nível em favor do usuário, usa a mesma filosofia adotada em linguagens funcionais, permitindo ao usuário especificar o que ele deseja e ficando a cargo do sistema decidir como executar a operação. Uma vez informada ,a meta pelo usuário, o sistema constrói uma seqüência de ativação de primitivas a serem usadas e

[34] Navegador de conhecimento.

executa-as, tomando decisões intermediárias baseado no contexto e nas condições existentes (ETZIONI et al.,1995).

O projeto **St.Bernard** (SEGAL,1992) descreve a utilização de *soft bots* cuja função é localizar arquivos em um sistema de arquivos **Unix**. A estratégia adotada é baseada na técnica *de satisficing search*. De acordo com Segal (1992), esta técnica consiste em tentar minimizar o custo de uma busca a um arquivo através do cálculo da probabilidade de que ela terá sucesso se realizada a partir de um nodo determinado.

Os resultados obtidos demonstraram que a técnica aumenta em aproximadamente três vezes a eficiência na localização de arquivos em diretórios, quando comparada à técnica de busca exaustiva tradicionalmente utilizada nos sistemas operacionais (SEGAL,1992).

Etzioni et al.(1995) discutem a questão das interfaces com o usuário. Segundo eles, apesar da atual evolução das interfaces, não é simples para o usuário formalizar consultas complexas ao sistema. Normalmente isto é feito através da combinação de comandos e menus, tendo em vista que os projetistas não podem antecipar cada necessidade dos usuários. Sistemas que disponibilizam acesso a interpretadores de linha de comando e/ou *shells*, que suportam scripts, permitem a construção de consultas mais elaboradas, mas requerem um maior nível de especialização dos usuários.

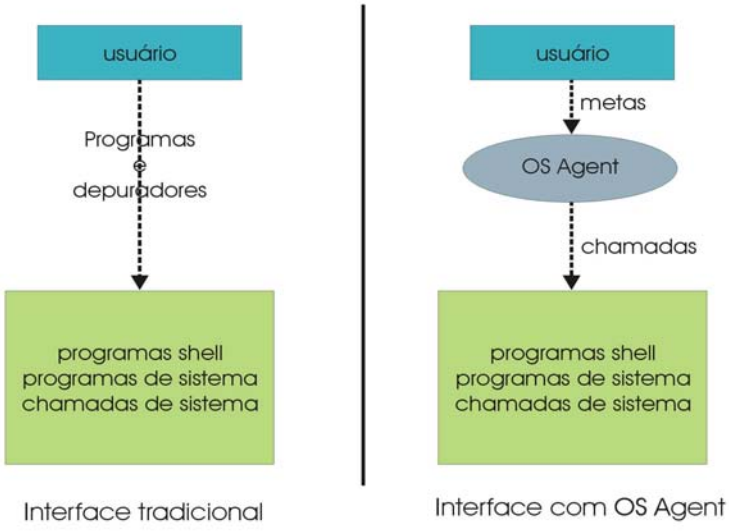


Figura 27 - Arquitetura de um OS Agent.

Comparando-se o lado direito da Figura 27 com o modelo evolutivo dos sistemas operacionais, percebe-se claramente a aplicação da fase 2 discutida na seção 2.6(Pág.85).

4.3.2 Visualização de informações

A área de visualização de informações utiliza sistemas gráficos 3D e técnicas de animação interativa para simular o reconhecimento de padrões e estruturas em informações armazenadas. Isto ocorre através da exploração do sistema de percepção humana de forma similar àquela utilizada em técnicas de visualização científica, a qual permite aos cientistas identificar padrões em grandes coleções de dados. Enquanto a área de visualização científica trabalha com dados geralmente obtidos através de simulações de processos físicos, a área de visualização de informações trabalha com a estrutura da informação inerente em grandes espaços de informação (ROBERTSON et al.,1998).

A seguir são apresentados dois projetos que servem para caracterizar o problema e como as soluções estão sendo encaminhadas.

4.3.2.1 Fisheye menus

Bederson (2000) descreve um esforço no sentido de aplicar técnicas tradicionais de visualização gráfica baseadas em distorção, do tipo *fisheye*(Figura 28), para tornar mais eficiente o mecanismo de seleção de itens de menus quando a lista de opções é muito grande. A solução adotada implica em incrementar o tamanho da fonte utilizada para escrever os itens mais freqüentemente utilizados e diminuir proporcionalmente a fonte dos menos usados.

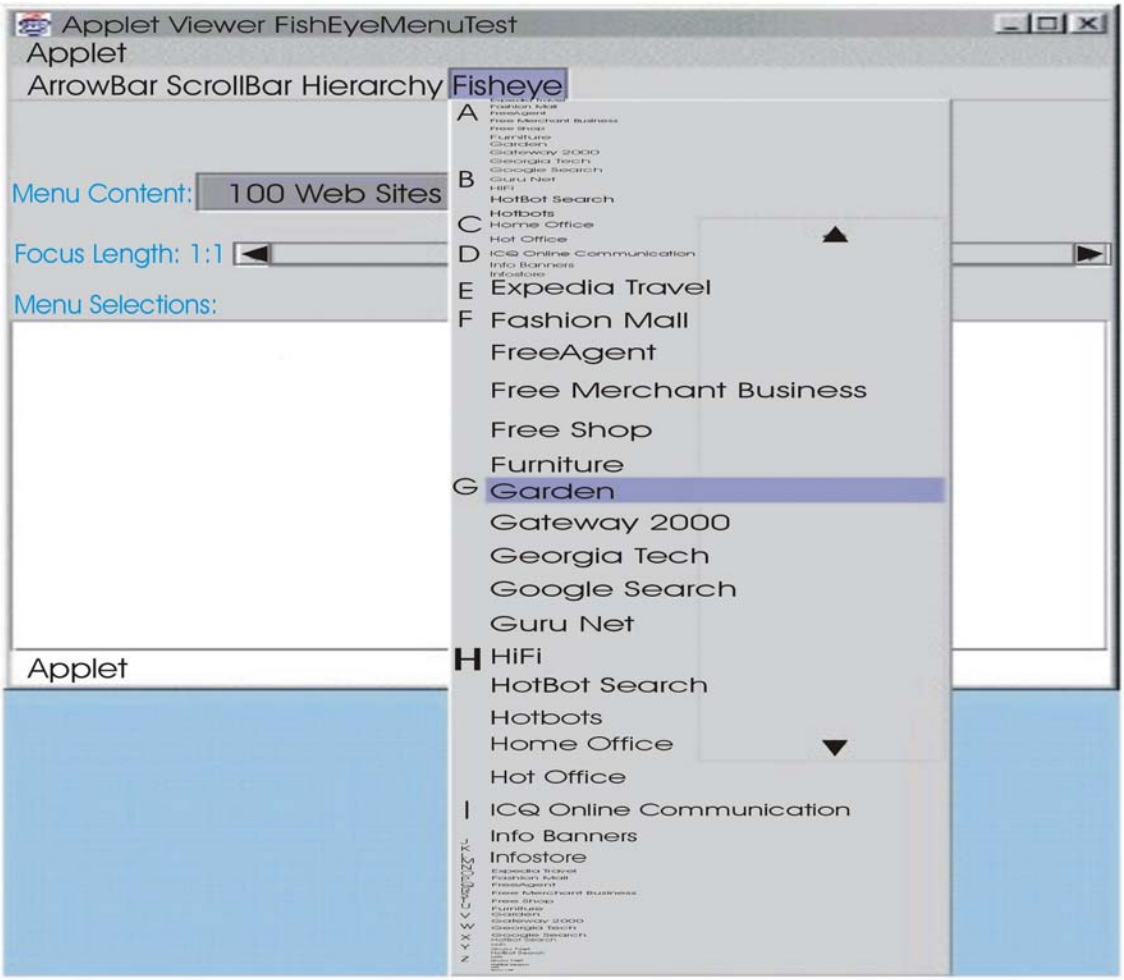


Figura 28 - Fisheye: menu apresentando os 100 mais populares sites web retirados da lista dos mais populares da revista PC Magazine.

FONTE: Adaptado de Bederson (2000).

Segundo Bederson (2000), a seleção de itens de menu é uma área bem conhecida. O padrão usual é agrupar os itens em lugares consistentes, utilizando nomes usuais do domínio do problema. Contudo, com o surgimento da *Web* e de aplicações de comércio eletrônico (*e-commerce),* tem se tornado comum o uso de menus para selecionar itens de dados, ao invés de selecionar operações como ocorria originalmente.

Alguns exemplos do uso de menus neste sentido são relacionados a seguir:

- Seleção de fontes (*Arial, times new roman*);
- Seleção de estados da federação;
- Seleção de países;
- Seleção de uma pasta ou arquivo no sistema de arquivos;
- Seleção de web sites a partir da lista de favoritos do navegador (*browser)*.

A estratégia de implementação está baseada no cálculo de uma função de grau de interesse para cada elemento a ser apresentado. A partir deste valor, são calculados os respectivos tamanhos das fontes de cada item de menu antes da apresentação para o usuário. Soluções semelhantes para este problema foram incorporadas, por exemplo, no sistema de apresentação de itens de menu dos aplicativos do sistema operacional Windows XP, o que corrobora o modelo cíclico descrito na seção 2.6(Pág.85).

4.3.2.2 Data Mountain

Robertson et al. (1998), no projeto **Data Mountain,** relata a solução proposta para o problema de gerenciamento efetivo de documentos através do uso de computadores . Segundo o autor, esta tem sido uma questão importante no que se refere ao projeto de interfaces gráficas com o usuário. O enfoque mais usual para a solução do problema envolve o uso de layouts de ícones distribuídos espacialmente em um ambiente 2D, os quais representam os documentos e pastas disponíveis. Robertson descreve uma técnica para gerenciamento de documentos que permite ao usuário distribuir os documentos do modo mais conveniente possível em um plano inclinado assentado em um ambiente tridimensional virtual, utilizando técnicas de interação 2D tradicionais (Figura 29).

Segundo Robertson et al. (1998), o aspecto importante é que todo código da aplicação foi desenvolvido utilizando as mesmas bibliotecas empregadas na implementação da versão OpenGL do sistema. Ou seja, esta solução também se enquadra no modelo cíclico descrito na seção 2.6(Pág.85).

Isdale (1998) apud Robertson et al. (1998) afirma que: "enquanto a lei de Moore prevê que a capacidade dos computadores dobra a cada 18 meses, a capacidade dos sistemas gráficos 3D está dobrando muito mais rápido – cerca de 6 a 8 meses". E ele continua:

> *"A Microsoft prevê taxas de 200 milhões de polígonos por segundo por volta de 2001 e em 2011 haverá sistemas 1000 vezes mais rápidos. Há cerca de 30 companhias trabalhando em sistemas de textura 3D de baixo custo e alta performance, orientadas para otimizar outras tarefas que não visualização de informações. Isto pode ser verificado pela ausência de suporte a manipulação de textos em 3D. Nós precisamos de camadas que mapeiem técnicas visuais em fontes de informação".*

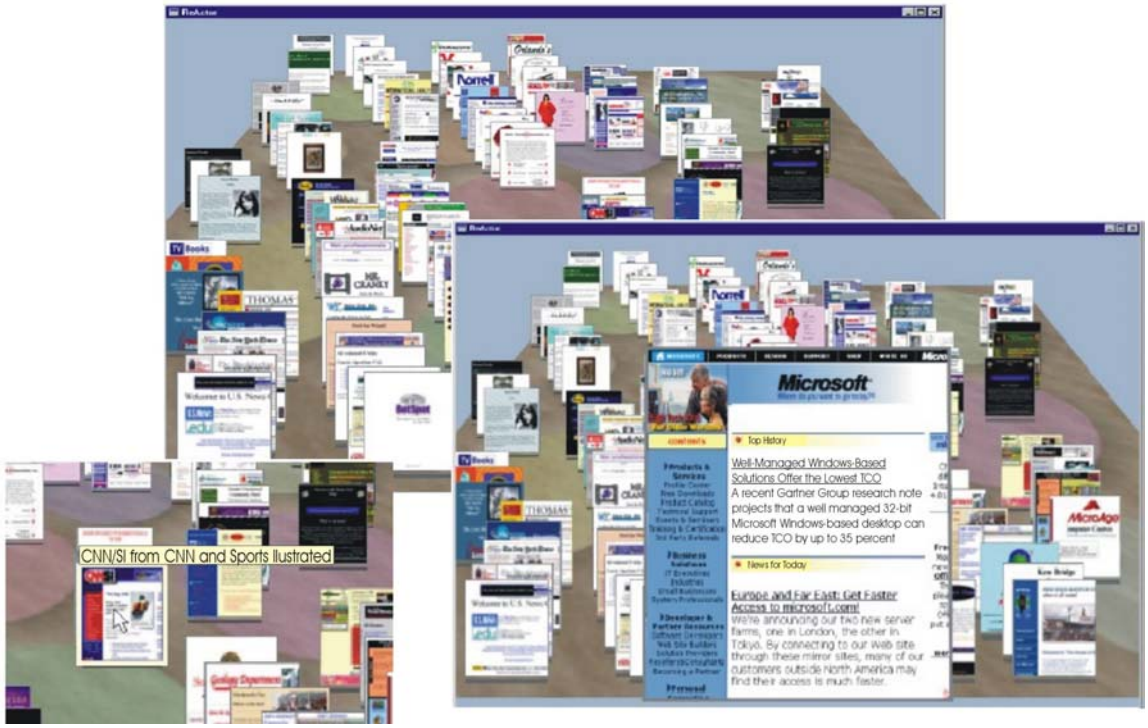


Figura 29– Data Mountain: exemplo de seleção de um documento.

FONTE: Adaptado de Robertson (1998).

Isdale (1998) apud Robertson et al. (1998) identificou os 2 principais desafios para a comunidade de visualização: *usability* e *perceptualization*. Além disso, ele identificou 4 áreas que devem ser trabalhadas na visualização de informações, quais sejam:

- Explorar o espaço de projeto das técnicas de visualização;
- Áreas de trabalho (*workspaces*) colaborativas (e, portanto, compartilhadas) e persistentes;
- Exploração de percepção cognitiva, e;
- Codificação de conhecimento visual.

A partir destas constatações, percebe-se a necessidade de uma nova proposta para fazer frente às demandas específicas da área de visualização de informações.

4.3.3 Análise de perfil

Segundo Arbanowski e Steglich (2001):

> *"uma tendência para os futuros ambientes computacionais que já é reconhecida pelo mercado consumidor é a ampliação (augmentation) dos ambientes tradicionais. Tanto os dispositivos como as facilidades do dia-a-dia deverão ter suas capacidades aumentadas através do uso de sensores (GPS em telefones celulares) e interfaces de voz e reconhecimento de escrita manual (livros eletrônicos – e-books aplicações baseadas em Interactive Voice Response -IVR)".*

A partir destes módulos acessórios é possível a aquisição de informação contextual e, a partir daí, o desenvolvimento de aplicações baseadas em contexto. Portanto, segundo Arbanowski e Steglich (2001),

> *"Serviços orientados por contexto são aqueles que podem ser adaptados às preferências individuais de cada usuário, em seu modo particular de interação com o sistema de comunicação".*

Arbanowski e Steglich (2001), descrevem um framework (Figura 30) que possibilita a criação e execução de serviços dependentes de contexto (*context aware*) que necessitam de personalização.

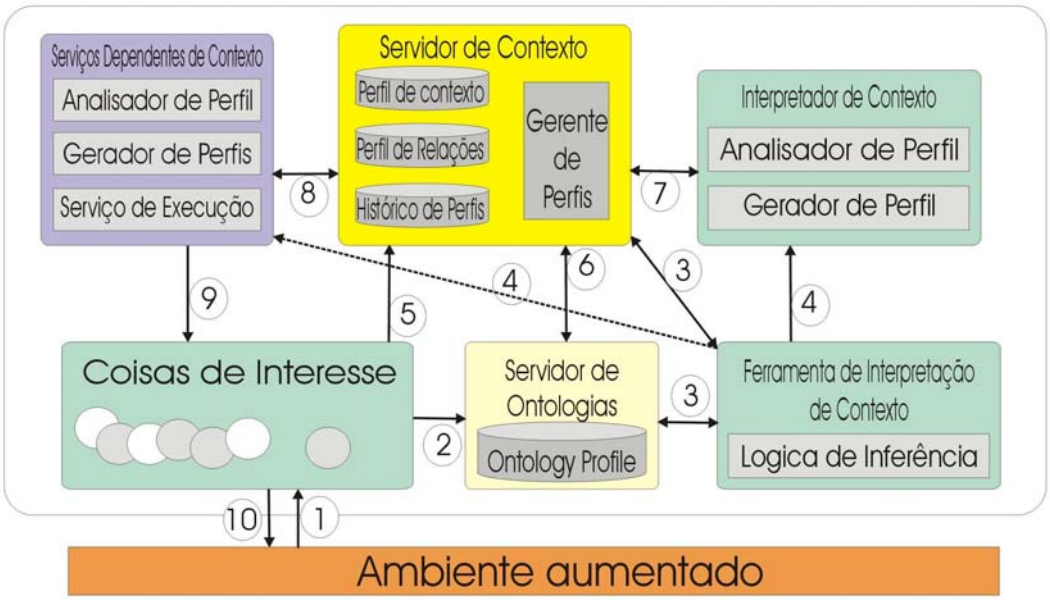


Figura 30– Infra-estrutura para aquisição de informações contextuais.
FONTE: Adaptado de Arbanowski (2001).

A estrutura é responsável pela gerência de artefatos envolvidos no contexto, pela identificação de causalidades entre eles, baseando-se em dados ambientais observados, pelo controle dos serviços por eles oferecidos e pela conversão das estruturas de dados e operações, de forma a permitir o compartilhamento entre serviços diferentes. Os principais blocos funcionais desta estrutura são apresentados a seguir:

- Coisas de interesse (ToI– *Things of Interest*) obtêm informação contextual que ocorre no ambiente aumentado, tais como detectar temperatura ou qualquer entrada de usuário. ToI gera uma representação em XML– *Extended Markup Language.* Antes de enviar qualquer ocorrência para o servidor de contexto, a ToI tem que registrar a ontologia (funções e dados

com correspondência semântica a descrição legível por humanos) correspondente no servidor de ontologias (*ontology server*).

- O servidor de contexto (*context server*) é responsável pela gerência de toda a informação contextual, incluindo alterações temporais (histórico). Ele gerencia os perfis: Contexto, Relação, Perfil e Histórico. A cada registro é associada uma classificação semântica (ou categorização), a qual permite a localização e endereçamento da informação de perfil.
- Interpretador de Contexto (*context interpreter*) solicita ao servidor de contexto que o notifique da ocorrência de algum evento em determinada categoria.
- O serviço contextual (*context-aware service*) é o componente que utiliza toda a informação provida pelo servidor de contexto. Por exemplo, se um serviço pode solicitar que o servidor de contexto informe sobre qualquer alteração de localização e tome as providências necessárias para adequar-se a ela.
- Interpretador de contexto (*context interpreter toolkit*) apresenta uma interface gráfica com o usuário, a partir da qual condições e ações para determinados contextos podem ser facilmente definidas pelo usuário.

4.3.4 Ferramentas baseadas em histórico

Segundo Wexelblat e Maes (1997),

> *"A informação digital carece de histórico, de textura, de riquezas que advém do uso por várias pessoas ao longo do tempo. O mundo físico é rico em pistas e indicações das quais nos valemos constantemente, nossos dados permanecem estéreis. Quando abrimos um documento ou visitamos uma página web, é como se nós fossemos os primeiros e as únicas pessoas a manipular estes dados. Nenhum dos problemas resolvidos, nada sobre o que foi feito, nenhuma das pessoas envolvidas é visível".*

O projeto **FootPrint,** desenvolvido por Wexelblat e Maes (1997), baseia-se no histórico das interações entre pessoas e objetos. Foram identificadas pelo menos três categorias de representações:

- Seqüências de ações, relacionamentos entre elementos sobre os quais as pessoas atuaram e estruturas resultantes;
- *Temporal collage* – múltiplos estados ou os processos que os produzem são apresentados diretamente. Isto é particularmente útil quando a mudança é muito rápida ou muito lenta para ser percebida;
- Estados recorrentes opostos: a história pode ser inferida a partir de contrastes entre o estado presente, lembranças de estados passados e expectativas de estados futuros.

Wexelblat e Maes (1997), identificaram pelo menos 4 áreas onde a técnica pode ser aplicada:

- Discernir que tipo(s) de experiência(s) um espaço de informação pode oferecer;
- Expressar diferentes formas de apresentação de um documento hipertexto (*web*);
- Entendimento de sentidos diferentes e ortogonais em um documento complexo;

- Melhorar a compreensão, fornecendo múltiplas formas de atingir a mesma informação.

Narayanan, Flinn e Satyanarayanan (2000), apresentam um outro enfoque para uso de informações históricas. Neste caso, as informações históricas são utilizadas para monitorar, registrar (*log*) e predizer o consumo de recursos das aplicações no sistema operacional **Odyssey** como uma função de fidelidade. Fidelidade "é uma noção específica da aplicação relativa à qualidade (*goodness*) de um resultado calculado ou objeto de dados: por exemplo, o fator de precisão de um cálculo de ponto flutuante".

Segundo Narayanan, Flinn e Satyanarayanan (2000),

> *"O enfoque adotado é empírico na medida em que a estratégia de solução envolveu a amostragem do espaço de fidelidade a partir da execução de cada aplicação alvo com diversos valores de fidelidade. A cada execução os registros de consumo de energia associada a cada ponto amostrado são armazenados. A última etapa envolveu a utilização de algoritmos de aprendizagem de máquina para fazer as predições baseadas nestes exemplos armazenados".*

No caso do sistema **Odyssey** as decisões sobre adaptação são tomadas conjuntamente entre o sistema operacional e as aplicações. O sistema, como arbitrador de recursos compartilhados, tem melhores condições de determinar a disponibilidade de recursos. Contudo, as aplicações são as únicas entidades que podem decidir como se adequar a determinada situação e devem ter liberdade para especificar suas próprias políticas de adaptação. Segundo Noble (2000), este modelo é denominado: modelo colaborativo de responsabilidade.

4.3.5 Outras pesquisas relacionadas

Coady, Kiczales e Feeley (2000), apresentam uma solução que utiliza o conceito de *Aspect-Oriented Approach* com o intuito de simplificar o código do sistema operacional. No experimento descrito no trabalho foram re-implementados os módulos para tratamento de *page-faults* e requisições de leitura ao sistema de arquivos, utilizando uma linguagem hipotética denominada **AspectC**[35]. O protótipo desenvolvido simplificou o código de tratamento de *page-faults* e criou um contexto que facilitou a compreensão dos algoritmos e estruturas de dados que implementam o sistema de gerenciamento de memória virtual e de *prefetching* de arquivos do sistema **FreeBSD**.

[35] Segundo os autores, uma variação da linguagem AspectJ para C.

O projeto **Choices** (CAMPBELL, HINE e RUSSO,1992) descreve um modelo de sistema, orientado a objetos, que incorpora a noção de configuração de sistemas operacionais para permitir sua adequação a diferentes configurações de hardware e software. O *kernel* é composto por uma coleção de objetos dinâmicos e suporta: uma interface de aplicação baseada em objetos local e distribuída, herança e polimorfismo. Os recursos do sistema, mecanismos e políticas são representados como objetos que pertencem a uma hierarquia de classes. Subsistemas são projetados como *framework* de classes. As pesquisas baseadas nesse projeto têm possibilitado contribuições significativas em áreas como sistemas de arquivos adaptativos e migração rápida de processos. Uma proposta de construção do sistema, utilizando uma estrutura de micronúcleo, originou o sistema ***Micro Choices***, com características como o suporte para a carga dinâmica de código nativo e para Java *byte codes* e como o suporte das aplicações multimídia.

Projetos de sistemas de arquivos distribuídos (ANDERSON et al.,1996; KISTLER e SATYANARAYANAN,1992) têm demonstrado sucesso para suportar os acessos a dados com uma forma transparente de localização. O projeto **WebOS** (VAHDAT et al.,1997) tem por meta disponibilizar recursos de computação distribuídos por meio da web. O núcleo deste sistema está baseado no **WebFS**, sistema de arquivos global capaz de disponibilizar acesso seguro aos dados.

Questões de autoconfiguração e adaptação para situações de flutuação de disponibilidade de serviços são abordadas por Kon (2000) e Neefe (1996). Por exemplo, o projeto **2K (**KON,2000) para ambientes heterogêneos e distribuídos inclui um novo modelo de gerenciamento de dependências em sistemas distribuídos. Possui um ORB (*Object Request Broker*) reflexivo que possibilita políticas dinâmicas de segurança e monitoramento. Também tem um modelo para suportar ambientes de usuários e gerência de recursos em sistemas heterogêneos, além de um protocolo orientado para aplicações multimídia em *wide-area* e uma infra-estrutura para reserva e configuração de serviços multimídia.

Recentes avanços na tecnologia de componentes permitem a construção de sistemas complexos através de sua composição. Contudo, ainda é difícil desenvolver sistemas eficientes, confiáveis e dinamicamente configuráveis, visto que os componentes freqüentemente são desenvolvidos por equipes diferentes, que utilizam metodologias

diferentes, o que naturalmente conduz a falhas inesperadas, comprometendo a confiabilidade do produto final.

Segundo Friedrich (2001), software baseado em componentes tem sido usado cada vez mais como meio de construção de sistemas de software a partir da montagem de blocos prontos. **Composes** é uma abordagem que pretende melhorar o processo de construção de software confiável para sistemas embutidos. O objetivo principal do projeto é aplicar o conceito de componentes no ambiente da computação embutida, especialmente no que diz respeito ao suporte operacional.

Na mesma linha da componentização, Stets, Hunt e Scott (1999) apresentam um projeto denominado **COP** (*Component-based Operating system Proxy*), o qual ataca a questão do formato monolítico e procedural das APIs de um sistema operacional. Segundo os autores, o formato atual das APIs é muito rígido, uma vez que não pode ser modificado sem comprometer as aplicações já desenvolvidas (legadas). Por exemplo:

> *"A especificação UINX 98 (endossada pela IBM, Sun e NCR) lista 21 chamadas reservadas para suporte a código legado. Muitas destas já foram superadas por novas e mais poderosas chamadas (ex. função de gerenciamento de sinais,* ***signal****(), foi substituída por uma mais poderosa* ***sigaction****()). A principal API da família Microsoft denominada Win32, contem cerca de 50 chamadas cujo objetivo é manter compatibilidade com a versão Windows 3.1 . Embora elas componham uma fração relativamente pequena do numero total de funções disponíveis, este fato contribui para o aumento na complexidade de implementação dos serviços bem como no aumento de overhead de tempo de execução (run-time)" Stets, Hunt e Scott(1999).*

Esta rigidez resulta em um aumento na complexidade da interface e do código que implementa as funções, com conseqüentes efeitos tanto em termos de performance como de facilidade de manutenção dos sistemas.

Na área de reflexão computacional, o trabalho de Kon (2000), relata a construção de um módulo reflexivo para suportar configuração automática de sistemas distribuídos baseados em componentes. O argumento do autor é que os sistemas atuais, particularmente aqueles baseados em componentes, não possuem suporte adequado para representar as dependências entre componentes, dificultando operações tais como: configuração automática, reconfiguração dinâmica, tolerância a falhas e adaptação.

O projeto **Tapestry** (ZHAO e KUBIATOWICZ,2001) explora mecanismos e políticas para sistemas paralelos. Mecanismos semelhantes aos incorporados no sistema **Choices** foram projetados para permitir a reconfiguração de componentes do sistema operacional.

Assim é possível suportar novas arquiteturas paralelas e aplicações. A substituição de políticas do sistema operacional (escalonamento de tarefas, gerência de mensagens ou gerência de arquivos), possibilita o estudo dos efeitos que elas produzem em um *framework* consistente.

Na área de controle de recursos há o projeto **Rialto** (JONES,REGEHR e SAROIU,2001), cujo mecanismo de escalonamento permite estabelecer garantias de qualidade de serviço de CPU no **Windows 2000**. O projeto **Eclipse** (BRUNO et al.,1998) também possui um mecanismo de escalonamento com qualidade de serviço. Esse é um sistema baseado em **Plan9** (DORWARD et al.,1996). A proposta **Nemesis** (STEVEN,1999) é caracterizada por apresentar uma arquitetura de sistema verticalmente estruturada, com garantias de qualidade dos serviços controlados pelas aplicações. O projeto **Scout** (HARTMAN et al.,1994) introduz técnicas de gerenciamento do consumo de recursos, através da metáfora de caminhos (*paths*) sobre a qual o sistema está baseado. Além disso, a capacidade de auto-configuração no nível das redes locais também tem sido pesquisada (RODEHEFFER e SCHROEDER,1991).

Há algumas propostas de solução para problemas de mobilidade entre arquiteturas heterogêneas. Um enfoque supõe a utilização de um interpretador transparente, como o sistema **Aglets** (LANGE e OSHIMA,1997), que utiliza a **JVM**. Um segundo enfoque é baseado em código nativo especial do sistema operacional (STEENSGAARD e JUL,1995). Ele permite a migração em certos pontos de sincronização, denominados paradas de ônibus (*bus stops*). No entanto, ambas as soluções são limitadas por serem específicas para uma única linguagem. Barak e La'adan (1998) investigam a questão da migração de processos para fins de balanceamento de carga em clusters **Unix**. Nesse estudo, por razões óbvias, os processos somente podem migrar entre máquinas homogêneas.

Certamente há outros projetos que poderiam ser relacionados. No entanto, em função do caráter dinâmico do processo, a cada momento surgem novas propostas, o que torna a tarefa de estudá-las e classificá-las virtualmente impossível. Contudo, as pesquisas relacionadas servem para caracterizar o esforço que vem sendo realizado na área tendo em vista atender as necessidades de aplicações para espaços inteligentes (direta ou indiretamente).

4.4 Abordagens adotadas em projetos de robótica

A presente seção tem por objetivo apresentar alguns conceitos fundamentais da área de Robótica que devem ser examinados antes de se abordar a solução proposta, uma vez que o software embarcado pode, em última instância, ser considerado como o sistema operacional do robô. Desse modo, esta seção explora alguns conceitos importantes utilizados nesta área, relativamente ao emprego do conceito de modelo de mundo, e discute as principais características deste enfoque, com vistas a uma adequação aos projetos de sistemas operacionais.

4.4.1 Modelo de mundo

> *"Um principio que norteia os projetos de robótica, independentemente de serem eles estruturais ou comportamentais, envolve o entendimento do ambiente dentro do qual o robô opera e as tarefas que devem ser executadas. Este enfoque ecológico em que as metas e arredores (surroundings) do robô têm muita influência no projeto, será um tema recorrente neste livro." Arkin (1999).*

As pesquisas em robótica[36] há muito tempo vêm trabalhando no desenvolvimento de arquiteturas tanto de hardware como de software, objetivando a construção de robôs móveis autônomos capazes de operar em ambientes complexos, parcialmente conhecidos e portadores de riscos, usando limitados recursos físicos e computacionais.

Arkin (1999) inicia o capítulo introdutório de seu livro "*Behavior-Based Robotics*" com a seguinte pergunta: " Se nós pudéssemos criar robôs inteligentes, como eles deveriam se parecer e o que eles deveriam ser capazes de fazer?". Segundo Arkin (1999), a "primeira resposta requer uma descrição da aparência (estrutura física) e do seu comportamento (performance). Contudo, a segunda parte da questão depende da primeira resposta".

O surgimento da área de Inteligência Artificial como campo distinto é geralmente associado à conferência de *Dartmouth Summer Research Conference,* ocorrida em agosto de 1955.

[36]Conjuntamente com a área de inteligência artificial (Arkin,1999).

Segundo Arkin (1999), na proposta original,

> *"Marvin Minski indica que uma máquina inteligente deveria tender a construir dentro de si um modelo abstrato do ambiente onde ela está colocada. Se fosse apresentado um problema, ela deveria primeiramente explorar as soluções dentro deste modelo abstrato interno do ambiente para depois tentar experimentos externos".*

Este enfoque dominou as pesquisas em robótica pelos 30 anos seguintes, durante os quais os pesquisadores da área de Inteligência Artificial desenvolveram forte dependência no uso de métodos de representação do conhecimento e de raciocínio deliberativo para a implementação de funções de planejamento robótico (ARKIN,1999).

A influência da área de Inteligência Artificial na área de Robótica até este ponto foi baseada nos preceitos de que o conhecimento e a representação de conhecimento são aspectos centrais para a construção de entidades inteligentes. Segundo Arkin (1999), "talvez isto tenha sido conseqüência imediata da preocupação dos pesquisadores da área de IA da época com aspectos de inteligência ao nível de seres humanos". Nesta época considerar formas de vida menores não era interessante.

A proposta de sistemas robóticos baseados em comportamento foi uma reação a estas antigas tradições. Segundo D'Inverno (1998),

> *"Brooks, em particular, reconheceu as limitações destes antigos enfoques os quais eram baseados na idéia de que um modelo de mundo completo poderia ser construído e manipulado. O sucesso destes sistemas foi possível porque os ambientes utilizados eram muito simples e podiam facilmente ser mapeados para algum modelo interno de representação".*

De acordo com Brooks (1987) apud D'Inverno (1998),

> *"Os sistemas inteligentes só podem ser construídos no contexto de participação em algum ambiente (world), ou seja, sistemas inteligentes devem ser localizados (situated) no mundo real o qual influenciará seu comportamento de forma imediata e direta. Além disso, estes sistemas devem estar associados (grounded) no mundo através de uma presença física (embodied), ou seja, devem possuir um corpo e experimentar o mundo diretamente".*

Portanto, a estratégia de modelagem baseada em comportamento apresenta pelo menos três características importantes:

- ***Situatedness*** – o robô é uma entidade situada e imersa (*surrounded*) no mundo real. Ela não opera sobre representações abstratas da realidade, mas pela realidade;
- ***Embodiment*** – um robô possui uma presença física (corpo). Esta realidade espacial tem conseqüências nas suas interações dinâmicas com o mundo, as quais não podem ser simuladas exatamente.

- **Emergência**: inteligência emerge a partir das interações do agente robótico com o ambiente. Ela não é uma propriedade do agente ou do ambiente isoladamente, mas o resultado do relacionamento entre ambos.

Segundo Schaad (1998), "localização é uma questão que envolve um agente compartilhando o controle sobre o fluxo de eventos com outra entidade denominada ambiente. Não há como analisar o agente dissociando-o de seu ambiente".

A referência a robôs autônomos imediatamente remete ao conceito de agente. Segundo Booker et al. (1999),

> *"O conceito de agente não é particularmente recente. No final da década de 70 surgiram os primeiros trabalhos em agentes móveis. Contudo, até a popularização da Internet e avanços nas tecnologias de rede e de processadores, não houve significativa necessidade de pesquisas na área. No entanto, nos últimos anos tem havido um grande investimento no desenvolvimento de plataformas e ambientes de desenvolvimento de agentes".*

No contexto do presente trabalho, consideram-se os seguintes conceitos:

Agente Autônomo: "um sistema localizado em e parte de um ambiente, o qual percebe (*senses*) alterações ambientais, e age sobre o ambiente a medida em que persegue sua própria agenda de tarefas. Isto é, o agente é estruturalmente encaixado (*coupled*) no ambiente" (MATURANA,1975 apud FRANKLIN**,** KELEMEN e MCCAULEY,1998).

Agente Cognitivo: "tais agentes autônomos de software quando equipados com facilidades cognitivas (interpretando-se genericamente) adquiridas a partir de múltiplos sensores, percepção, memórias de curto e longo prazos, atenção, planejamento, raciocínio, solução de problemas (*problem solving*), aprendizado, emoções, etc., serão denominados agentes cognitivos" (FRANKLIN, KELEMEN e MCCAULEY,1998).

É a partir das idéias de Brooks de presença física (*embodiment*) e localização (*situadeness*) que o conceito de agente passou a ser discutido com mais ênfase. A noção de agente possibilitou uma forma de estender os sistemas tradicionais com um grau extra de flexibilidade a ponto de torná-los mais robustos em face das mais variadas e imprevisíveis formas de interação. Este conceito é distinto daquele que tem prevalecido no processo de interação entre usuários e computadores em sistemas tradicionais, onde o usuário controla a execução (D'INVERNO, 1998).

Embora inicialmente tenha havido resistência a esta mudança de modelo, a noção de sentir e agir dentro do ambiente começou a tomar forma em pesquisas de Inteligência Artificial relacionadas à Robótica (ARKIN,1999).

Segundo Medeiros (1998) e D'Inverno (1998), os principais requisitos[37] a serem atingidos em projetos de robótica são:

- Reatividade ao ambiente,
- Comportamento inteligente,
- Integração de múltiplos sensores,
- Resolução de múltiplos objetivos,
- Robustez e confiabilidade,
- Adaptabilidade e raciocínio global,
- Mobilidade,
- Autonomia,
- Interação entre Agente-Agente e Agente-Humanos,
- Modularidade e flexibilidade,
- Facilidade de expansão.

Como em toda área de pesquisa, há algumas decisões de compromisso que devem ser consideradas em relação à autonomia e a complexidade dos projetos de robótica. Por exemplo, reduzindo-se o nível de autonomia, através da atribuição de tarefas complexas a agentes externos, pode-se simplificar o projeto em detrimento da capacidade de decisão do robô; reduzindo-se a complexidade do ambiente, através da introdução de pontos de referência, também diminui a complexidade das atividades de planejamento de rota, mas limita a capacidade de aplicação dos robôs; e finalmente, alguma solução de compromisso deve ser adotada para compatibilizar a necessidade de respostas rápidas em detrimento da assimilação de informação necessária ao aprendizado do robô (MEDEIROS, 1998).

Segundo Schaad (1998), esta questão pode ser analisada sob a seguinte perspectiva: assumindo-se que um agente situado não possui limites em sua capacidade de percepção e em seus recursos computacionais, seria possível programá-lo a fim de que:

- Perceba tudo sobre o mundo que deve ser conhecido;
- Atualize seu modelo interno perfeito de acordo com estas percepções;
- Construa um plano ótimo (seqüência de passos) em direção ao estado de metas, as quais, com certeza, conduzirão a conclusões pertinentes, baseadas na projeção de futuros estados e levando em consideração o estado atual das informações disponíveis;

[37] Cabe observar que os dois últimos requisitos referem-se a requisitos não-funcionais enquanto os primeiros referem-se a requisitos funcionais.

- Execute o primeiro passo nesta seqüência.
- Recomece novamente.

Se fosse possível construir tal agente, ele apresentaria duas propriedades cruciais em agentes situados:

- Reatividade: as ações sempre refletem as mais recentes alterações no ambiente – o agente reage a mudanças imediatamente e otimamente;
- Coerência: as ações pertencem a um plano que com certeza atinge suas metas. As ações não desfazem os efeitos de outras ações a partir da observação de restrições de ordenação.

Vias de regra tais agentes não existem, tendo em vista que:

- Percepção: em geral os agentes não possuem um conjunto completo de informações sobre o ambiente em função da natureza da percepção e da dinâmica do ambiente;
- Modelo de Mundo: não há recursos computacionais suficientes para construir e atualizar modelos de mundo precisos;
- Computação: não há recursos computacionais suficientes para construir planos ótimos em tempo real a cada alteração do mundo. A fase de planejamento geralmente consome muitos recursos e seu tempo de execução é raramente previsível.

4.5 Considerações Finais

> *"A programação de agentes situados é uma tarefa complexa, visto que a interação em um ambiente dinâmico, imprevisível e potencialmente grande, introduz problemas significativos. A maior parte deles está relacionada ao modo como os agentes usam os planos que determinam seu comportamento. Tradicionalmente, os planos eram executados literalmente; o agente fazia exatamente o que o plano estabelecia. Isto impunha uma enorme responsabilidade no indivíduo que construía os plano,s porque ele tinha que antecipar todos os possíveis caminhos pelos quais o processo de interação do agente com o meio poderia conduzi-lo. Hoje, está claro que um agente deve ser capaz de interpretar planos de uma forma mais sensível e dependente de contexto; ele deve ser capaz de improvisar, interromper, retomar e seqüenciar atividades para ativamente alimentar-se de informações e usar a situação atual para eliminar referências ambíguas nos seus planos." Schaad(1998).*

A introdução à tese de doutorado de Schaad (1998), intitulada "Representação e execução de seqüências de ações situadas", apresenta algumas considerações importantes, as quais são abordadas a seguir:

- Tais considerações são compartilhadas por vários autores citados no levantamento bibliográfico de Schaad, bem como em outros autores citados neste trabalho, demonstrando que a questão de localização (*situadeness*) é extremamente relevante em projetos que pretendem exibir alguma forma de comportamento inteligente;
- Com algumas exceções, como em projetos de sistemas embarcados (*embedded*), onde as características de vida útil da bateria e disponibilidade de banda de transmissão são requisitos de projeto, os projetistas, tanto de sistemas operacionais como de aplicações, tendem a desconsiderar o aspecto de localização (*situadeness*) em seus projetos.

- Ao se traçar um paralelo entre as considerações de Schaad e as apresentadas em capítulos anteriores, claramente identifica-se uma discrepância entre a orientação das pesquisas na área de sistemas operacionais e nas áreas de Agentes e Robôs Autônomos.
- Como foi demonstrado, existe uma acirrada competição entre correntes da área de Inteligência Artificial e da Robótica na busca pela melhor forma de construir robôs inteligentes. Além disso, pesquisadores de outras áreas vêm aproveitando-se de técnicas dessas duas áreas em seus experimentos já há bastante tempo. Por isso tudo, chama a atenção o fato de que os pesquisadores da área de sistemas operacionais tenham ignorado e continuem a desconsiderar esse importante movimento evolutivo.

Substituindo-se deliberadamente algumas palavras na afirmação de Schaad, acima apresentada, obtém-se uma descrição de um cenário conhecido na área de Engenharia de Software. As alterações são indicadas da seguinte forma: as palavras substituídas no texto original estão escritas com tipo de letra riscado, enquanto as palavras introduzidas estão escritas com tipo de letra sublinhado, como segue:

> *"A programação (~~de agentes situados~~) é uma tarefa complexa, visto que a interação em um ambiente dinâmico, imprevisível e potencialmente grande introduz problemas significativos. A maior parte deles está relacionada ao modo como os (~~agentes usam os planos~~) programadores desenvolvem programas que determinam seu comportamento. Tradicionalmente, os (~~planos eram~~) programas são executados literalmente; o (~~agente fazia~~) programa faz exatamente o que (~~o plano estabelecia~~) foi programado. Isto (~~impunha~~) impõe uma enorme responsabilidade no indivíduo que (~~construía os planos~~) constrói os programas porque ele (~~tinha~~) tem que antecipar todos os possíveis caminhos pelos quais o processo de interação do (~~agente~~) programa com o meio poderia conduzi-lo. ~~Hoje está claro que, um agente deve ser capaz de interpretar planos de uma forma mais sensível e dependente de contexto; ele deve ser capaz de improvisar, interromper, retomar e seqüenciar atividades para ativamente alimentar-se de informações e usar a situação atual para eliminar referências ambíguas nos seus planos~~.*

A partir da afirmação acima, cabem as seguintes considerações:

- Sob a ótica de um projetista de sistema operacional, um computador é um ambiente hostil e dinâmico. Placas controladoras podem ser substituídas ou removidas. O comportamento dos programas é imprevisível. A finalidade de uso do computador é dependente de usuário. Erros em programas podem alterar a confiabilidade do sistema e propagar problemas. E estas questões podem ocorrer durante a execução do sistema ou entre execuções (ou seja, entre cada processo de reinicialização). Estas considerações caracterizam um ambiente complexo, dinâmico e potencialmente grande.
- A analogia entre planos, em robótica, e programas, em um sistema operacional, é imediata. Contudo, as pesquisas em agentes autônomos consideram os planos no **tempo passado,** enquanto as pesquisas em sistemas operacionais não fazem referências a este aspecto em particular;
- Grande parte dos problemas detectados em programas ocorre justamente quando sua seqüência lógica de operações desvia por um caminho não suficientemente planejado e que, por isso mesmo, pode apresentar situações cujas demandas não possam ser atendidas. Isto caracteriza um ambiente onde há incerteza e informações imprecisas.
- O aspecto mais intrigante nesta afirmação é que, mesmo considerando-se as demandas de computação ubíqua, não são projetadas alterações na concepção de arquitetura das soluções

ao nível do sistema operacional, mas as propostas analisadas apresentam soluções de *kernel implants* discutidas na seção 5.3(Pág.145).

O presente capítulo apresentou algumas das abordagens de tecnologia de informação que estão sendo consideradas para instrumentalizar o sistema operacional de forma a atender algumas demandas ainda não satisfeitas pelo modelo atual. A constatação imediata é que as estratégias descrevem melhoramentos específicos. Este fato contribui para a manutenção do atual modelo, uma vez que não são propostas alterações significativas. A Tabela 20 apresenta um resumo das características das estratégias apresentadas neste capítulo. Assim sendo, pode-se considerar que as propostas enquadram-se na fase 2 do processo evolutivo dos sistemas operacionais (seção 2.6Pág.85), ou seja, caracterizam-se pelo desenvolvimento de bibliotecas específicas as quais eventualmente serão incorporadas pelas gerações seguintes.

Tabela 20 - Resumo das características de uso da tecnologia de informações

Características da TI

• Enfoque de instrumentalizar o sistema operacional a partir do uso de técnicas de extração de informações apresentadas no capítulo 2;
• Abordagens localizam-se na fase 2 do ciclo evolutivo dos sistemas operacionais, ou seja, baseiam-se na construção de bibliotecas especializadas que executam sobre o sistema;
• Técnicas utilizadas:
  ○ Inteligência Artificial;
  ○ Agentes;
  ○ Análise de perfil baseada em histórico;
  ○ Visualização de informação.

Além disso, foram apresentados alguns conceitos fundamentais relativos ao conceito de mundo, o qual é muito utilizado na área de Robótica. E concluímos com uma análise que demonstra claramente a dissociação entre os objetivos dos projetos de sistemas operacionais, as demandas de computação ubíqua e a orientação dos projetos de Robótica. A partir das informações coletadas, produziu-se a Tabela 21, identificando os requisitos funcionais e não funcionais de projetos de robôs autônomos.

Tabela 21 - Requisitos funcionais e não funcionais de projetos de robôs autônomos

Requisitos Funcionais:
• Reatividade ao ambiente, • Comportamento inteligente, • Integração de múltiplos sensores, • Resolução de múltiplos objetivos, • Robustez e confiabilidade, • Adaptabilidade e raciocínio global, • Mobilidade, • Autonomia, • Interação entre Agente-Agente e Agente-Humanos.
Requisitos Não-Funcionais:
• Modularidade e flexibilidade, • Facilidade de expansão.

O próximo capítulo apresenta a caracterização do problema de pesquisa que o presente trabalho pretende atacar e produz uma declaração formal do mesmo.

5 CARACTERIZAÇÃO DO PROBLEMA

"O processo total pelo qual o homem reconhece uma necessidade sua que pode ser satisfeita pelas respostas de um computador compreende quatro etapas: (1) os dados necessários devem ser colocados no computador de modo que este possa usá-los; (2) alguém deve dizer ao computador o que fazer; (3) o computador deve ler os dados, processá-los e escrever, então, as respostas; (4) as respostas do computador devem ser colocadas de modo a poderem ser usadas pelas pessoas" Ivan E.Sutherland (1966).

5.1 Introdução

Este capítulo tem por objetivo apresentar a definição do problema, através da enunciação de seus conceitos básicos, seu domínio do problema e campo de aplicação, gerando, dessa forma, uma declaração formal sobre ele.

Esta definição inicia a partir de algumas considerações sobre o estado da arte em projetos de sistemas operacionais, com a finalidade de enfatizar que o modelo legado possui deficiências conceituais. Ou seja, caracteriza-se a dissociação que existe entre as novas demandas projetadas e a forma de concepção dos projetos baseados no atual modelo. Algumas destas deficiências e as tarefas que elas impedem de realizar, são discutidas sob os aspectos de confiabilidade, segurança e complexidade de utilização.

O capítulo encerra analisando as projeções e tendências que ressaltam a necessidade de uma nova concepção de projeto de Sistema Operacional.

5.2 Definição do Problema

"Quantas vezes você viu alguma coisa acontecer em seu PC e desejou perguntar àquela coisa estúpida o que ela estava fazendo e porque ela estava fazendo aquilo? Quantas vezes você desejou que um sistema não repetisse o mesmo erro que ele cometeu da última vez, mas, ao contrário, pudesse aprender sozinho – ou pelo menos, permitisse que você o ensinasse como evitar o problema? Apesar da nossa tentação em antropomorfizar o computador e dizer que um sistema que não está fazendo nada visível "está pensando", nós sabemos que ele não está."[38] *Brachman (2002).*

Hoje, os usuários e as pessoas que desenvolvem e programam sistemas computacionais enfrentam muitos problemas. Os usuários encontram obstáculos na configuração de hardware e software, na incompatibilidade de versões de software, nas

[38] How many times have you watched something happen on your PC and just wished you could ask the stupid thing what it was doing and why it was doing it? How often have you wished that a system wouldn't simply remake the same mistake that it made the last time, but instead would learn by itself – or at least allows you to teach it how to avoid the problem? Despite our temptation to anthropomorphize and say that a system that is just sitting there doing nothing visible is "thinking", we all know that it's not (BRACHMAN,2002).

bibliotecas e/ou *device drivers*. Além disso, estão sujeitos a flutuações de disponibilidade de recursos da máquina e da rede a que estão conectados. Há vulnerabilidade em vários aspectos, que comprometem a estabilidade, a disponibilidade e a segurança das informações armazenadas em determinado computador. As mensagens de erro, em geral, não são auto-explicativas. E é raro agregarem valor a uma possível solução. Os sistemas de ajuda são limitados e inflexíveis. A cada momento há necessidade de readaptação às novas versões de software, com conseqüente necessidade de treinamento (MATTOS e PACHECO,2001).

As pessoas que desenvolvem aplicações são colocadas diante de uma série de outros problemas. As dificuldades compreendem desde questões básicas, relativas à complexidade de aquisição e transferência de conhecimento do campo para o ambiente de projeto, até a diversidade e heterogeneidade de plataformas e versões de sistemas operacionais e ambientes de desenvolvimento disponíveis - em permanente evolução. Os problemas que os usuários detectam no fim da cadeia são mínimos, frente aos desafios cada vez maiores gerados pela necessidade de rápida adequação às flutuações tecnológicas.

A situação dos técnicos responsáveis pelo desenvolvimento de software básico e, em última instância, pelo desenvolvimento dos sistemas operacionais sobre os quais são desenvolvidas as aplicações, é igualmente complexa. As dificuldades, nesse caso, abrangem questões de qualidade no desenvolvimento do software (*bugs*), de compatibilidade com a enorme quantidade de dispositivos diferentes que podem ser conectados ao sistema e com versões anteriores dos próprios sistemas. Este emaranhado de tecnologias e requisitos diferentes causa um sensível impacto na performance geral dos sistemas.

5.3 Deficiências do estado da arte

> *"Desde o início da evolução dos computadores modernos, tornou-se claro que o computador poderia ser um soberbo arquivista que forneceria acesso instantâneo a qualquer uma de suas informações – desde que os arquivos estivessem total e adequadamente organizados e que estivessem completamente programados os tipos de respostas que fossem pedidas ao computador" Marvin L.Minsky(1966).*

O que pode ser observado ao longo da evolução dos sistemas operacionais em geral, e das abordagens adotadas na concepção de seus núcleos, é a constante busca no sentido de identificar a estrutura adequada a uma determinada realidade. Sendo uma camada de software interposta entre as aplicações e o hardware propriamente dito, sua estrutura tem um impacto fundamental na performance e abrangência das aplicações que são construídas sobre ele (MONTZ et al.,1995).

Segundo Seltzer,Small e Smith (1995), muitas das melhorias de performance citadas em recentes pesquisas em sistemas operacionais, descrevem melhoramentos específicos na funcionalidade dos sistemas, os quais produzem aquele resultado em um determinado conjunto de casos. Geralmente, mudanças dessa natureza melhoram a performance de uma determinada aplicação, mas, em contrapartida, prejudicam as de outras.

Seltzer,Small e Smith (1995) sustentam que a busca incessante por minimização do modelo de núcleo é uma indicação clara de que o modelo atual de sistemas operacionais não é apropriado. As interfaces atuais não fornecem a flexibilidade necessária para personalizar o núcleo a cada aplicação para viabilizar o espectro cada vez maior de aplicações diferentes com diferentes requisitos.

A afirmação a seguir resume apropriadamente o contexto atual:

> *"Nós falhamos no passado em sermos oniscientes[39] sobre os requisitos futuros dos sistemas operacionais. Não há razão para acreditarmos que teremos sucesso projetando uma nova interface fixa. Ao invés disso, a única solução genérica (general-purpose) é construir uma interface de sistema operacional que seja facilmente extensível." Seltzer,Small e Smith(1995).*

De uma forma geral, os sistemas operacionais convencionais apresentam uma interface fixa com um conjunto pré-definido de políticas que a implementam Essas políticas constituem o mínimo denominador comum dos requisitos necessários às aplicações (ENDO et al.,1994). Quando uma política em particular é inapropriada para uma

[39] Onisciente: que tem saber absoluto, pleno; que tem conhecimento infinito sobre todas as coisas (HOUAISS ,2002).

aplicação em particular, uma de duas alternativas deve ser selecionada: deixar a aplicação sofrer com a política inapropriada ou reimplementar o mecanismo do núcleo em modo usuário.

Sistemas de gerenciamento de banco de dados caracterizam um exemplo de aplicação que, por não dispor dos serviços adequados para gerenciamento de memória fornecidos pelos sistemas operacionais, implementam suas próprias políticas de gerenciamento da mesma a revelia do sistema (embora usando primitivas do sistema). Isto causa um impacto na performance da máquina uma vez que, a aplicação passa a competir com o sistema operacional na gerência de recursos.

Outro cenário confuso envolve os mecanismos de escalonamento dos sistemas operacionais. Projetados de forma simples, estão sendo cada vez mais submetidos a aplicações de tempo-real, *multi-threaded* e aplicações distribuídas. As soluções apresentadas pelos projetistas, como as propostas em Steere et al. (1999), Anderson et al. (1991), Alfieri (1994) e Iyer e Druschel (2001), são complexas. Note-se que elas poderiam ser simplificadas facilmente, apenas provendo primitivas que permitissem às aplicações especificar suas próprias disciplinas de escalonamento (SELTZER,SMALL e SMITH,1995).

Segundo Seltzer,Small e Smith (1995), inúmeras soluções intermediárias, que Seltzer qualifica como implantes no núcleo –*kernel implants,* têm sido apresentadas para solução desse problema. Como, por exemplo, em Ballesteros et al. (1999), Ballesteros,Hess,Kon e Campbell (1999), Hunt e Brubacher (1999), Satyanarayanan,Flinn e Walker (1999), Golm et al. (2001) e Tamches e Miller (1999). O próprio modelo para testes de robustez, apresentado anteriormente, utiliza este artifício para alterar a funcionalidade do sistema através da inserção de seqüências aleatórias de eventos.

Lowekamp et al. (1999) discutindo um aspecto específico de projeto relacionado ao acesso das aplicações a interface de rede apresenta as seguintes considerações:

> *"Uma das questões mais espinhosas relacionadas à definição de uma interface entre aplicações e a rede é decidir quais aspectos da rede devem ser expostos às aplicações. Uma solução é expor o máximo de informações possível. Contudo, exportar informação de baixo nível ou específica do sistema conflita com outras metas, particularmente aquelas relacionadas com a portabilidade da API através de redes heterogêneas. Isto também apresenta a questão sobre aspectos de facilidade de uso e facilidade de expansão (scalability). Além disso, a Internet é muito grande para expor todas as informações. Uma alternativa é disponibilizar as informações em um nível de abstração muito alto com foco nas características de performance de interesse da aplicação. Um nível de abstração muito alto pode facilitar o uso e evitar sobrecarga (overload) de informações, mas pode também resultar em imprecisões e incertezas".*

As considerações de Lowekamp indicam uma questão de compromisso entre visibilidade e segurança não discutida por Seltzer, mas que é extremamente relevante no contexto da exposição das interfaces do núcleo às aplicações.

Segundo Seltzer,Small e Smith (1995), o problema fundamental, repetidamente trabalhado, é que a interface do sistema operacional não é flexível o suficiente para permitir às aplicações a funcionalidade e/ou a flexibilidade de que elas necessitam. Em cada caso descrito na literatura, as técnicas sugeridas enfocam um conjunto específico de sintomas, mas não resolvem o problema como um todo. A seguir são alguns exemplos são apresentados caracterizando esta afirmação.

Recentes avanços na tecnologia de componentes permitem a construção de sistemas complexos através de sua composição. No entanto, ainda é difícil desenvolver sistemas eficientes, confiáveis e dinamicamente configuráveis, visto que os componentes freqüentemente são desenvolvidos por equipes diferentes e que utilizam metodologias diferentes, o que naturalmente conduz a falhas inesperadas, comprometendo a confiabilidade do produto final.

O suporte a aplicações multimídia também possui requisitos não totalmente atendidos pelos sistemas atuais. Isto porque a performance de cada processador virtual[40] (multiplexado por fatias de tempo), é influenciada pela carga dos demais processadores virtuais e os mecanismos para controlar esta interferência não são disponibilizados às aplicações. No entanto, aplicações multimídia necessitam cada vez mais de tais mecanismos. Uma forma de controlar esta interface tem sido viabilizada através do uso de vários processadores reais[41] (através de cartões periféricos: sintetizadores de som, placas aceleradoras de vídeo, etc), os quais aliviam a carga do processador central executando tarefas especializadas.

Coady, Kiczales e Feeley (2000), afirmam que os projetos de sistemas operacionais apresentam um problema sério de modularização. Segundo os autores, um estudo realizado no sistema **OS/360**, no início da década de 70, demonstrou que o número de módulos envolvidos em uma alteração "aumentou de 14,6% nos releases 2 a 6 para 31,9% nos releases 12 a 16, em função de interações não intencionais entre os componentes". Esta situação parece não ter mudado uma vez que:

[40] Ver Figura 44 na página 179 no lado direito da linha tracejada.
[41] Ver Figura 14 na página 64.

> *"Modernos sistemas operacionais comerciais como Windows NT, necessitem que os projetistas de sistemas de arquivos third party[42] estejam familiarizados com os padrões de interação que necessariamente existe entre o sistema de arquivos, o gerente de cachê e o gerente de memória virtual. Pesquisas em sistemas extensíveis também têm falhado em suportar personalização (customization) incremental pela mesma razão. Isto é, não é possível garantir que uma alteração no sistema irá requerer um esforço proporcional a quantidade de código envolvida quando a questão refere-se à falta de localidade" Coady, Kiczales e Feeley (2000).*

Conforme apresentado no capítulo 2(Pág.55), a tendência em projetos de sistemas operacionais, ao longo do tempo, tem sido no sentido de mover-se partes de código tradicionalmente encontrado no núcleo para fora dele. Ao mesmo tempo, os projetos estão exportando cada vez mais interfaces de baixo nível, de forma a dotar os programadores de recursos para construir aplicações que possam implementar políticas específicas utilizando *library operating systems* implementados ao nível de usuário (HULSE e DEARLE,1998).

Há, portanto, diferentes abordagens para a construção de núcleos de sistemas operacionais. Todas elas possuem pontos fortes e pontos fracos, tornando-as mais adequadas para propósitos e equipamentos específicos. Estas questões estão presentes tanto em sistemas comerciais tradicionais de propósitos gerais, como em comerciais dedicados (ou embarcados – *embedded*) e até mesmo em sistemas de pesquisa.

No entanto, apesar de finalidades aparentemente diferentes e em algumas situações com requisitos conflitantes, invariavelmente as propostas recaem sobre a definição apresentada inicialmente, ou seja: um componente de software que se pretende que venha a controlar o hardware e o torne disponível ao usuário. A Tabela 18(Pág.91) apresenta as principais características destes projetos.

Segundo Mosberger (1997),

> *"Apesar dos avanços tecnológicos experimentados nos últimos tempos, particularmente na última década, a estrutura fundamental dos sistemas operacionais tem permanecido invariavelmente a mesma". E ele continua: "A luz de eventos recentes como, por exemplo, a popularização da internet, é razoável questionar se o enfoque em computação (computing) é aceitável. A recente ênfase em interligação de equipamentos (networking) pode indicar que o enfoque em comunicação (communicating) pode logo vir a se tornar à razão de ser dos computadores".*

Acrescente-se a esta afirmação toda a discussão sobre computação ubíqua apresentada no capítulo 3(Pág.94).

[42] Desenvolvido por equipes outras que aquelas que projetaram e desenvolveram o sistema operacional.

À medida que a evolução tecnológica produz hardware mais rápido e com mais recursos, surgem novas necessidades em termos de software e, à medida que estas facilidades são incorporadas "ao sistema", os usuários tornam-se cada vez mais exigentes, o que conduz a um círculo vicioso. Naturalmente, à medida que novas facilidades são disponibilizadas ao usuário, novos recursos são agregados ao núcleo do sistema operacional, ou novas aplicações de suporte fazem-se necessárias para que elas a funcionem adequadamente.

Kon (2000) afirma que,

> *"Os sistemas atuais não são projetados para adaptarem-se rapidamente a mudanças ambientais tais como: atualizações de software e hardware e, flutuações na carga de processador e banda de rede disponível. Não há um mecanismo que permita a inserção de componentes auto-adaptáveis que possam otimizar a performance do sistema de acordo com as diversidades acima comentadas".*

Para atender a esta demanda cada vez maior por suporte a novos recursos, a tendência dos projetistas de sistemas operacionais tem sido a de desenvolver sistemas cada vez mais genéricos. Nicol et al.(1989) afirmam que este enfoque não é satisfatório por 2 razões principais:

- A funcionalidade é limitada por restrições ou generalizações inapropriadas, tendo em vista a abrangência necessária; e
- Ocorre redução na performance do sistema, uma vez que os recursos são ajustados para a maior possibilidade de uso e não para a melhor.

Em Beos (1998) são relacionadas algumas das suposições para que os sistemas operacionais atuais apresentem tais deficiências. São elas:

- **Modelo monoprocessador**: a arquitetura dos PCs atuais foi projetada há cerca de duas décadas atrás, época em que o custo dos microprocessadores era alto; hoje eles são commodities. No entanto, ainda são usados sistemas operacionais que podem ser executados em somente um processador.
- **Tratamento de mídia digital**: muitas das necessidades requeridas hoje em termos de mídia digital (imagens, sons, internet e outros), eram talvez consideradas em teoria quando os atuais sistemas foram projetados. Em conseqüência, sua arquitetura não possui a funcionalidade necessária para tratá-las adequadamente. Ou seja, a indústria está tentando liberar soluções para a próxima década usando arquiteturas projetadas para resolver problemas da década passada.
- **Compatibilidade**: a cada ano, os avanços de hardware melhoram significativamente a velocidade e reduzem custo dos processadores e periféricos. No entanto, o usuário final somente percebe uma fração dos recursos liberados a cada nova versão de hardware, em função da questão de compatibilidade binária com versões anteriores.

Neste sentido, Hydari (1999) afirma que o crescimento da Internet tem permitido um nível de conectividade entre computadores que atinge um nível de onipresença. Mas este progresso tem gerado um determinado número de problemas que não são adequadamente tratados pelos atuais sistemas operacionais disponíveis. São eles:

- Utilização de um grupo de máquinas para execução de computação distribuída;
- Migração de dados, computação e usuários;
- Manutenção e atualização de software, dependências entre componentes e portabilidade;
- Flexibilidade de Crescimento (*Scalability*);
- Confiabilidade de sistemas de software;
- Administração distribuída do sistema;
- Segurança.

Assim sendo, o que se observa é que o sistema operacional, apesar de sua importância fundamental, não consegue acompanhar a evolução tecnológica, pelo menos na velocidade em que ela ocorre. Em função disto, à medida que surgem soluções temporárias, novas fontes de instabilidade são agregadas ao conjunto de problemas potenciais a que o usuário está sujeito.

A seguir são discutidos aspectos específicos relativos à confiabilidade e segurança dos dados armazenados em um computador, considerando-se as demandas identificadas no capítulo 3(Pág.94).

5.4 Confiabilidade e Segurança

> *"É importante observar que provavelmente nenhum grande sistema operacional, que utiliza a atual tecnologia de projeto, pode fazer frente a um ataque determinado e bem coordenado. E que a maioria das invasões documentadas têm sido notadamente fáceis [43]" Hebbard et al.(1980).*

Segundo Deitel (1990), o nível de segurança que deve ser aplicado a um sistema é dependente do valor que os recursos protegidos possuem. À medida que a computação evolui, ela se torna também mais acessível. E, à medida que os computadores passam a enviar/receber informações pessoais, a questão da segurança cresce cada vez mais em importância. No passado, textos sobre segurança consideravam aspectos relativos ao controle de acesso físico às instalações. Outra abordagem referia-se a políticas de acesso

[43] Its important to note that probably no large operating system using current design technology can withstand a determined and well-coordinated attack, and that most such documented penetrations have been remarkably easy (HEBBARD et al.,1980).

que eram aplicadas a usuários, determinando que tipo de acesso era permitido e a que entidades do sistema. Hoje, este problema encontra-se potencializado com acessos "invisíveis" via rede.

É cada vez mais improvável encontrar-se pessoas que ainda não tenham tido problemas com os famosos vírus em computadores. O grau de gravidade de uma infecção varia desde a perda de informações, até a divulgação de informações reservadas. Elas podem fazer com que o sistema passe a funcionar cada vez mais lentamente ou destruí-lo completamente, causando a necessidade de uma completa reinstalação.

As pesquisas na área de segurança vêm evoluindo na mesma medida em que evoluem os sistemas. Como decorrência, surgiu em 1966 a proposta de sistemas baseados em *capabilities*[44] (DENIS e VAN HORN, 1966). Um conjunto de direitos de acesso que determinado objeto possui é denominado de domínio de proteção. Há também questões relativas à criação de novas *capabilities*, sua movimentação, armazenamento e revogação. Pode-se citar alguns projetos da década de 70 (POPEK e KLINE, 1974; MCCAULEY, 1979; WALKER, KEMMERER e POPEK, 1979) que já tinham por objetivo o desenvolvimento de propostas orientadas ao desenvolvimento de núcleos mais seguros.

No entanto, o aspecto de vulnerabilidade continua cada vez mais presente nos dias de hoje. Segundo Deitel (1990), como a tendência atual é a de desenvolverem-se núcleos pequenos, torna-se razoável examinar mais cuidadosamente o próprio núcleo e demonstrar formalmente sua correção. Um dos aspectos mais críticos de um sistema é o mecanismo que garante proteção às funções de controle de acesso, registro (*logging)* e monitoramento, e às que gerenciam a memória real, a virtual e o sistema de arquivos. Estas funções geralmente ocupam uma grande porção do código do sistema, tornando praticamente impossível a sua implementação dentro do núcleo. E continua Deitel afirmando:

> *"Parece que novas abordagens de projeto de sistemas operacionais serão necessárias para reduzir o tamanho dos núcleos seguros ao ponto em que realmente possam ser construídos sistemas seguros".*

[44] Uma *capability* é um *token* que confere ao possuidor os direitos de acesso a determinados recursos.

5.4.1 Aspectos de confiabilidade de sistemas comerciais

Dentro deste contexto cabe destacar o trabalho de Miller,Fredriksen e So (1990). Desenvolvido em 1990, ele descreve um estudo sobre a confiabilidade de utilitários de linha de comando do sistema operacional **Unix**. Esse estudo mostrou que, entre 25% e 33% das aplicações testadas, ou abortaram ou paralisaram quando expostas a testes de entradas aleatórias (Figura 31).

Uma continuação desse estudo, apresentada em 1995 (MILLER et al.,1995), avaliou uma coleção maior de aplicações, incluindo algumas aplicações gráficas baseadas em **X-Windows**. Os resultados apresentados foram semelhantes ao do estudo original. Especificamente, até 40% dos utilitários de linha de comando do **Unix** abortaram ou paralisaram e 25% das aplicações baseadas em **X-Windows** também apresentaram este tipo de falha.

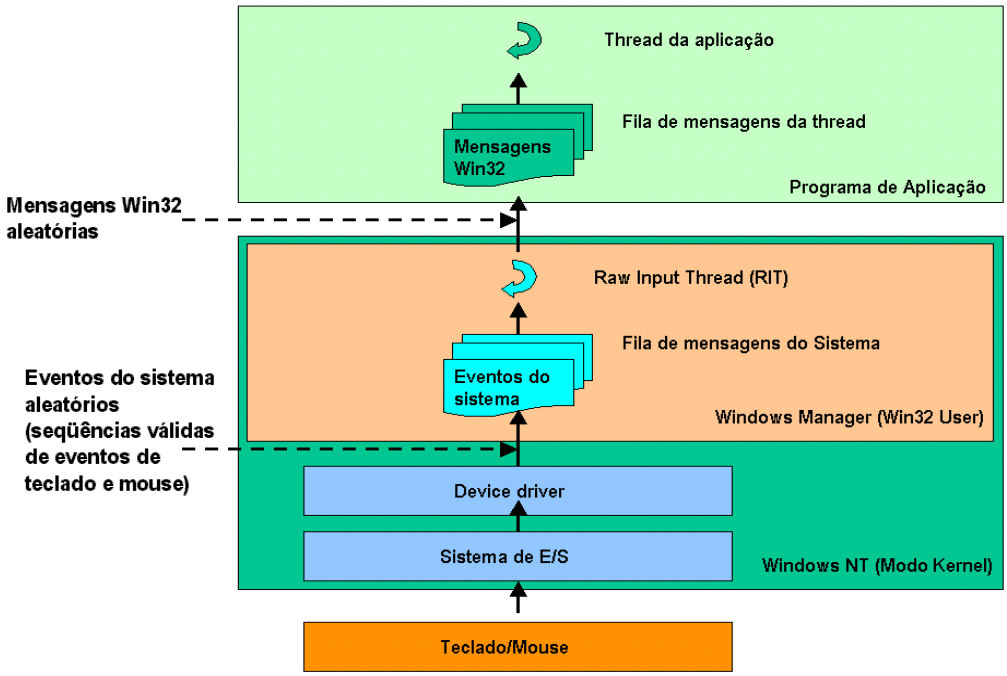


Figura 31 - Mecanismo de inserção de mensagens aleatórias no Windows NT5.0.
FONTE: Adaptado de Forrester (2000).

Na linha dos sistemas baseados em **Unix**, o trabalho de Chou et al. (2001) apresenta um estudo sobre os erros encontrados em núcleos **Linux** e **OpenBSD** através do emprego de ferramentas automáticas de análise do código fonte. Dessa forma foram analisadas várias versões de código fonte daqueles sistemas operacionais, possibilitando estimar,

por exemplo, que o tempo médio de permanência ativa dos erros no sistema, antes de serem corrigidos, é de aproximadamente 1,8 anos. O trabalho relata, também, que os *device drivers* apresentam taxas de erros até sete vezes maiores do que as encontradas em outros módulos, conforme apresentado na Figura 32.

Estes resultados são significativos na medida em que robustez é um dos pré-requisitos para se obter sistemas seguros. E tornam-se ainda mais significativos quando se constata que continuam sendo realizadas tentativas de adaptar os sistemas baseados em **Unix** aos requisitos de segurança, embora seja de conhecimento publico que eles, de uma forma geral, não são seguros.

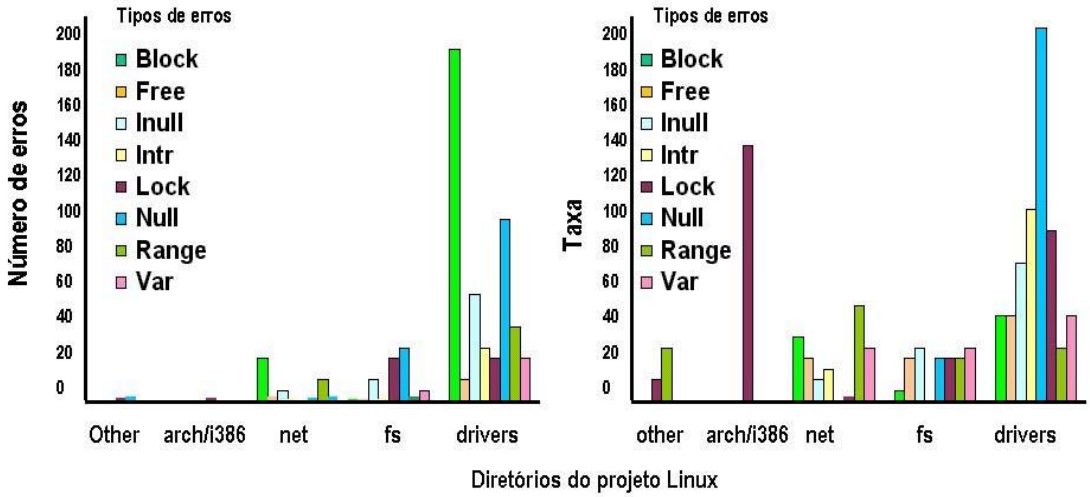


Figura 32 - (a) numero de erros por diretório no linux; (b) taxa de erros comparada com outros diretórios.
FONTE: Chou et al. (2001).

Forrester (2000) descreve os resultados dos testes de robustez em aplicações baseadas em **Windows NT**. A Figura 31 apresenta graficamente a estratégia usada para inserir eventos aleatórios nas filas de mensagens do windows, de forma a excitar as aplicações. As estatísticas atualizadas são as seguintes:

- 21% das aplicações testadas na versão **NT 4.0** abortaram quando submetidas a testes de entradas aleatórias (mensagens válidas eventos de mouse e teclado). Os mesmos testes na versão **NT 5.0** (Windows 2000) apresentaram resultados similares;
- Até 100% das aplicações testadas falharam quando submetidas a fluxos aleatórios de mensagens da API Win32.

Uma conclusão a que Forrester (2000) chegou a partir dos testes realizados é que, "qualquer aplicação executando em plataformas **Windows NT** está vulnerável a

seqüências aleatórias de mensagens geradas por qualquer outra aplicação executando no mesmo sistema. Isto denota um aspecto critico na arquitetura de mensagens do sistema".

Uma outra observação de Forrester (2000) é que os programadores também contribuem para que esta taxa de erros seja tão elevada. Segundo o autor, "parece haver um acordo entre os programadores para utilizar ponteiros diretamente a partir do conteúdo das mensagens sem antes realizar verificações de segurança". Esta situação conduz a uma infinidade de sub-versões e *patches* corretivos que já fazem parte do dia-a-dia da sociedade informatizada (Figura 33).

Figura 33 – Versões de correção de erros

Estas conclusões, se analisadas considerando-se as questões apresentadas por Linde (1975), demonstram que, apesar do grande esforço e desenvolvimento realizado pelos sistemas operacionais, particularmente nos últimos anos, os aspectos de segurança continuam a ser o seu ponto fraco do sistema e, por conseguinte, a principal origem de muitos dos problemas relacionados à sua instabilidade. O próprio mecanismo de geração de mensagens aleatórias, por si só, já caracteriza uma falha de segurança, pois permite que todas as funções do sistema sejam acessadas sem controle.

A questão é tão importante que foi objeto de um estudo financiado pela NSA (DTOS,2001), denominado projeto **Synergy**. No escopo deste projeto maior, desenvolveu-

se o projeto **DTOS,** encerrado em 1997, cujo objetivo principal foi o de desenvolver uma arquitetura baseada em micronúcleo para sistemas distribuídos seguros. Os esforços do projeto *Synergy* fazem parte de uma estratégia de longo prazo do governo americano, que tem por objetivo encorajar os vendedores a incluir mecanismos de segurança consistentes (*strong*) nas próximas gerações de sistemas operacionais comercialmente disponíveis. Para tanto foram realizados estudos envolvendo diversos sistemas operacionais que possuíam mecanismos de segurança flexíveis, com o objetivo de identificar os mecanismos necessários para suportar um grande espectro de políticas de segurança. Como resultado deste esforço de pesquisa, várias indicações foram produzidas, bem como um protótipo sobre o micronúcleo Mach 3.0 (DTOS,2001).

Uma proposta mais recente, também desenvolvida pela NSA, foi apresentada à comunidade em janeiro de 2001. Segundo a *press release* (NSA,2001):

> *"Reconhecendo-se o papel critico que os mecanismos de segurança dos sistemas operacionais em suportar aplicações sensíveis e críticas, pesquisadores da NSA têm investigado uma arquitetura de sistema operacional que possa prover a funcionalidade de segurança necessária de uma forma que possa atender as necessidades de segurança de um amplo espectro de ambientes computacionais. A NSA anuncia que desenvolveu e está tornando publico uma versão protótipo de um sistema linux security-enhanced".*

Os mecanismos de segurança implementados possuem suporte flexível para uma grande variedade de políticas de segurança. O sistema está sendo disponibilizado sob os termos da GNU *(General Public License*) com a intenção de permitir que a comunidade refine os melhoramentos de forma a incluí-los no **Linux**.

Com base nas informações apresentadas anteriormente, percebe-se que, apesar do desenvolvimento das várias abordagens descritas na seção 2.4[45], os problemas de segurança continuam presentes e não se vislumbra uma solução que seja ao mesmo tempo viável e robusta[46]. A seção a seguir apresenta alguns aspectos relativos a estudos sobre invasões de sistemas computacionais, no sentido de corroborar a afirmação anterior.

[45] Estruturas organizacionais clássicas.
[46] Viável no sentido de não comprometer a performance do sistema em termos de recursos computacionais e robusta no sentido de garantir confiabilidade das informações preservando a sua integridade.

5.4.2 Estudos sobre Invasões

Estudos sobre invasões de computadores também não são novos. Linde (1975) em 1975 descreve os resultados de um determinado número de estudos sobre invasão de computadores. A seguir são relacionados os pontos destacados pelo autor como problemas funcionais:

- Autenticação: em muitos sistemas os usuários não podem determinar se o hardware e/ou software que está sendo executado é aquele que supostamente deveria estar sendo executado. Este fato facilita a tarefa do invasor em substituir programas de autenticação;
- Criptografia: a lista mestra de senhas deveria ser armazenada de forma criptografada. Nem sempre o é;
- Implementação: Um projeto de um bom mecanismo de segurança pode ser impropriamente implementado;
- Confiabilidade implícita: é um problema comum. Uma rotina assume que outra está funcionando corretamente quando, de fato, deveria cuidadosamente examinar os parâmetros recebidos;
- Compartilhamento implícito: o sistema pode, inadvertidamente, armazenar informação critica em espaço de endereçamento não protegido;
- Comunicação entre processos: um invasor pode utilizar mecanismos de send/receive para testar várias possibilidades;
- Verificação: o sistema pode não estar realizando suficiente validação de parâmetros;
- Queda de linha: em sistemas *dial-up*, quando a linha é perdida por alguma razão, o sistema operacional deveria detectar e imediatamente colocar o usuário em um estado que requer uma nova autorização para obtenção de acesso. Em alguns sistemas a queda da linha deixa uma possibilidade aberta para um intruso obter a identidade do usuário que estava autorizado e, portanto, seus direitos de acesso;
- Passagem de parâmetros por referencia versus por valor: é mais seguro passar parâmetros por registradores do que fazê-los apontar para posições de memória que contem os parâmetros. Passagem por referência pode levar a uma situação em que os parâmetros ainda estejam no espaço de endereçamento do usuário após a verificação ter acontecido. Neste caso, um intruso poderia substituir os parâmetros conforme seu interesse;
- Senhas: são freqüentemente fáceis de identificar ou obter por tentativas sucessivas;
- Confinamento: os sistemas deveriam suportar um esquema de confinamento como uma primeira linha de defesa contra invasões;
- Privilégio: em muitos sistemas, muitas pessoas (programadores, por exemplo) possuem muitos privilégios. Isto pode causar problemas;
- Proibições: freqüentemente os usuários são avisados para não utilizar determinadas facilidades porque elas podem causar resultados imprevisíveis. No entanto, tais facilidades permanecem habilitadas para os usuários;
- Resíduos: freqüentemente um invasor pode encontrar uma lista de senhas, simplesmente vasculhando uma lixeira. Resíduos algumas vezes permanecem em memória depois que uma determinada rotina executou. Assim, um programa específico poderia vasculhar a memória

procurando por informações úteis para o invasor. Portanto, informações sensíveis deveriam ser sobrescritas em memória para garantir segurança;

As próximas seções analisam alguns aspectos de segurança relativos às demandas identificadas no CAPÍTULO 3 - NOVAS DEMANDAS: COMPUTAÇÃO UBÍQUA, TELETRABALHO e COMÉRCIO ELETRÔNICO.

5.4.3 Aspectos de segurança relacionados à computação ubíqua

Aspectos de segurança relativos à área de computação ubíqua ainda não estão muito bem definidos, uma vez que a área, em si, encontra-se em fase inicial de prototipação e avaliação em ambientes restritos e controlados.

Segundo Seigneur, Farrell e Jensen (2002), "a computação ubíqua nem sempre pode fazer uso dos esquemas tradicionais de envolvimento (*enrollment*), especialmente em redes móveis e infra-estrutura global de computação". Estes autores indicam que se faz necessário um novo esquema de reconhecimento de entidades, o qual é mais geral do que autenticação. Neste cenário, nem sempre é necessária uma fase explícita de envolvimento, mas um processo onde mais ou menos atenção é dedicada a entidades presentes nas redondezas (*surrounding entities*), dependendo da sua importância estimada. Assim sendo, diferentes aplicações podem demandar diferentes modos de reconhecimento de colaboradores (SEIGNEUR, FARRELL e JENSEN, 2002).

Um recente relatório (EURESCOM,2001), produzido pelos participantes do projeto P1005 da EURESCOM[47] em junho de 2001, apresenta e discute os requisitos básicos de segurança relativos à computação ubíqua tanto do ponto de vista dos provedores como dos usuários de serviços de telecomunicações. O mesmo documento analisa, também, como os requisitos identificados podem ser atendidos pelas tecnologias atuais que suportam a computação ubíqua. Neste sentido, as considerações a seguir constituem uma compilação do referido relatório.

Tendo em vista a dimensão de uma rede de telecomunicações, é de vital importância para as companhias provedoras deste tipo de serviço identificar os aspectos perigosos (*threats*) que os sistemas de computação ubíqua e código móvel podem oferecer, a fim de se protegerem em relação a eles. Neste sentido, a contribuição do

[47] Produzido por colaboradores das empresas de telecomunicações: Portugal Telecom S/A, Deutsche Telekom AG e Broadcom/Eircom Plc.

relatório P1005 é identificação destes pontos fracos e dos requisitos de segurança necessários para superá-los. Dependendo do tipo de serviço oferecido por uma companhia de telecomunicações, pode haver diferentes requisitos de segurança a serem atendidos. O relatório identificou os requisitos das quatro principais áreas envolvidas, os quais são apresentados na Tabela 22.

Tabela 22 - Requisitos de segurança em computação ubíqua na visão de companhias de telecomunicações

Serviços de informações	• Envolvem serviços relacionados ao fornecimento de informações baseadas em localização (mapas de hotéis próximos, restaurantes) para usuários em movimento em áreas desconhecidas. Falhas neste contexto podem possibilitar a um atacante gerar/acessar perfis de movimento de um cliente, o que é constitui uma brecha na privacidade do usuário.
	Requisitos de segurança
	• Confidencialidade de informação de localização do cliente, para evitar a geração de perfil de movimentação do usuário.
	• Acesso configurável às informações sobre a localização do cliente, pelas companhias de telecomunicações, para oferecer serviços ao mesmo dependendo de sua própria configuração.
	• Identificação segura e verificação do nível de confiança de provedores de serviços baseados em localização.
Serviços de comercio eletrônico, banco eletrônico e serviços de pagamento.	Exemplos deste tipo de serviço são: acesso a serviços bancários, compras eletrônicas, aquisição remota de entradas de cinema, shows, etc.
	Requisitos de Segurança
	• Confidencialidade.
	• Autenticação.
	• Proteção de integridade.
	• Não repúdio (*non-repudiation*)[48].
Compartilhamento de dispositivos	• Envolve o compartilhamento de recursos, como equipamentos em salas de conferências ou escritórios, os quais podem ser utilizados por todos os participantes em um encontro ou conferência. Além disso, pode envolver também quiosques públicos, onde o usuário utiliza dispositivos móveis (*notebooks, palmtops*) para ter acesso a recursos alugados, como impressão.
	Requisitos de segurança
	• Acesso restrito aos recursos.
	• Confidencialidade dos dados.
Provisão de *links* de comunicação	• Tarifação para uso de recursos através das operadoras de telecomunicações em alguma das categorias.
	Níveis de segurança
	• Comunicação desprotegida.
	• Comunicação autenticada.
	• Comunicação criptografada.
	• Comunicação autenticada e criptografada.

[48] Segundo Ford,W (1994) *non-repudiation* é fundamentalmente diferente de outros serviços de segurança. Sua principal função é proteger usuários de comunicação contra *threats* de outros usuários legitimados, ao invés de atacantes desconhecidos. O autor concorda que a palavra *non-repudiatio*n em si não é muito explicativa. De fato, ela não evita que qualquer das partes envolvidas repudie reclamações de outra parte, relativas à ocorrência de algum evento. O que ela faz é garantir a disponibilidade de evidências irrefutáveis para suportar a solução rápida de quaisquer desacordos (*disagreement*). O motivo para serviços de *non-repudiation* é o fato de que, na realidade, não há sistemas perfeitos e que circunstancias podem ocorrer onde duas partes tenham visões diferentes sobre fatos ocorridos.

Outros aspectos também são considerados neste relatório como requisitos de segurança para provedores e usuários de serviços de código móvel e identificação de pontos falhos de segurança na infra-estrutura atual. Contudo, o aspecto mais importante, do ponto de vista do presente, está relacionado à conclusão do referido relatório, o qual afirma:

> *"comparando-se os requisitos de segurança identificados com as facilidades de segurança que os atuais sistemas possuem, somos forçados a concluir que ainda há muito o que fazer no setor de segurança. A nossa analise mostrou que os atuais sistemas apresentam falhas básicas de segurança. Assim sendo, as companhias de telecomunicações devem analisar cuidadosamente quando oferecer serviços de computação ubíqua e, se possível, aplicar mecanismos de segurança adicionais ao nível de aplicação. Contudo, mesmo este enfoque não resolverá todos os problemas de segurança associados à tecnologia Jini & Friends. Neste sentido, sob o ponto de vista dos autores, há uma necessidade urgente de estender a infra-estrutura Jini & Friends com relação a aspectos de segurança" EURESCOM (2001).*

5.4.4 Aspectos de segurança relacionados a teletrabalho

Um dos maiores obstáculos à implementação do conceito de tele-trabalho refere-se à questão de segurança das informações. Logicamente, o acesso externo às instalações físicas da empresa, aí incluído seu sistema de informações, sofre restrições severas. O teletrabalhador, porém, mesmo encontrando-se fora das referidas instalações, precisa ter acesso amplo às informações para poder executar seu trabalho.

A segurança da informação é caracterizada pela preservação dos seguintes atributos básicos (ISSO/IEC 17799:2000 apud MACHADO,2002):

- Confidencialidade: segurança de que a informação pode ser acessada apenas por quem tem autorização;
- Integridade: certeza da precisão e da totalidade da informação: e
- Disponibilidade: garantia de que os usuários autorizados tenham acesso às informações e aos recursos associados quando necessário.

A preservação destes atributos constitui o modelo básico da Norma Internacional para Gerenciamento da Segurança da Informação, a ISSO 17799, e de toda a ciência da Segurança da Informação (MACHADO,2002).

Machado (2002), citando o relatório 2001 *CSI/FBI Computer Crime and Security Survey*, relaciona as principais ameaças à segurança da informação no ano de 2001, que eram:

- Vírus de computador;

- Uso interno indevido do acesso à rede;
- Roubo de *notebooks*;
- Acesso interno não autorizado; e
- Penetração externa ao sistema.

O relatório observa, ainda, que é crescente o número de ameaças oriundas da Internet e que a principal causa de perdas financeiras são os vírus de computador (MACHADO, 2002).

Como o conjunto de regras para implementação de políticas de segurança é muito complexo, o que ocorre na realidade é que, após algum tempo, os usuários e administradores tendem a utilizar a configuração mais simples. Com isso, deixam de tomar todas as medidas necessárias, o que torna o sistema vulnerável (DALAL,1999 apud MACHADO,2002). Portando, esta é uma questão que ainda não possui uma solução adequada no modelo atual.

A partir da compilação do trabalho de Machado (2002)[49] foi construída a Tabela 23 (próxima página), que relaciona os principais problemas e as principais ferramentas e medidas para garantir a segurança da informação.

[49] A partir de referências aos trabalhos de Leonhard (1996), Fraser (1997), Niles (1997), Dalal (1999), Junior (1999), Girard(1999), Openheimer (1999), Gonçalves (2000), Hirsch(2000), Percel(2000), Reid(2000), Weisserflus (2000), Cartwright (2001) e Schneier (2001) citadas em Machado(2002).

Tabela 23 - Problemas e medidas de segurança em tele-trabalho.

Principais Problemas	Ferramentas e medidas de segurança	Ferramentas de Mercado
• Conexões podem ser grampeadas;	• Uso de programas antivírus nos micros dos usuários;	• Antivírus: software capaz de detectar e eliminar vírus de computador, assegurando a integridade e disponibilidade das informações;
• *Hackers* e empregados desonestos e criminosos *high-tech* podem sutilmente manipular ou destruir dados sensíveis;	• Criptografia baseada no aplicativo PGP (*Pretty Good Privacy*) para arquivos do micro e correio eletrônico;	• Backup: sistema que possibilita a reprodução e a posterior restauração de informações a partir de meios magnéticos, óticos ou outros;
• Transferência de arquivos via Internet podem ser monitoradas;	• *Firewalls* pessoais capazes de deter programas intrusos (*trojans*);	• Biometria: sistema que emprega características biométricas, tais como impressão digital e mapeamento da íris, para identificar o usuário;
• Vírus e código malicioso podem contaminar o sistema;	• *Pass phrase* (senha a partir de uma frase longa) para acesso ao sistema e aplicações;	• *Call-Back*: sistema onde o usuário remoto somente pode acessar a rede da empresa após o retorno da sua ligação telefônica efetuada pelo servidor de acesso remoto da empresa;
• Informações podem ser roubadas quando o computador do teletrabalhador sofrer suporte ou manutenção;	• Especial cuidado com mídia removível (Zip *drive*, HD, CD-ROM, DVD) que devem conter dados criptografados;	• Criptografia: processo de codificação e decodificação de dados empregando algoritmos criptográficos matemáticos complexos, protegendo a confidencialidade da informação;
• Pessoas podem ver o teletrabalhador digitando a senha em locais públicos;	• Emprego de VPN (*Virtual Private Network*);	• *Firewall*: sistema baseado em software ou hardware capaz de controlar o acesso entre duas redes ou sistemas, impedindo acessos indevidos e ataques;
• O Sistema Operacional normalmente empregado é o Windows 9x, o qual não oferece segurança;	• Emprego de servidor de comunicação com autenticação RADIUS ou TACACS;	• IDS: *Intrusion Detection System*: sistema capaz de identificar a atividade de um invasor na rede e iniciar procedimentos de alerta e contra-ataque;
• No lar do teletrabalhador o equipamento está sujeito a riscos e avarias que não existem na empresa, como crianças ou espiões;	• Emprego de *notebooks* cedidos pela empresa e previamente configurados com mecanismos de controle de segurança da empresa;	• PKI- *Public Key Infrastructure*: processo de certificação digital que possibilita a identificação inequívoca do usuário, procedência e conteúdo das informações, baseado na troca de chaves criptográficas;
• A utilização do equipamento do teletrabalhador pela família pode causar avarias nos dados na medida em que são instalados certos aplicativos;	• Emprego de *token cards* com senha para identificação dos usuários;	• Servidor de Comunicação RADIUS ou TACACS: sistema de comunicação capaz de concentrar e autenticar, de forma segura, conexões remotas, geralmente por via telefônica, dos usuários do sistema;
• Amigos podem ter acesso ao computador e avariar os dados;	• Controle de acesso com uso de permissões limitadas para acesso remoto;	• *Token Card*: sistema de identificação baseado em um cartão de identificação que possibilita a autenticação da identidade do usuário;
• Empregados desonestos podem ceder suas senhas para	• Vários esquemas de proteção de HD com senhas;	• VPN: *Virtual Private Network*: sistema implementado por

ataques; • Hotéis que fornecem serviços de acesso à rede podem ter sniffers (programas para capturar senhas) instalados.	• Biometria; • Backup do disco rígido • Emprego de NC (*network computers*) • Uso de senhas no BIOS; • Emprego de *call-back*; • Auditoria.	software ou hardware capaz de assegurar uma conexão de dados segura em meios públicos (como a Internet) através de mecanismos de tunelamento, autenticação e criptografia.

Fonte: Adaptado de Machado (2002).

5.4.5 Aspectos de segurança relacionados a comércio eletrônico

Tanto o comércio eletrônico como o teletrabalho introduzem novos requisitos às características básicas de uma infra-estrutura capaz de permitir a troca de informação e colaboração entre usuários, o que conduz a uma mudança fundamental no significado do termo *computing*. Isto porque, segundo Kalakota e Whinston (1996), o PC, antes visto somente como uma plataforma para criação, gerenciamento e análise de informações isoladas, agora deve suportar a troca de informações e colaboração. A solução para esta questão envolve a integração entre aplicações no *desktop* com aplicações em servidores remotos. Esta demanda de integração conduz ao conceito de aplicações *boundary less* o que implica na distribuição da funcionalidade dos serviços do sistema operacional através da funcionalidade da aplicação. Uma forma de se obter esta distribuição pode ser através do uso de componentes ou objetos que possam ser compartilhados dentro do contexto de uma tarefa do usuário (CIR 1994[50] apud KALAKOTA e WHINSTON,1996).

Um dos problemas que emergem desta integração, além daqueles relacionados na seção anterior, é a questão do formato dos arquivos produzidos pelas diversas ferramentas. A solução encontrada para este problema denomina-se *Active Document Architectures* – ADA. Este enfoque possibilita o acesso distribuído e a troca dinâmica de formato de arquivos de maneira transparente, através da alteração do significado do termo documento. Nos ambientes tradicionais, o usuário inicia uma aplicação para então abrir um documento. Num ambiente ADA, o usuário ativa um documento e a aplicação a ele associada é carregada automaticamente. Dessa forma, o sistema operacional, o software de visualização (edição) e a rede, conspiram de forma a prover o conhecimento necessário para visualização e ou modificação do conteúdo embutido (texto, gráficos, sons ou vídeo).

Várias soluções comerciais têm sido apresentadas neste sentido, como OLE (*Object Linking and Embedding*) da Microsoft, OpenDoc da Apple, IBM e WordPerfect. Estas soluções executam sobre plataformas mais gerais. Por exemplo, a solução OLE executa sobre a plataforma COM (*Component Object Model*) da Microsoft, enquanto a solução OpenDoc executa sobre a plataforma SOM (*System Object Model*) da IBM. A

[50] *The Emerging Component-Based Desktop: The Object is Integration. Computer Industry Report, December 9,1994,pág.1.*

integração multi-plataforma utiliza a arquitetura CORBA (*Common Object Request Broker Architecture*), a qual é baseada no conceito de objetos distribuídos para encapsular dados e instruções em objetos. Estes objetos podem ser transportados pelas redes (protocolos) mais comumente conhecidas, executar em diferentes plataformas de hardware, acessar aplicações legadas (através de *wrappers*) e gerenciar os recursos que elas controlam (KALAKOTA e WHINSTON, 1996).

Cabe observar que estas categorias não possuem uma divisão clara. À medida que os sistemas operacionais tornam-se mais orientados a objetos, a linha de separação entre estes níveis e os serviços do sistema operacional vai sendo unificada (KALAKOTA e WHINSTON, 1996).

A próxima seção apresenta uma análise das deficiências do modelo atual de sistemas operacionais especificamente em relação à sua complexidade.

5.5 Complexidade de Utilização

Um outro aspecto importante a ser considerado refere-se à questão de facilidade de uso e administração dos sistemas em geral. Observe-se que não se pretende realizar uma análise sobre critérios de usabilidade, mas tão somente caracterizar a problemática envolvida na operação dos sistemas.

Conforme citado anteriormente, a realidade tem demonstrado que a atividade de administração de um sistema operacional é uma tarefa árdua, complexa e, muitas vezes, uma questão "vaga", que depende da experiência de alguns chamados "gurus". Esta afirmação pode ser considerada como uma verdade irrefutável para praticamente todos os sistemas operacionais já desenvolvidos. Vale também para aqueles mais "modernos" que possuem uma interface mais amigável com o usuário. Mais cedo ou mais tarde, seus segredos necessitarão ser devidamente ajustados por um profissional mais experiente para garantir um perfeito funcionamento (ou pelo menos próximo disto) do sistema alvo (MATTOS, MARTINS e PACHECO,2000).

Um exemplo é o sistema operacional **Unix**. Desde o início da década de 80, inúmeros livros já foram publicados abordando vários de seus aspectos. Estes livros podem ser considerados de fundamental importância para usuários que estejam começando a utilizar aquele sistema operacional e até mesmo para os mais experientes. A tarefa de administrar um sistema operacional **Unix** envolve uma gama muito grande de

conhecimentos que abrange desde sua configuração, passa por ajustes de performance e chega até às mais complicadas soluções de problemas.

Esta dificuldade não é exclusiva dos sistemas **Unix**. Basta observar-se a vasta gama de cursos de certificação nas várias plataformas disponíveis e a solicitação cada vez maior por profissionais bem treinados para operar os diversos sistemas. Além disso, este problema é potencializado na medida em que as organizações passam a utilizar várias plataformas diferentes, em vários setores, com diversos níveis de exigência e conhecimento dos usuários finais (MATTOS, MARTINS e PACHECO,2000).

De acordo com Hugh (1997),

> *"Quando as interfaces com o usuário foram originalmente projetadas, não havia muita preocupação com a qualidade do que era apresentado, muito em função dos escassos recursos disponibilizados pelas mesmas (monitores monocromáticos de baixa resolução). Nesta época, as interfaces eram baseadas em caractere, em terminais alfanuméricos com teclas de função sendo utilizadas para complementar e/ou ativar funções especiais de uma aplicação. Contudo, a medida em que novos recursos foram sendo disponibilizados (monitores coloridos, com maior resolução) começaram a surgir questões relacionadas aos melhoramentos que poderiam ser realizados no sentido de minimizar os problemas de interação com as máquinas de forma a tornar seu uso mais fácil e eficiente. Em função destes esforços, começaram a surgir vários padrões e coleções de regras de usabilidade as quais levam em conta fatores técnicos e aspectos psicológicos do processo de interação".*

Nielsen (1994) apresenta um conjunto de 10 regras heurísticas que foram usadas para examinar uma base de dados contendo 249 relatos de problemas de usabilidade. Aplicando-as para avaliar o sistema operacional como um todo, pode-se chegar a algumas conclusões e identificarem-se alguns problemas ainda sem solução imediata, os quais são discutidos a seguir.

5.5.1 Visibilidade do estado do sistema:

Segundo Nielsen (1994), o sistema deve sempre manter os usuários informados sobre o que está acontecendo, através de um feedback apropriado e dentro de um tempo razoável.

O nível de domínio do sistema restringe-se principalmente ao tratamento de exceções. Estas são de muito baixo nível e não agregam informação útil ao usuário. Se os programas são bem comportados, não há muito feedback a ser fornecido além dos tradicionais: quantidade de memória e disco disponíveis, percentagem de uso de processador, etc. Apesar de manter informações estatísticas e de estado dos periféricos e

demais componentes, estas informações não são imediatamente disponibilizadas às aplicações.

Hazzah (1997,pág.135) esclarece que no sistema operacional **Windows 95**, em determinadas situações, um *device driver* precisa escrever alguma informação na tela, mas como o subsistema de janelas está ocupado (*busy*), o que o sistema faz é chavear o modo de vídeo para modo texto e apresentar a mensagem neste modo.

Este é um exemplo de como o sistema operacional não tem controle absoluto sobre tudo o que nele está ocorrendo em determinado momento. Este tipo de ocorrência geralmente conduz a erros inesperados, visto que o sistema provavelmente entrou nesta situação a partir de uma transição de estados não prevista pelos projetistas.

5.5.2 Sincronismo (*match*) entre sistema e o mundo real

Segundo Nielsen (1994) o sistema deve falar a língua do usuário, com frases e conceitos familiares, ao invés de usar termos do domínio do sistema, e seguir convenções do mundo real, fazendo as informações aparecerem de forma lógica e natural.

Figura 34 - Mensagem de erro quando o usuário tenta remover um arquivo do S.O.

Esta questão ainda está longe de ser solucionada, visto que, como afirmado acima, o sistema possui pouco conhecimento sobre seu real estado e, por outro lado, desconhece o perfil do usuário e das aplicações[51]. Esta afirmação deve ser analisada sob o seguinte aspecto: por um lado, o sistema possui estruturas de dados que descrevem a utilização de recursos da máquina, mas, por outro, não tem como saber o que determinada aplicação está efetivamente fazendo. Ao entregar o processador para a aplicação executar, qualquer

[51] Apesar de haver pesquisas neste sentido, conforme apresentado nas seção 4.3.3.

coisa pode acontecer. A Figura 34 apresenta uma mensagem de erro quando o procedimento de instalação de uma nova aplicação tenta remover um componente do próprio sistema operacional. Daí a necessidade da criação de mecanismos de proteção (senhas, permissões, etc). Quando o mecanismo não é genérico o suficiente, problemas acontecem.

5.5.3 Controle do usuário e liberdade

Segundo Nielsen (1994), eventualmente os usuários escolhem alguma função por engano e necessitam de uma saída de emergência para sair deste estado sem ter que navegar por vários menus (Figura 35).

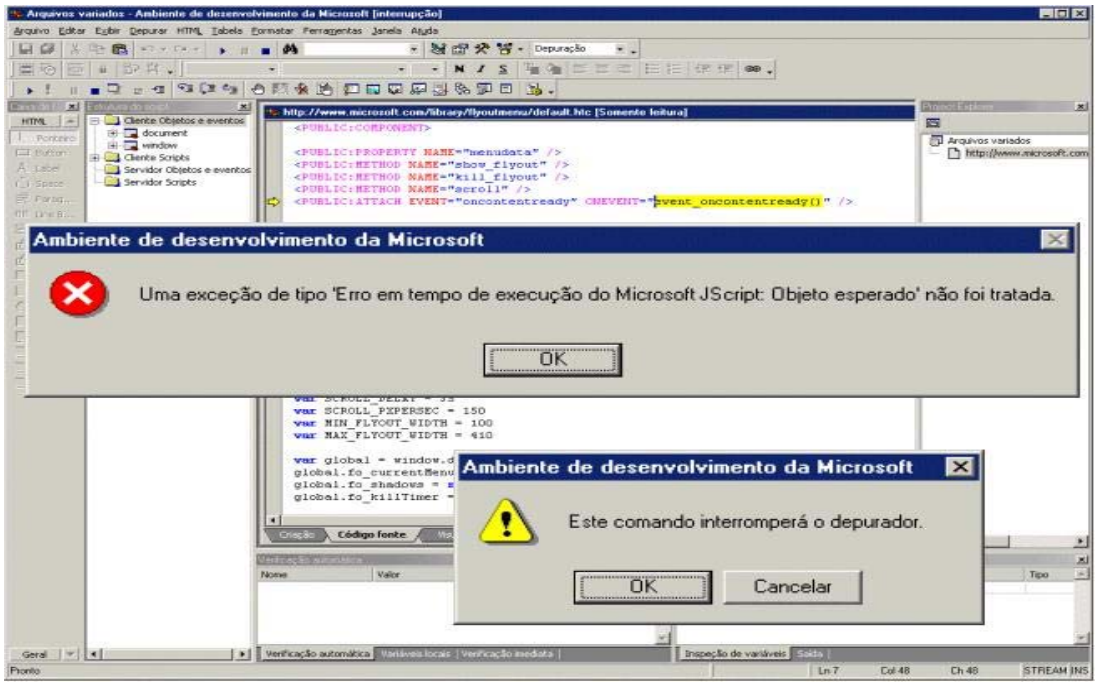


Figura 35 - Mensagem de aviso durante o cancelamento de operação com erro

Como o sistema não tem controle sobre a finalidade das aplicações, a busca dessa saída de emergência fica a cargo de cada aplicação, individualmente. Isto que faz com que o usuário necessite treinamento específico para cada tipo de aplicação e que ele perceba o comportamento geral do sistema como sendo orientado por aplicativos (Figura 36). É interessante observar que, por conta dessa anomalia, quando uma aplicação produz uma situação de erro a responsabilidade é geralmente imputada ao sistema operacional.

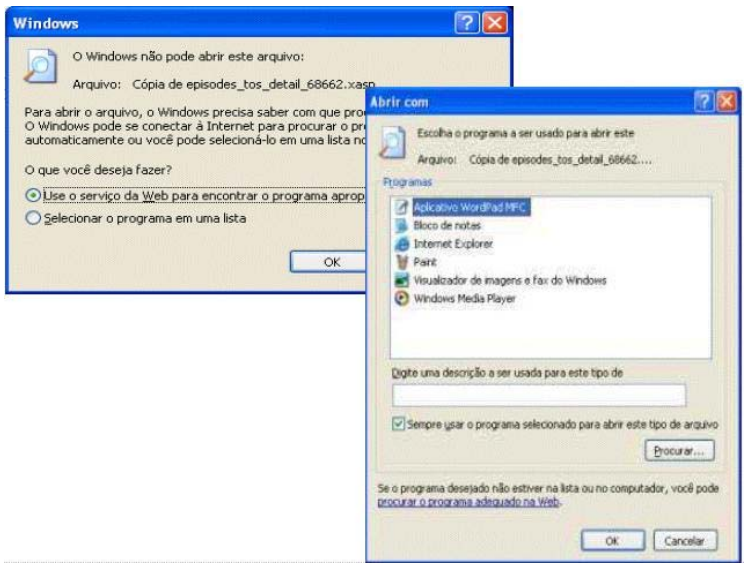


Figura 36- Associação entre formato de arquivo-aplicação.

5.5.4 Consistência e padronização

Segundo Nielsen (1994), os usuários não deveriam ter que se preocupar com diferentes palavras, situações ou ações que significam a mesma coisa. Em tais circunstâncias, eles devem seguir os padrões da plataforma:

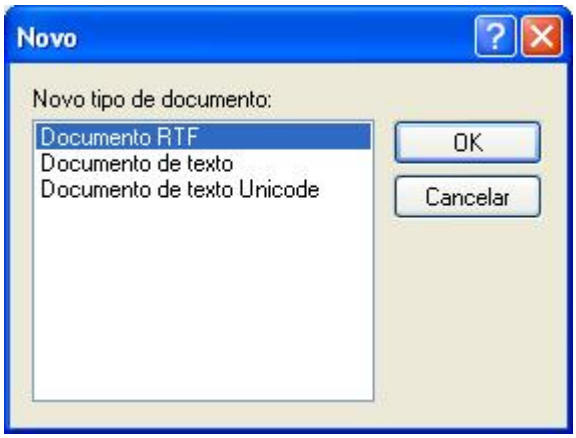


Figura 37- Uso de linguagem técnica na interface com o usuário.

Esta questão é interessante. Sendo uma camada de software genérico, um sistema operacional, geralmente por questões mercadológicas, tem que manter compatibilidade com versões antigas de aplicativos, ao mesmo tempo em que insere novos conceitos de interface e recursos adicionais. Esta situação pode conduzir a ambigüidades na medida em que tarefas podem ser executadas de várias formas, dificultando o aprendizado de usuários menos experientes.

5.5.5 Prevenção de erros

Segundo Nielsen (1994), melhor do que apresentar mensagens de erro é desenvolver um projeto que previna a ocorrência de problemas.

Esta questão remete a área de Engenharia de Software e todos os fatores associados a metodologias de desenvolvimento de sistemas, planejamento de testes e questões correlatas. No entanto, alguns problemas poderiam ser antecipados e solucionados pelo próprio sistema, ao invés de simplesmente reportar mensagens de erro. Pode-se imaginar a extensão do problema na situação em que determinado componente de um ambiente inteligente necessita ser reinicializado e solicita a confirmação do usuário antes de fazê-lo.

5.5.6 Identificar ao invés de relembrar

Segundo Nielsen (1994), os sistemas, de uma forma geral, deveriam tornar objetos, ações e opções visíveis. O usuário não deveria ter que lembrar informações de um diálogo para outro. Instruções para uso do sistema deveriam ser visíveis ou facilmente recuperáveis sempre que apropriado.

Este é um dos aspectos mais críticos na interface dos atuais sistemas operacionais. Quando um problema acontece, freqüentemente torna-se necessário percorrer vários menus e aplicações para relembrar configurações atuais e corrigi-lo a partir delas. Por exemplo: configurações de rede.

5.5.7 Flexibilidade e eficiência de uso

Segundo Nielsen (1994), aceleradores podem agilizar a execução das tarefas por usuários experientes Entretanto, em se tratando de usuários com pouca experiência, os aceleradores devem ser escondidos para permitir que eles utilizem conjuntos de ações freqüentes.

Aceleradores vêm sendo utilizados já há muito tempo em ambientes de sistemas operacionais, através de linguagens de scripts. Mas como seu uso requer um bom grau de conhecimento, eles se tornam ferramentas inúteis para usuários não especializados.

5.5.8 Projeto harmonioso e simples

Segundo Nielsen (1994), os menus não devem conter informações irrelevantes ou raramente utilizadas. Em um diálogo, cada unidade extra de informação compete com questões relevantes e diminui sua visibilidade relativa.

Esta facilidade já pode ser encontrada em sistemas (**Windows 2000**, **ME**) e aplicativos (ex: pacote Office da Microsoft) mais recentes. Sua principal característica é a limpeza das listas de opções dos menus. No mesmo pacote O*ffice* encontra-se a figura do agente que procura antecipar ações do usuário, auxiliando, assim, os menos experientes e conhecedores. Deve-se observar, contudo, que essas melhorias estão restritas a aplicações específicas, não beneficiando o conjunto de todas as aplicações.

5.5.9 Auxiliar o usuário a reconhecer, diagnosticar e recuperar-se de erros.

Segundo Nielsen (1994), mensagens de erros não devem ser expressas através de códigos. Ao contrário, devem indicar precisamente o problema e sugerir soluções de forma construtiva.

Este aspecto está longe de ser solucionado. A própria estrutura de API dos sistemas é orientada por códigos de erro, o que conduz a mensagens semelhantes às apresentadas na Figura 38 e na Figura 39. Há algumas iniciativas para minimizar o efeito dos códigos de erro através do uso de *wizards* (**Windows**). Sua abrangência, no entanto, ainda é limitada. O apêndice 4 (seção 10.3,pág.363), apresenta mensagens sobre erros na forma de registros de *log*.

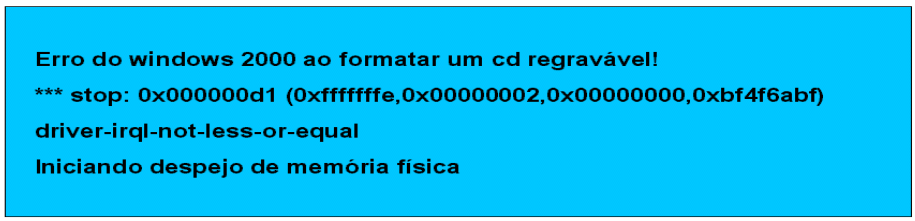


Figura 38- Exemplo de mensagem de erro crítico

Figura 39- Mensagens de erro dependentes de API.

5.5.10 Ajuda (*help*) e documentação

Segundo Nielsen (1994), mesmo que o sistema possa ser utilizado sem documentação, ela pode ser necessária para prover auxilio em determinado momento. Qualquer fonte de informação deve ser concisa, de fácil consulta e orientada para tarefas, listando passos concretos a serem seguidos.

Esses requisitos, de certo modo, são conflitantes com as características de um sistema operacional, devido à sua abrangência e generalidade.

Hugh (1997) finaliza suas considerações sobre *Human Computer Interaction* (HCI) afirmando que a questão é complexa e que efetivamente não há um padrão estabelecido. Este problema é potencializado à medida que novas aplicações são incluídas no contexto de espaços inteligentes *(Capítulo 3)*, onde a diversidade e a heterogeneidade de equipamentos interligados é muito maior e os requisitos de qualidade de serviço aumentam.

5.6 Outras Considerações

Conforme visto na seção 2.2, um dos objetivos de um sistema operacional é prover uma camada abstrata do hardware, permitindo, entre outras questões, o compartilhamento transparente dos recursos. Isto implica em que os analistas, ao conceber suas aplicações, e os programadores, ao implementá-las, consideram existir uma máquina virtual[52] cujos recursos estão sempre disponíveis à aplicação.

[52] Virtualizada através do conceito de multiprogramação.

Entretanto, esta não é a realidade que se percebe em um ambiente de execução. O fato concreto é que os programas concorrem pelo uso dos recursos e que esta concorrência pode acarretar flutuações na disponibilidade dos mesmos. Estas flutuações podem ser de intensidades variadas, como mostra a Figura 40 (consumo de tempo de processador numa operação de indexação de arquivos).

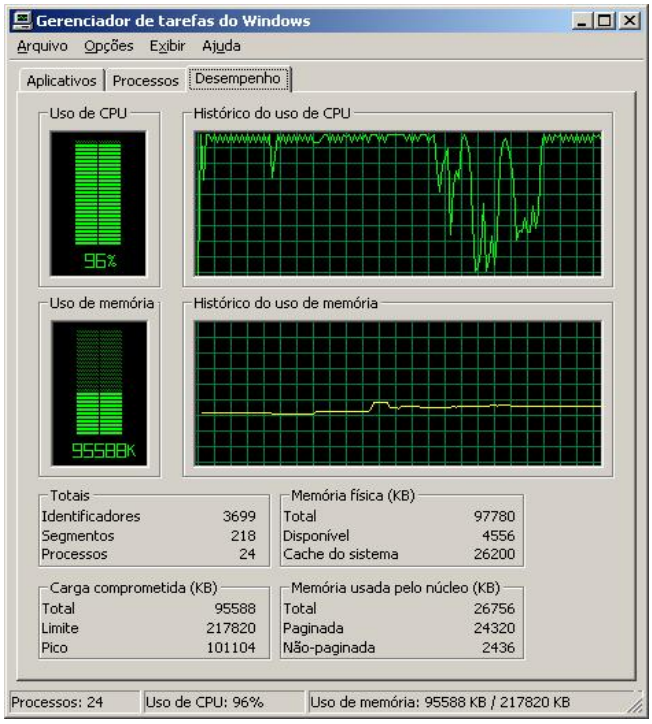


Figura 40 - Flutuações na disponibilidade de processador

Neste exemplo, a flutuação de disponibilidade do processador é evidente. No entanto, as aplicações não são construídas de forma a adequar-se a esta realidade. Assim, o que ocorre é que as filas do sistema começam a aumentar, causando um aparente "peso" na máquina. Ou seja, o tempo de resposta às intervenções do usuário passa a ocorrer com um atraso perceptível, lançando por terra o conceito de que o computador é uma máquina com recursos virtualmente infinitos.

Ao se deparar com situações como estas, os usuários geralmente consultam o "estado do sistema" através de comandos (ou utilitários) apropriados (***sar*** no **Unix**[53] e *Gerenciador de Tarefas* no **Windows**), os quais são implementados segundo alguma das estratégias apresentadas na seção 2.5. Esta atitude geralmente compromete ainda mais os

[53] Ver apêndice 6 (seção 10.5, pág 368) para um exemplo de saída de um comando sar.

tempos de resposta do sistema, uma vez que a obtenção das informações sobre seu atual estado também consome recursos computacionais.

5.7 Domínio do Problema

A questão central abordada neste trabalho é a inadequação dos sistemas operacionais para fazer frente às demandas cada vez maiores dos usuários. Esta afirmação pode ser constatada a partir da análise dos capítulos precedentes, onde fica clara a tendência de que o sistema operacional, apesar de sua importância fundamental, não consegue acompanhar a evolução tecnológica. Em função disto, à medida que surgem soluções temporárias, novas fontes de instabilidade são agregadas ao conjunto de problemas potencias a que o usuário está sujeito.

Um dos problemas recorrentes nas soluções apresentadas é a construção de aplicações específicas para coletar dados estatísticos sobre vários aspectos de baixo nível do sistema operacional. Algumas aplicações são concebidas para fazer uso desses dados, a fim de adequar-se a flutuações ambientais. Contudo, há uma série de outras aplicações que não são concebidas com essa preocupação, inclusive as do próprio sistema operacional. Nesse contexto, a viabilidade da execução de uma aplicação **A** pode ser afetada por sua inabilidade em adaptar-se às aplicações **B** e **C**.

O uso de tecnologias apropriadas permite que pessoas ou organizações tenham acesso a informações do usuário sem o seu consentimento e, o que é pior, de forma absolutamente transparente. Um exemplo disto é apresentado na Figura 41, onde o usuário tem a recepção do e-mail confirmada para o remetente mesmo sem desejar.

As seguintes considerações relacionam os aspectos mais importantes do resumo das discussões do evento ECOOP'99 - 2nd ECOOP *Workshop on Object-Orientation and Operating Systems* (1999):

- Percebe-se um movimento no sentido de substituir o uso de grandes computadores por pequenos computadores interligados em rede;
- Há 2 anos atrás a adaptabilidade constituía a maior discussão. Hoje, melhoramentos em adaptabilidade através de ferramentas com esta finalidade específica, automatização de decisões de adaptação e *scaling adaptability*, são os tópicos em discussão.
- Ao que tudo indica, sistemas baseados em conhecimento são o caminho para auxiliar os administradores a gerenciar sistemas altamente adaptáveis.

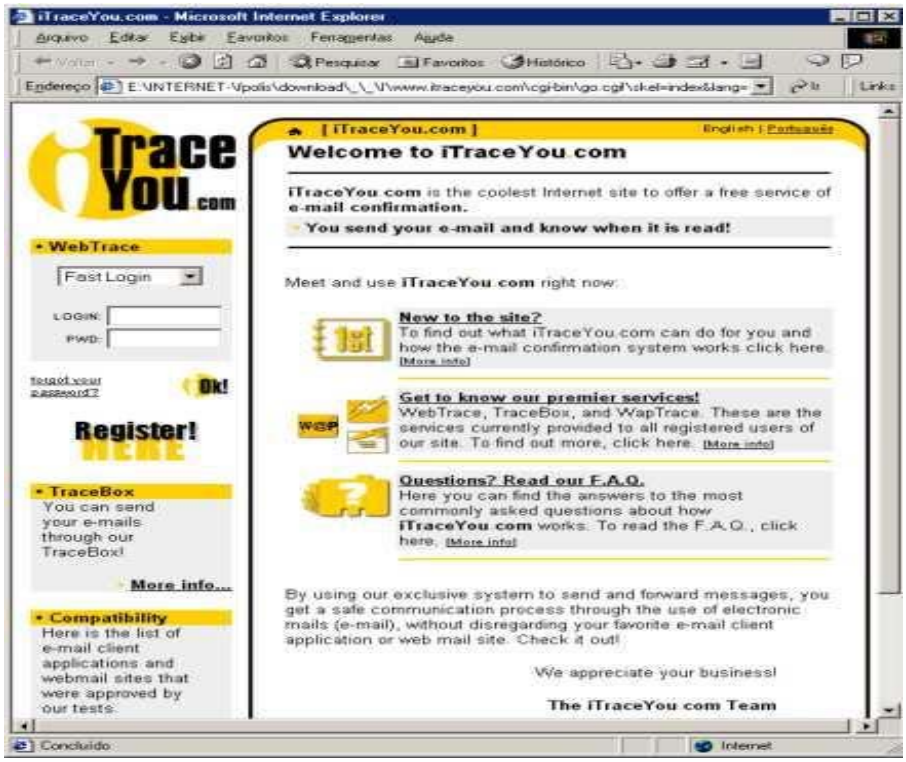


Figura 41 – Tecnologia usada para coletar dados sem que o usuário tenha autorizado

- Ainda há pontos relacionados à base de conhecimento a serem definidos como: (i) Onde a base de conhecimento deve ficar? No sistema local ou distribuída? (ii) Onde colocar o conhecimento sobre questões de distribuição? (iii) Como efetivá-la e tomar decisões a partir dela?
- A questão da linguagem a ser utilizada pelos programadores, se Prolog ou outra mais poderosa, ainda não foi resolvida. Uma possível solução pode ser a combinação de linguagens simples para questões de conhecimento e uma linguagem mais poderosa para questões mais complicadas.
- Relativamente à aquisição de conhecimento deve-se: (i) utilizar a engenharia de conhecimento (estudando o código fonte, comentários, documentação, entrevistas com administradores e traduzindo as informações através de regras manualmente); (ii) analisar-se a estrutura do sistema e ferramentas (extraindo a hierarquia de classes a partir do código fonte de um sistema orientado a objetos); e (iii) observar o sistema em tempo de execução. Este último item é mais difícil, pois obter dados em tempo de execução, sem perturbar o sistema base, e derivar regras das seqüências de eventos, é uma tarefa bastante complexa.
- Como resultado pode-se esperar um grande esforço de pesquisa neste sentido, a médio e longo prazo. Conclui o relatório que o conceito de meta-programação representa um vasto campo de pesquisas. É importante que os futuros sistemas tenham como objetivo chegar a meta conceitos simples. Talvez a área de *aspect-orientation* (AOP) seja uma direção provável (ECOOP'99,1999).

Além destas considerações, a visão de uma quantidade enorme de pequenos computadores envolvendo (*surrounding*) as pessoas "mostra que nós precisamos repensar

os conceitos estabelecidos, tais como a distinção entre memória transiente e persistente e o fim da era PC"(ECOOP'99,1999).

A próxima seção apresenta uma tabela comparativa entre os requisitos de projetos de sistemas operacionais, computação ubíqua, teletrabalho, comércio eletrônico e robótica.

5.8 Análise comparativa

A partir da análise realizada nos capítulos precedentes produziu-se a Tabela 24(pág.175), que confronta os requisitos de projetos de sistemas operacionais, computação ubíqua, teletrabalho, comércio eletrônico e robótica. O que se pode observar dessa análise é que há uma disparidade de interesses e demandas, o que pode ser um indicativo de resposta para as questões apresentadas neste capítulo.

Tabela 24 - Comparativo entre requisitos de projetos de sistemas operacionais, computação ubíqua, teletrabalho, comércio eletrônico e robótica.

Requisitos de Sistemas Operacionais	Requisitos de Computação ubíqua, Teletrabalho e Comércio Eletrônico.	Requisitos de Robótica
• Estrutura organizacional (monolítico, camadas, etc). • Performance do núcleo/ serviços do sistema.		• Modularidade e flexibilidade.
• Modelo de aplicação com código compilado estático com suporte a: o *Late binding;* o Código interpretado; o Pesquisas em paralelismo de instruções (*pre-fetch*). • Políticas de segurança através de permissões, restrição de acesso, *capabilities*. • Suporte a primitivas de sincronização e intercomunicação de processos (*threads*). • Suporte a uma ou mais políticas de escalonamento. • Suporte a uma ou mais estratégias de gerenciamento de memória. • Suporte a tratamento de exceções.		
• Suporte a gerência de meios de armazenamento. • Interface do sistema com as aplicações através de APIs.	• Sistema de arquivos com maior nível de abstração.	
• Facilidade de extensão dos serviços do sistema para permitir a adaptação por parte das aplicações. • Suporte à multitarefa. • Compatibilidade (ou não) com versões anteriores. • Facilidades de intercomunicação e interligação com outros equipamentos.	• Facilidades para usuários especializados aumentarem as bibliotecas do sistema operacional sem comprometimento dos aspectos de segurança e confiabilidade.	• Adaptabilidade. • Facilidade de expansão.
• Interação com o usuário através de interfaces gráficas (e mais recentemente através de interação por voz). • Controle de recursos. • Robustez. • Política de segurança.	• Interação com o usuário. • Controle de recursos com qualidade de serviço. • Robustez. • Segurança.	• Interação entre Agente-Agente e Agente-Humanos. • Robustez e confiabilidade.

• Administração do sistema.		
	• Mobilidade de código estático e dinâmico.	• Mobilidade.
	• Capacidade de tratar informações parciais e/ou imprecisas.	• Resolução de múltiplos objetivos.
	• Capacidade de extração de conceitos.	
	• Capacidade de raciocínio aproximado (difuso) e de aprendizado.	• Raciocínio global.
	• Identificação de perfil de usuário e adequação do ambiente a este perfil.	
		• Reatividade ao ambiente.
		• Integração de múltiplos sensores.
		• Autonomia.
		• Comportamento inteligente.
	• Privacidade de informações sobre transações.	
	• Confidencialidade das informações.	
	• Integridade das informações.	

5.9 Considerações finais

Neste ponto do trabalho inserimos a Figura 42 que representa o contexto onde os projetistas de sistemas operacionais trabalham e, em última instância, a visão que o sistema operacional tem do contexto onde está operando.

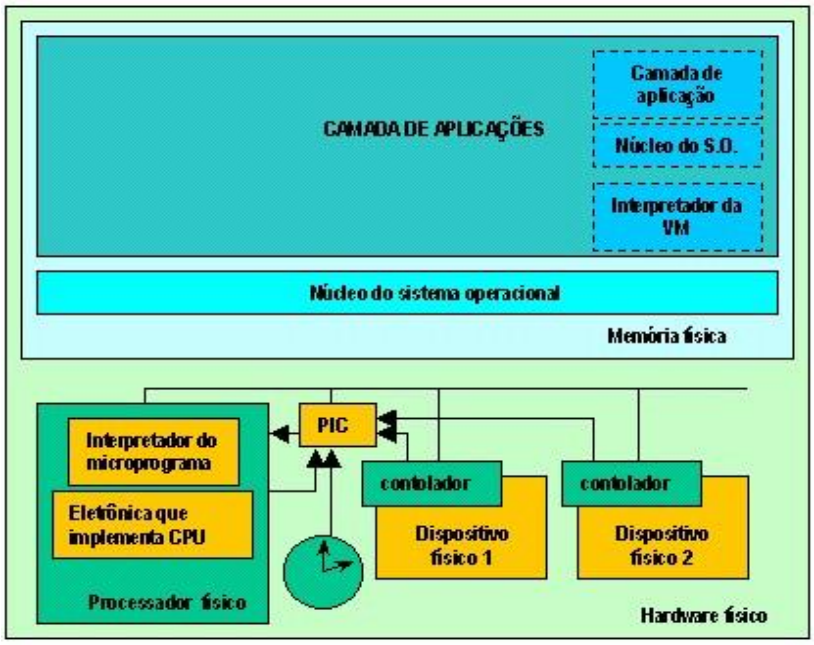


Figura 42 – Contexto onde os projetistas de SO trabalham

Os projetistas de sistemas operacionais consideram basicamente as características do hardware sobre o qual o sistema operacional será construído, tais como:

- Velocidade de processador;
- Formato de instruções;
- Freqüência de relógio;
- Características de barramento;
- Características dos controladores de periféricos, etc.

Por outro lado, inserimos a Figura 43 que representa a visão sobre a qual está baseada toda a tecnologia de computação. Esta visão pode ser resumida da seguinte forma: cada aplicação em particular é concebida para ser executada em uma máquina:

- Abstrata exportada pelo conjunto hardware físico + sistema operacional;
- Contendo dispositivos lógicos sempre disponíveis e sem limite de tamanho;

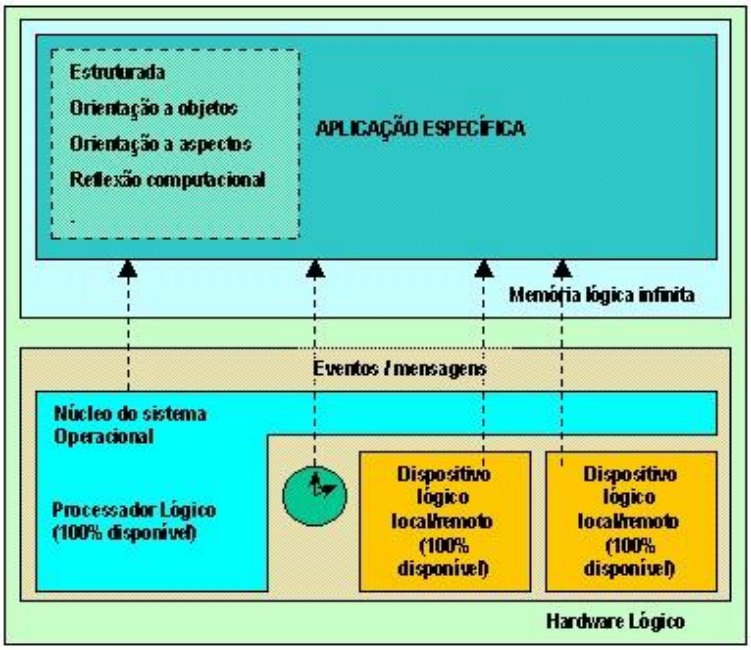


Figura 43 - Visão sobre a qual está baseada a tecnologia de computação.

- Com um processador 100% do tempo disponível para a aplicação[54];
- Tendo como meio de comunicação de ocorrências físicas ou lógicas a abstração de mensagens;
- Com uma capacidade de memória infinita[55];
- E que, através de meta-canais de comunicação (RPC, Mensagens, *Jini*, CORBA, RMI), pode comunicar-se com outras aplicações (locais ou remotas) e com elas trocar informações de alto nível.

Sobre esta máquina abstrata é que são desenvolvidas e projetadas soluções de Computação Ubíqua, Comércio Eletrônico, Ensino à Distância, Teletrabalho e as demais tecnologias de informação conhecidas. Os métodos de concepção e ferramentas de desenvolvimento de software também utilizam esta máquina abstrata como premissa básica. As soluções para os problemas apresentados anteriormente também partem desta visão. A Figura 44 confronta as duas visões as quais são separadas pela linha pontilhada.

O aspecto mais importante a ser ressaltado é que a visão que os *hackers* geralmente utilizam para a construção de vírus de computador e a partir da qual realizam seus ataques, ignora as abstrações promovidas pelo uso de tecnologias como Orientação a Objetos/Aspectos, Reflexão Computacional ou qualquer outra tecnologia empregada e segue rigorosamente a visão apresentada à esquerda da linha pontilhada na Figura 44.

[54] Exceto para aplicações tempo-real, onde a dimensão tempo é um requisito presente no momento da concepção do software.
[55] Exceto quando se desenvolve software embarcado, o qual geralmente apresenta restrições de memória.

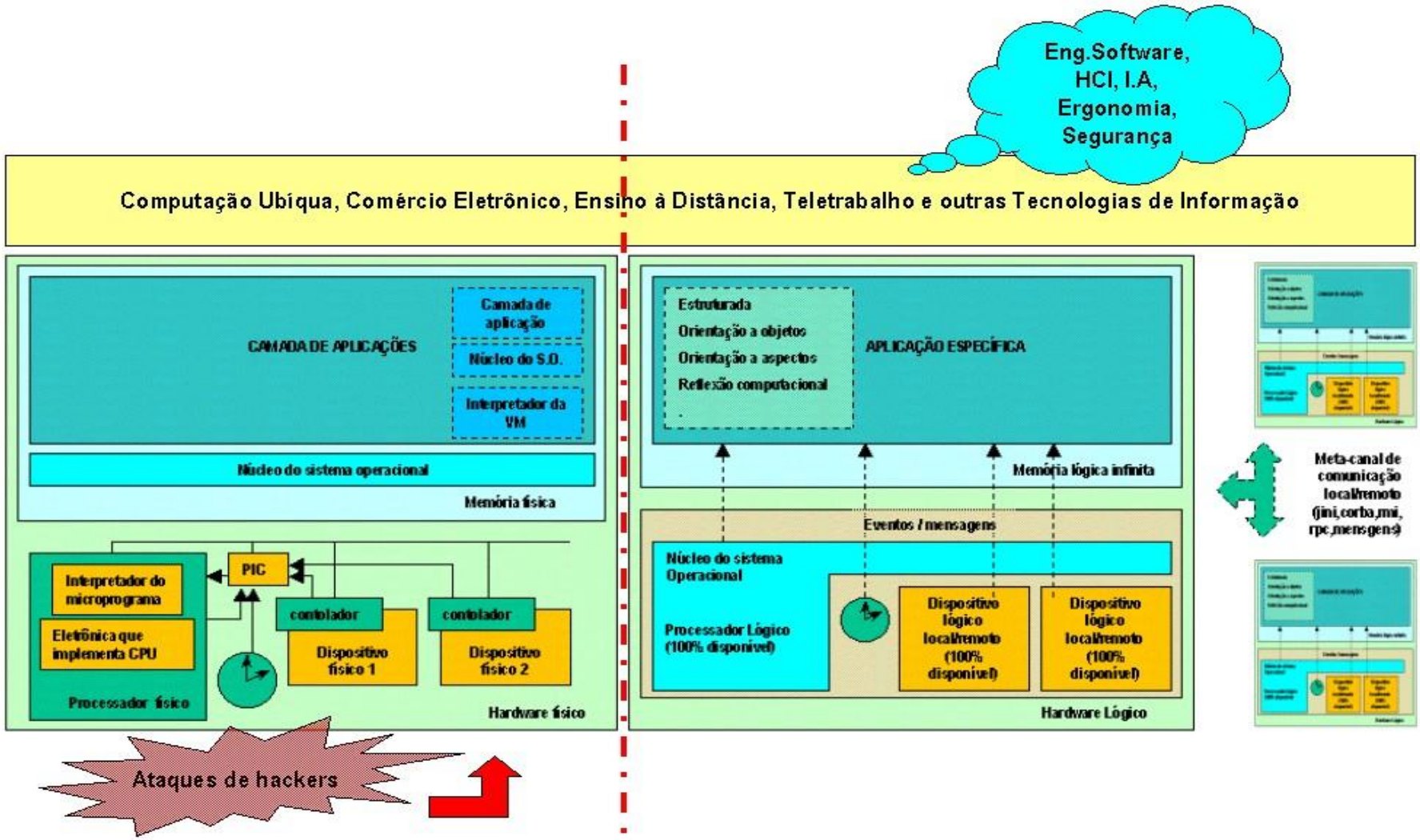


Figura 44 – Confronto entre a visão de projeto de SO e a visão geral do atual modelo de computação.

As soluções de segurança geralmente utilizam o sistema de arquivos para armazenamento das senhas, embora criptografadas. Ou então envolvem listas de controle de acesso ou permissões de acesso a um sistema de arquivos em particular. Ocorre que a abstração de sistema de arquivos é muito alta. Ela existe desde há muito tempo para facilitar o processo de desenvolvimento de aplicações e não para armazenar dados importantes. A melhor solução de controle de acesso pode ser facilmente quebrada a partir de alguma das seguintes estratégias:

- Acesso ao dispositivo físico através da instalação de mais de um sistema operacional na mesma máquina; ou,
- A instalação de um disco rígido (ilicitamente obtido) como dispositivo secundário em uma outra máquina de mesma arquitetura, disponibiliza o acesso às informações lá armazenadas, à revelia de senhas, listas de controle de acesso e toda a parafernália de segurança projetada a partir das abstrações apresentadas à direita da linha pontilhada.

Além disso, o enfoque principal concentra-se no sentido de instrumentalizar o sistema operacional a partir do uso de técnicas de extração de informações apresentadas no capítulo 2. Observa-se claramente que as abordagens, via de regra, localizam-se na fase 2 do ciclo evolutivo dos sistemas operacional, também apresentada no capítulo 2. Ou seja, baseiam-se na construção de bibliotecas especializadas que executam sobre o sistema, as quais, numa versão seguinte, poderão incorporar o escopo do sistema ou não (dependendo de fatores estratégicos geralmente orientados pelo mercado). Para isto lança-se mão das seguintes técnicas: Inteligência Artificial, Agentes, Análise de perfil baseada em histórico e Visualização de informação entre outras.

Assim sendo, tomando-se a Tabela 24 como referência e comparando-se:

- as características apresentadas e os requisitos dos projetos de sistemas operacionais identificados no capítulo 2,
- os requisitos de computação ubíqua a partir da análise apresentada no capítulo 3,
- a forma como as demandas não previstas são corrigidas a partir da análise apresentada no capítulo 4 e,
- os requisitos dos projetos de robótica a partir da análise apresentada na seção 4.4,

percebe-se claramente que a maioria destas questões permanece em aberto ainda hoje.

É interessante observar que a categoria de projetos de robótica transcendeu há muito tempo ao modelo de "fazer-pela-máquina", ou seja, transcendeu ao modelo de programação pré-estabelecida da seqüência de ações a serem executadas.

Isto foi possível a partir da adoção de quatro conceitos importantes:

- O conceito de mundo;
- O conceito de incorporação (*embodiment*);
- O conceito de planos em substituição ao conceito de programa; e
- O conceito de aprendizagem.

Estas questões dominaram o centro das discussões na primeira fase dos projetos de robótica e de Inteligência Artificial Simbólica. Enquanto isso, os projetos de sistemas operacionais tradicionais mantêm-se fiéis ao modelo de Entrada-Processamento-Saída e ao conceito de multiprogramação (que virtualiza o *hardware*).

Como visto, as propostas discutidas neste capítulo, bem como aquelas apresentadas nos capítulos precedentes, embora eficazes no que se propõem a resolver, tratam o problema pontualmente visto que não basta, por exemplo, ajustar a qualidade de serviço de uma aplicação multimídia para considerá-la um sistema inteligente e amigável.

O aspecto mais relevante deste capítulo é o fato de que, apesar do enorme esforço desenvolvido pelos pesquisadores da área, problemas identificados por Linde em 1975 e, portanto, há aproximadamente 28 anos atrás, ainda continuam presentes e recorrentes. Isto permite concluir, irrefutavelmente, que as abordagens atuais não são suficientemente adequadas para atender aos requisitos esperados dos sistemas computacionais atuais.

O próximo capítulo apresenta o modelo proposto de solução para os problemas apresentados anteriormente, o qual deve congregar os melhores esforços de cada área estudada no sentido de produzir um sistema operacional que verdadeiramente apresente um comportamento inteligente frente ao usuário.

6 BASE CONCEITUAL DO MODELO PROPOSTO

"Tudo indica que o computador está se tornando o correlativo contemporâneo da máquina a vapor que propiciou a revolução industrial. O computador é uma máquina de informação. Informação é uma utilidade não menos intangível que a energia e quando muito, é mais infiltradora nos assuntos humanos. O controle da informação possibilitado pelo computador torna possível também inverter as tendências para a uniformidade da produção em massa iniciada pela revolução industrial. O aproveitamento desta oportunidade pode apresentar as questões mais urgentes no campo de engenharia, bem como no campo político e social da próxima geração" John McCarthy(1966).

6.1 Introdução

O presente capítulo apresenta a base conceitual a partir da qual o modelo proposto foi concebido. Ele inicia com as principais considerações a respeito de comportamento inteligente e conhecimento, os quais fazem parte da definição que será apresentada no próximo capítulo. A seguir são discutidos aspectos relacionados à questão da percepção do tempo como base para a criação de regularidades estatísticas históricas – a base para a construção do conhecimento. O formalismo DEVS é introduzido a seguir, como parte da estratégia de concepção e viabilização do conceito de tempo implícito e, finalmente, é apresentado o conceito de modelo de mundo utilizado nos projetos de robôs autônomos.

6.2 Comportamento inteligente e conhecimento

"(...) a inteligência deverá ser um elemento chave no projeto de futuros sistemas operacionais" Fleish (1983).

Como o objetivo geral é a elaboração dos fundamentos conceituais de um novo modelo de sistema operacional baseado na abstração de conhecimento e comportamento inteligente, faz-se necessário explicitar estes dois conceitos.

Deve-se destacar que, no presente trabalho pretende-se abordar a questão do conhecimento sob uma perspectiva antropológica, isto é, como uma ferramenta produzida pelo homem e históricamente realizada. Neste sentido, a questão epistemológica do que é conhecimento (que procura identificar as condições de possibilidade metafísica do conhecimento) está fora do escopo do trabalho.

Bittencourt (1998) em seu trabalho "Representação do Conhecimento: da metafísica aos programas", procura relacionar as técnicas computacionais utilizadas em representação de conhecimento com seus fundamentos filosóficos e matemáticos.

Segundo o autor "estes aspectos em geral não são abordados em cursos de representação de conhecimento e, no entanto, são fundamentais para o seu entendimento". O trabalho, além de caracterizar o sentido das palavras "formalização" e "computação", procura esclarecer o sentido de frases do tipo: "*este programa é inteligente*" ou "*este programa tem representado todo o conhecimento a respeito de tal assunto*".

Segundo Costa (1993) a área de inteligência artificial possui várias correntes filosóficas (simulação do pensamento, pragmatista, logicista, neurofisiologismo, agentes autônomos e vida artificial). Contudo, há uma marca fundamental que une estas correntes "ambas se valem de uma comparação com uma inteligência natural, humana ou animal, para medir o sucesso de suas construções". Esta abordagem coloca o desenvolvimento de máquinas, programas e sistemas numa perspectiva que Costa (1993) denominou: "substituição estrutural com preservação funcional". Portanto, independentemente das diversas intenções com que os trabalhos são concebidos, a intenção mais geral que termina dominando todos eles é a de buscar "uma artificialização de entidades, isto é, de fazer com que onde antes funcionava um ser vivo, agora passe a funcionar uma máquina" (COSTA,1993).

Costa (1993) classifica esta concepção de inteligência artificial como uma visão artificialista e, em contrapartida, em sua tese de doutorado apresenta a visão de inteligência de máquina como um fenômeno natural – a visão naturalista[56].

> *"Por conteúdo real e específico da noção de inteligência de máquina entendemos um fenômeno real e específico que ocorre nas máquinas, assim como a inteligência humana é um fenômeno real e específico que ocorre em seres humanos. Com isso, também queremos propor que a atitude a ser tomada no trabalho em inteligência artificial com relação ao estudo da inteligência de máquina é uma atitude de investigação empírica e experimental, com forte carga de formalização, dada a natureza do trabalho de pesquisa sobre máquinas computadoras, porém não uma atitude de engenharia artificialista" Costa(1993,pág.18).*

Estabelece Costa (1993):

> *"Inteligência de máquina é o termo final do desenvolvimento da estrutura de regulação das interações funcionais da máquina com o ambiente. Sendo os objetos físicos despidos, em si mesmos, de qualquer significação, os significados de um objeto surgem, então, somente pela sua inserção em um sistema de operações realizadas real ou potencialmente por um agente, e só existem para esse agente e são relativos à medida e à maneira como ele os insere nesse seu sistema de operações. Ou seja, os significados são atribuídos aos objetos físicos, e não extraídos deles. Se esta significação é puramente sensório-motora ou se é representativa, se é não intencional ou se é intencional, isto depende da estrutura do agente e não especialmente do objeto físico".*

[56] Cabe destacar que, em seu trabalho, Costa (1993) está interessado em compreender o conhecimento como ferramenta que tem sua propriedade independente do sujeito, ou seja, inteligência como racionalidade (a máquina tem que ter uma propriedade que constitui na capacidade de se organizar por si própria).

Segundo Sowa (2000), as palavras conhecimento e representação têm provocado controvérsias filosóficas desde o tempo de Sócrates. Complementam Rich e Knight (1991, pág.9), afirmando que,

> *"Um dos poucos resultados rápidos e difíceis a surgir nas três primeiras décadas da pesquisa em IA é que a inteligência requer conhecimento. Para compensar sua principal característica, a indispensabilidade, o conhecimento possui algumas propriedades menos desejáveis, incluindo: (i) ele é volumoso; (ii) é difícil caracterizá-lo com precisão; (iii) ele está mudando constantemente; (iv) ele difere de simples dados por organizar-se de uma maneira que corresponde ao modo como será usado".*

Neste sentido, as técnicas de IA nada mais são do que métodos que exploram o conhecimento, o qual deve ser representado de tal forma que (RICH e KNIGHT,1991, pág.9):

- O conhecimento capture generalizações;
- Ele precisa ser compreendido pelas pessoas que o fornecem;
- Ele pode ser facilmente modificado para corrigir erros e refletir mudanças do mundo e da nossa visão do mundo;
- Ele pode ser usado em inúmeras situações, mesmo que não seja totalmente preciso nem esteja completo;
- Ele pode ser usado para ajudar a superar seu próprio volume, auxiliando a limitar as várias possibilidades que, em geral, têm de ser consideradas.

Segundo Sowa (2000), a representação do conhecimento é um assunto multidisciplinar que utiliza teorias e técnicas de outros três campos:

- Lógica: fornece a estrutura formal e as regras de inferência;
- Ontologia: define os tipos de coisas que existem no domínio da aplicação;
- Computação: suporta as aplicações que distinguem representação de conhecimento de filosofia pura.

Sem a lógica, uma representação de conhecimento é vaga, sem critérios para determinar se os comandos são redundantes ou contraditórios. Sem Ontologia, os termos e símbolos são mal-definidos, passíveis de serem confundidos e de produzir confusão. E sem modelos computáveis, a lógica e a ontologia não podem ser implementadas em programas de computador. Portanto, a representação do conhecimento é a aplicação de lógica e ontologia à tarefa de construir modelos computacionais para algum domínio (SOWA,2000, pág. XII).

Os aspectos envolvidos na criação de um programa inteligente baseado em uma representação simbólica do mundo tem a mesma natureza e complexidade de problemas

que vêm ocupando filósofos, especialmente nas áreas de metafísica, lógica e epistemologia (BITTENCOURT,1998), os quais podem ser relacionados a seguir:

- Que representação é adequada para ser enriquecida com novas experiências?
- Além de representações para objetos do mundo físico, que outros tipos de representação são úteis ou necessários?
- Como obter novos conhecimentos a partir de observações do mundo e da estrutura de representação interna?
- Em que medida uma representação pode ser utilizada para representar conhecimento sobre a própria representação;
- Em que linguagem deve-se representar o conhecimento?

Com muita propriedade Bittencourt (1998) coloca que, "apesar da pertinência destas perguntas para a fundamentação teórica da IA, o simples fato de que um programa deva ser implementado em um computador tendo como único mundo externo operadores humanos ou periféricos eletrônicos, implica certas pressuposições que tornam a maioria dessas questões filosóficas irrelevantes.

As pressuposições a que se refere Bittencourt são baseadas nas seguintes afirmações de McCarty e Hayes (1969):

- Existe o mundo físico e existem pessoas inteligentes;
- Informações sobre o mundo podem ser obtidas pelos sentidos e representadas internamente;
- O senso comum e o conhecimento científico atual correspondem a uma visão do mundo "aproximadamente" correta;
- O conhecimento sobre o mundo não precisa, necessariamente, obedecer a alguma propriedade matemática global, como coerência ou completude. Sua adequação deve ser medida pela sua utilidade na solução de problemas práticos;
- Para fundamentar a especificação de um programa computacional, um sistema filosófico deveria apresentar um tal grau de precisão em suas definições que descarta a maioria dos sistemas propostos em filosofia por serem excessivamente vagos;
- Uma pessoa, ao escolher entre alternativas, avalia as conseqüências de suas possíveis ações. Um programa que faça o mesmo, sendo uma máquina determinística, só pode escolher em um sentido restrito e formal. Isto torna concreto o problema do livre arbítrio, no sentido em que toda possível liberdade de escolha de um programa deve, paradoxalmente, ser programada.

A teoria clássica da IA, tomando por base estas pressuposições, tem buscado, incessantemente, uma representação de mundo coerente com o senso comum e precisa a ponto de permitir que um programa que a tome por base apresente um comportamento interessante. Assim, as possíveis representações dos aspectos da realidade, relevantes a

uma visão de mundo coerente com o senso comum, podem ser analisadas segundo a sua adequação frente aos fatos do mundo real e aos problemas que se pretende solucionar (BITTENCOURT,1998).

McCarty e Hayes apud Bittencourt (1998) relacionam três tipos de adequação para uma representação de conhecimento:

- Metafísica: se um mundo construído de acordo com a representação não apresenta contradições com os fatos pertinentes aos aspectos da realidade que desejamos representar;
- Epistemológica: se a representação pode ser utilizada, na prática, para representar os fatos disponíveis sobre os aspectos de interesse da realidade;
- Heurística: se os processos de raciocínio necessários para a solução dos problemas de interesse podem ser expressos na representação.

Bittencourt (1998) complementa que,

> *"Existem diversos exemplos de representações metafisicamente adequadas para expressar conhecimentos científicos ou práticos a respeito de objetos do mundo físico –por exemplo, o modelo atômico de Demócrito, a teoria das mônadas de Leibniz, ou o modelo padrão da física moderna que, no entanto, não são epistemologicamente adequadas para representar os conhecimentos necessários para resolver problemas típicos do cotidiano. O contrário, em geral, não ocorre: representações com possibilidades de serem epistemologicamente adequadas para representar o senso comum são, na maioria dos casos, plenamente adequadas metafisicamente. Dessa maneira, a adequação metafísica não é um problema central para a IA".*

Para afirmar que um programa age de uma maneira inteligente, Bittencourt (1998) estabelece que "antes é necessário explicitar o significado do verbo ´poder´ quando o sujeito é um programa".

Segundo Bittencourt (1998),

> *"McCarty e Hayes introduzem o conceito de sistema de autômatos para caracterizar esta questão. Um autômato é uma máquina abstrata caracterizada por um estado interno **S** e que, a cada entrada recebida, produz uma saída deterministicamente especificada pelo estado atual do autômato e pelo valor da entrada. De acordo com a representação proposta no exemplo, cada entidade do mundo e o mundo como um todo são vistos como sistemas de autômatos em interação. Cada entidade separada do mundo será associada a um subsistema de autômatos, que pode ser caracterizado pelas suas entradas e saídas, isto é, pelas conexões entre autômatos do subsistema e autômatos do restante do sistema global. A partir deste modelo, pode-se afirmar que, uma entidade representada por um subsistema de autômatos **P** "**pode**" realizar "**algo**", representado no caso por um estado do sistema global de autômatos, se suas saídas, ao serem manipuladas como se fossem entradas externas ao sistema global, permitirem a ocorrência de uma seqüência de eventos tal que o autômato global alcance o estado desejado".*

Embora Bittencourt (1998) afirme que:

> *"Apesar da representação por autômatos não ser epistemologicamente adequada para a representação de comportamentos inteligentes complexos tendo em vista que mesmo um processo simples necessitaria de um número astronômico de estados (a grande maioria dos mesmos seria desconhecido"),*

ele concorda que,

> *"a simplicidade da representação permite definir de maneira clara o que poderia significar poder e acreditar em relação a um programa, além de ressaltar que tal significado depende não só do mundo externo (o sistema global de autômatos) e das possíveis interações entre seus componentes, mas principalmente da delimitação do subsistema considerado."*

Contrariando a primeira afirmação de Bittencourt, o modelo de mundo proposto neste trabalho está justamente baseado no conceito de mundo (seção 6.5$_{\text{Pág.201}}$) e nas inúmeras possibilidades de combinações. E o modelo de execução está baseado no conceito de autômatos a partir da junção do formalismo **DEVS** (seção 6.4$_{\text{Pág.198}}$) com o modelo de execução apresentado por Schaad (seção 6.6$_{\text{Pág.205}}$) conforme será apresentado no próximo capítulo.

Com o desenvolvimento da pesquisa em IA simbólica, aos poucos estabeleceu-se um certo consenso sobre as propriedades básicas que deve possuir qualquer processo capaz de raciocinar inteligentemente sobre o mundo. Segundo Bittencourt (1998),

> *"Brian Smith no prólogo de sua tese de doutorado introduz a chamada hipótese de representação de conhecimento afirmando que: qualquer processo inteligente realizado por uma máquina, será formado por ingredientes estruturais que (a) serão naturalmente percebidos pelos observadores externos como uma descrição proposicional do conhecimento exibido pelo processo, e (b) independentemente de tal atribuição semântica externa, terão um papel formal, causal e essencial na geração do comportamento que manifesta tal conhecimento" Bittencourt (1998).*

Esta hipótese implica em que todo processo inteligente necessita de um interpretador capaz de manipular representações percebidas por um observador externo como descrições proposicionais do conhecimento exibido pelo processo, de forma a gerar um comportamento inteligente. Além disso, um requisito fundamental para que um processo exiba uma certa sofisticação intelectual, é a capacidade de reflexão. Isto porque, sem poder raciocinar sobre suas próprias capacidades e limitações, um programa dificilmente conseguirá traçar estratégias para a solução de problemas complexos (BITTENCOURT,1998).

A partir desta constatação, Smith apud Bittencourt (1998) em sua hipótese da reflexão afirma que:

> *"Se um processo inteligente pode ser construído para raciocinar sobre o mundo externo graças a um processo interpretador e a representações formais adequadas, então também deve ser possível construir um processo que raciocine sobre si mesmo, a partir de um processo interpretador similar e representações formais sobre a estrutura de suas representações e sobre sua operação."*

Kirsh apud Bittencourt (1998), identifica as seguintes suposições que orientam e dividem opiniões entre pesquisadores da área de IA, sobre como a inteligência pode ser capturada por um mecanismo artificial:

- A simulação da inteligência requer conhecimento declarativo e algum tipo de mecanismo de raciocínio;
- A inteligência pode ser estudada independentemente dos detalhes de percepção e controle motor;
- A evolução dos estados de conhecimento de um agente pode ser descrita em forma de linguagem (lógica ou natural);
- A inteligência pode ser estudada independentemente dos processos de aprendizagem, desenvolvimento psicológico e mudança evolutiva;
- Existe uma arquitetura única na qual qualquer tipo de inteligência pode ser simulado.

Embora não haja um consenso na área sobre estas questões, é possível identificar grupos de pesquisadores que defendem algum subconjunto destas questões. Por exemplo: o enfoque lógico adota todas as suposições acima menos a última, sobre a qual permanece neutro. O enfoque conexionista é baseado na aprendizagem e nega a necessidade de conhecimento declarativo e da descrição dos estados de conhecimento sob a forma de linguagem, dividindo-se sobre a questão da independência em relação à percepção e ao controle motor. O enfoque moboticista (BROOKS,2002) apresenta a posição mais radical, não aceitando nenhuma das suposições e afirmando que a maioria dos comportamentos inteligentes pode ser produzida por um sistema de unidades de controle interligadas e cuidadosamente calibradas (BITTENCOURT,1998).

Uma vez que não há consenso na área, e não pretendendo derivar a discussão para o aspecto filosófico da questão *conhecimento*, adotou-se a seguinte definição apresentada em Costa (1993):

> *"**o conhecimento de uma máquina é o conjunto de estruturas operacionais e operatórias que organizam seu funcionamento**".*

Esta definição é confirmada por Bertolino (2003) quando afirma:

> *"**Conhecimento** é uma racionalidade expressando um determinado fenômeno ou conjunto de fenômenos, ou seja, é um conjunto de conceitos, idéias, leis, normas esclarecendo um determinado fenômeno ou conjunto de fenômenos"*

Conforme Costa (1993), "A. Newel (1982) apresenta a noção de nível de conhecimento, o qual difere dos que o precedem na hierarquia pelo fato de que ele não

dispõe de leis de conexão, isto é, Newel não reconhece estrutura no nível do conhecimento". Contrariando esta visão, Costa (1993) define que:

> *"(...)o nível do conhecimento é um nível essencialmente estrutural estabelecendo que os componentes, os sistemas e o meio do nível do conhecimento são todos constituídos de estruturas operacionais e operatórias, umas atuando sobre as outras, segundo leis de interconexão e leis de funcionamento essencialmente operatórias, isto é, lógico-algébricas. Não só na sua forma, no entanto, mas também no seu aspecto energético da ação, isto é, no aspecto dos valores envolvidos na ação, há estrutura: o próprio domínio dos valores admite estruturação".*

Complementa Costa (1993,pág.136) que as dimensões do conhecimento[57]

> *"Nos colocam o problema de saber onde, no espaço dos conhecimentos construtíveis, está o lugar que os sistemas artificiais podem alcançar. (...) em resumo, as máquinas atuais parecem amarradas ao nível sensório-motor do conhecimento, e especificamente às fases que antecedem o nível sensório-motor intencional. Quer dizer, toda a aquisição de que possam ser capazes se dá por uma constatação a posteriori do interesse que ele pode apresentar, e não por uma antecipação, criativa ou por experimentação exploratória, do novo conhecimento, posto que isto supõe a intencionalidade. Cremos que o reconhecimento destes fatos coloca a perspectiva realista em inteligência artificial dentro de um caminho menos ambicioso e aventuresco, porém certamente mais controlável em sua objetividade, que o caminho normalmente seguido pela artificialização, tal como ela tem sido feita até agora".*

Neste contexto Rich e Knight (1991) afirmam:

> *"(...) os últimos 30 anos da IA demonstraram que a inteligência requer mais do que a capacidade de raciocinar. Ela requer também uma grande quantidade de conhecimento acerca do mundo e também de mecanismos para a manipulação deste conhecimento a fim de criar soluções para novos problemas. Uma série de maneiras de representar o conhecimento (fatos) têm sido exploradas em programas de IA."*

E complementam,

> *"(...) antes de falarmos sobre elas (técnicas de representação de conhecimento), precisamos considerar o seguinte ponto, que tem a ver com todas as discussões sobre representação, especificamente o fato de estarmos lidando com dois tipos diferentes de entidades: Fatos e Representações" Rich e Knight (1991,pág.125).*

A partir desta abordagem, a definição de fatos e representações apresentada por Rich e Knight (1991,pág.126), é a seguinte:

- Fatos: verdades em algum mundo relevante. São as coisas que se deseja representar;
- Representações de fatos em algum formalismo escolhido. São as coisas que realmente podem ser manipuladas.

Segundo Rich e Knight (1991,pág.127), "é importante ter em mente que, geralmente, as funções de mapeamento disponíveis (de fatos para representações) não apresentam

[57] Costa (1993) denominou de "dimensões do conhecimento" as distinções que Piaget faz entre: (i) inato e adquirido;(ii) sensório-motor e representativo;(iii) uso dos conhecimentos e sua construção e,(iv) entre ações intencionais e não intencionais.

relacionamento um-para-um. Na realidade, elas normalmente não são nem funções, mas sim relações muitos-para-muitos".

Segundo Davidsson (1995) o conhecimento de um agente pode ser representado implicitamente (através de algoritmos de percepção e controle) ou explicitamente (é quando o conhecimento é separado do algoritmo que o utiliza). A distinção é semelhante, àquela que diferencia entre conhecimento procedimental e conhecimento declarativo. O conhecimento implícito é simples e eficiente, mas pouco flexível. O conhecimento explícito, ao contrário, é complexo e ineficiente, mas geral muito flexível. Assim sendo, um sistema de representação de conhecimento em um determinado domínio deve possuir as quatro propriedades (RICH e KNIGHT,1991,pág.130) que a seguir se descrevem:

- Adequação representacional: capacidade de representar todos os tipos de conhecimento necessários naquele domínio;
- Adequação inferencial: capacidade de manipular as estruturas representacionais de modo a derivar novas estruturas que correspondam a novos conhecimentos, inferidos a partir de conhecimentos antigos;
- Eficácia inferencial: capacidade de incorporar à estrutura de conhecimento informações adicionais que podem ser usadas para focalizar a atenção dos mecanismos de inferência nas direções mais promissoras;
- Eficácia aquisitiva: capacidade de adquirir novas informações facilmente.

Sobre esta última questão Rich e Knight (1991), afirmam que: "o caso mais simples de aquisição de conhecimento envolve a inserção direta, por meio de uma pessoa, de novos conhecimentos na base de dados. Idealmente, o próprio programa deveria ser capaz de controlar a aquisição de conhecimento".

O modelo atual de sistemas operacionais e, em última, instância todas as tecnologias de informação desenvolvidas sobre este modelo, está baseado no caso mais simples. Embora não sejam feitas referências explícitas ao termo "base de conhecimento", o que de fato ocorre é que, ao instalar um novo software em sua máquina, o usuário está, na verdade, transferindo conhecimento procedimental[58] (portanto de toda a equipe envolvida no processo de desenvolvimento que deu origem

[58] Segundo Rich e Knight(1991), a representação procedimental é aquela em que as informações de controle necessárias ao uso do conhecimento estão embutidas no próprio conhecimento. "Para usar uma representação procedimental, precisamos ampliá-la com um interpretador que siga as instruções fornecidas no conhecimento", afirmam Rich e Knight(1991,pág.199). Eles mesmos complementam que a representação declarativa é aquela em que o conhecimento é especificado, mas o uso que será feito dele não é fornecido. "Para usar uma representação declarativa, temos que ampliá-la com um programa que especifique o que deve ser feito com o conhecimento e como".

àquele software) para dentro dela sem que o sistema operacional tenha qualquer possibilidade de apreender alguma parte daquele conhecimento.

Para concluir as questões relacionadas ao conhecimento, cabe ainda caracterizar o seu papel nos sistemas baseados em conhecimento, quais sejam (RICH e KNIGHT,1991, pág.513):

- O conhecimento pode definir o espaço de busca e os critérios para determinar uma solução para um problema. Este tipo de conhecimento é caracterizado como conhecimento essencial;
- O conhecimento pode aumentar a eficiência de um procedimento de raciocínio informando a esse procedimento os melhores lugares onde procurar uma solução. Este tipo de conhecimento é chamado de conhecimento heurístico.

Wilkins (1984) afirma que, "uma questão importante relativa ao projeto de um modelo de representação para um sistema de planejamento é como representar os efeitos que uma ação tem sobre o estado do mundo. Isto significa que o problema do frame[59] (MCCARTHY e HAYES,1969; RICH e KNIGHT,1991, pág.151; HARNAD,1993) deve ser resolvido de forma eficiente".

A questão planejamento-representação envolve a representação do domínio, das metas e dos operadores. Os operadores constituem a representação interna do sistema sobre ações que podem ser executadas no domínio, ou, em um caso hierárquico, abstrações das ações que podem ser executadas no domínio. Um operador inclui uma descrição de como cada ação altera o estado do mundo. No entanto, deve-se caracterizar que há uma diferenciação entre os estados internos do mundo e as informações que podem ser expressas por ele. Por exemplo, McCarthy e Hayes (1969), afirmam: "nós nunca conhecemos uma pessoa o suficiente para listar seus estados internos. O tipo de informação que nós temos sobre ela necessita ser expresso em alguma outra forma".

Segundo Hofstadter (1980) apud Amaral e Castro (1993), as seguintes características são consideradas essenciais para um sistema exibir comportamento inteligente:

- Representar conhecimento declarativo e procedimental sobre o domínio da aplicação;
- Possuir autonomia na realização de suas tarefas;
- Apresentar comportamento flexível, isto é, serem capazes de se adaptar às condições do ambiente, em tempo de execução;

[59] Refere-se à questão levantada por McCarthy e Hayes (1969) relativamente à representação de fatos que mudam enquanto algum tipo de inferência está sendo executada. Segundo Rich e Knight (1991,pág.151), "em alguns domínios, a única parte difícil é representar todos os fatos enquanto em outros, determinar que fatos mudam não é uma tarefa trivial".

- Permitir modificações dinâmicas em seu comportamento, resultantes do conhecimento adquirido durante sua atuação; e,
- Interagir com o meio externo, fornecendo auxílio e relato de suas atividades.

6.2.1 A questão do aprendizado em sistemas inteligentes

> *"Uma máquina de aprender, genericamente falando, é qualquer dispositivo cujas ações são influenciadas por experiências" Nilsson(1965) apud Arkin(1999).*

Um aspecto do comportamento inteligente é que ele é claramente condicionado pelo conhecimento. Segundo Brachman e Levesque (2000) "para um grande número de atividades nós tomamos decisões sobre o que fazer baseados naquilo que nós conhecemos (ou acreditamos) sobre o mundo, atividade esta que ocorre sem esforço e inconscientemente". Um outro aspecto refere-se à questão da aprendizagem. Segundo Rich e Knight (1991,pág.513),

> *"Uma das críticas mais comuns à IA é que as máquinas só podem ser consideradas inteligentes quando forem capazes de aprender coisas novas e se adaptarem a novas situações, em vez de simplesmente fazer o que lhes foi mandado. Não pode haver muita dúvida de que uma importante característica das entidades inteligentes é a capacidade de adaptar-se a novos ambientes e de resolver novos problemas"*

Simon[60] (1983) apud Rich e Knight (1991), propõe que aprendizado denota:

> *"Mudanças adaptáveis no sistema, no sentido de que permitem que o sistema, da próxima vez, faça a mesma tarefa ou tarefas tiradas do mesmo grupo com mais eficiência e eficácia".*

Posto desta forma, aprendizagem abrange uma ampla escala de fenômenos que vão desde o refinamento de habilidades até à aquisição de conhecimento. O aspecto de aprendizado é visto como uma parte essencial em um sistema inteligente. Segundo Davidsson (1995), aprendizado de máquina é uma sub-área da Inteligência Artificial onde são estudadas técnicas automatizadas de aquisição de conhecimento (geralmente em domínios específicos).

---

[60] SIMON,H.A. Why Should Machines Learn? In Machine Learning, An Artificial Intelligence Approach,ed. Michalski, R.S. Carbonell, J.G. e Mitchell,T.M. (eds). Palo Alto, CA, Tioga Press, 1983.

Segundo Arkin (1999):

> *"O aprendizado produz alterações em um agente tornando-o capaz de executar mais efetivamente dentro do ambiente". Embora esta definição não seja satisfatória, ela fornece um meio para caracterizar o aprendizado através da definição de métricas de performance a partir das quais um agente pode ser avaliado antes, durante e depois que o aprendizado ocorreu."*

Os processos de adaptação e aprendizagem são aspectos relacionados. O processo de adaptação refere-se a um aprendizado do agente através do ajuste de parâmetros, de forma a torná-lo mais adequado ao ambiente. Neste sentido, há pelo menos três tipos de adaptação:

- **Comportamental**: o comportamento de um agente é ajustado relativamente a outro;
- **Evolucionária**: descendentes mudam com o tempo baseando-se no sucesso ou fracasso de seus ancestrais;
- **Sensorial**: o sistema torna-se mais ajustado (afinado) com o ambiente.

O processo de adaptação pode produzir dois efeitos característicos:

- ***Habituation:*** quando um estímulo é apresentado inúmeras vezes, pode causar um eventual decréscimo ou cessação de resposta comportamental. Este processo é útil para eliminar respostas comportamentais espúrias ou desnecessárias
- ***Sensitization*****:** é o movimento oposto. Neste caso há um acréscimo na probabilidade de uma resposta comportamental quando um estímulo é repetido freqüentemente.

Por outro lado, o processo de aprendizagem pode contribuir para melhorar a performance de um agente da seguinte forma:

- Introduzindo novos conhecimentos (fatos, comportamentos, regras) ao sistema;
- Generalizando conceitos para instâncias particulares que estão de alguma forma diferentes do padrão;
- Reorganizando as informações dentro do sistema de modo a torná-lo mais eficiente;
- Criando ou descobrindo novos conceitos;
- Criando explicações de como as coisas funcionam;
- Reutilizando experiências passadas.

As pesquisas em Inteligência Artificial têm destinado considerável esforço para determinar mecanismos pelos quais um sistema robótico pode aprender algumas destas questões. Há um grande número de técnicas, dentre as quais se podem destacar as seguintes:

- *Reinforcement learning*;
- Redes neurais;

- Aprendizado evolucionário;
- Aprendizagem por experiência (*memory-based learning* e *case-based learning*);
- Aprendizagem indutiva;
- Aprendizagem baseada em explicações;
- Aprendizagem baseada em várias estratégias.

O processo de aprendizagem geralmente está associado à questão da identificação de conhecimento procedimental e declarativo. Segundo Sun, Merrill e Peterson (1998), "acredita-se que ambas as formas de conhecimento sejam essenciais à criação de agentes situados em ambientes complexos". Ainda conforme Sun, Merrill e Peterson (1998), uma parte significativa do esforço de pesquisas relacionado a *skill learning* assume um enfoque *top-down*, ou seja, primeiro adquire-se uma grande parcela de conhecimento declarativo do domínio da aplicação e posteriormente, através da prática, transforma-se este conhecimento em uma forma procedural, o que conduz a uma representação eficiente. Contudo, este enfoque não é adequado em situações de ausência de conhecimento explícito do domínio da aplicação.

Segundo Sun, Merrill e Peterson (1998), as diversas pesquisas na área de *skill learning* indicam que *procedural skills* não são necessariamente acompanhados pela representação explícita de conhecimento declarativo. Caso o sistema não possua outra forma de aquisição de conhecimento que não o tradicional enfoque *top-down*, esta atividade pode ser comprometida.

6.3 A questão do tempo e suas implicações

A questão do tempo é um aspecto que ainda está sendo trabalhado tanto na física como na filosofia. Segundo George Musser,

> *"As leis da física incluem uma variável do tempo, mas ela não consegue capturar aspectos muito importantes de como esta dimensão é vista pelas pessoas em geral, especialmente a diferença entre passado e futuro. Quando os físicos tentam formular as leis mais fundamentais, o tempo parece evaporar-se completamente. Este problema conhecido como problema do tempo congelado surge quando os teóricos tentam transformar a Teoria da Relatividade Geral de Einstein numa teoria quântica usando o processo conhecido como quantização canônica. O processo funcionou de maneira brilhante quando foi aplicado à teoria do magnetismo, mas no caso da relatividade, produz uma equação – a equação de Wheeler-DeWitt - sem a variável tempo. Interpretada literalmente, a equação indica que o tempo do universo está congelado" Musser(2002).*

Complementa Feynman (1999),

> *"(...) não há nada na física para distinguir um momento do tempo do próximo. Os físicos dizem que o mundo é 'invariante sob traduções do tempo', significando que tomar a meia-noite ou o meio-dia como tempo zero nas medições não faz diferença alguma para a descrição dos fenômenos físicos. Os processos físicos não dependem de um zero absoluto. Revela-se que essa simetria sob a tradução do tempo implica diretamente uma das mais básicas, e também mais úteis, leis da física: a lei da conservação da energia. Esta lei afirma que você pode mudar a energia de lugar e alterar sua forma, mas não pode criá-la ou destruí-la. (..)A mecânica quântica mostra que a conservação da energia está intimamente relacionada à outra propriedade importante do mundo: as coisas não dependem do tempo absoluto! Se realizarmos uma experiência em dado momento e a repetirmos em um momento posterior, ela se comportará exatamente da mesma maneira. "*

Segundo Rich e Knight (1991, pág.615), "a noção mais básica de tempo é que ele é ocupado por eventos. Esses eventos ocorrem durante intervalos, espaços contínuos de tempo". Einstein (2001,pág.14), afirma que "toda a descrição do lugar onde ocorreu um evento ou onde se encontra um objeto se baseia em indicarmos o ponto de um corpo rígido (corpo de referência) com o qual o evento coincide".

Segundo Wanadoo (2001),

> *"Um fenômeno ocorre necessariamente como componente de estruturas polares, como pólos de fenômenos dipolares. Dentro de um dipolo, um dos componentes pode ser considerado como o referencial, ou a fundação do outro. Em outras palavras, o conhecimento de um fenômeno não pode ser dissociado daquele que é o referencial e qualquer proposição expressando este conhecimento deve refere-se explicita ou implicitamente a este referencial."*

Einstein (2001), abordando mais especificamente o conceito de tempo afirma que:

> *"Cada corpo de referência possui seu tempo próprio, portanto, uma especificação temporal só tem sentido quando se indica o corpo de referência ao qual esta indicação se refere. Antes da Teoria da Relatividade a física sempre admitia tacitamente que o significado das indicações do tempo era absoluto, isto é, que elas não dependiam do estado de movimento do corpo de referência"*

Do ponto de vista computacional, Costa (1993) define uma computação como sendo "uma realização temporal de uma construção em um domínio de objetos computacionais, ou seja, uma computação pode ser pensada como uma atribuição de conjuntos de operações aos instantes de tempo".

Segundo Scott(1970) apud Costa (1993), a produção de informação por uma máquina para seu ambiente é um processo monotônico no sentido do crescimento da informação produzida, pois uma vez a informação tendo sido produzida para o receptor, o emissor não tem mais como cancelar essa produção.

Costa (1993) complementa que,

> *"é certo que os acontecimentos físicos ocorrendo dentro da estrutura irreversível de seqüenciamento do tempo físico, realmente não podem ser revertidos uma vez realizados, posto que os instantes de tempo em que ocorreram não podem mais ser materialmente alcançados, após terem ocorrido. Desde o ponto de vista físico, portanto, o processo de interação é necessariamente monotonicamente crescente, no que diz respeito ao comprimento das seqüências que o compõem".*

Porém, como os objetos físicos são desprovidos, em si mesmos, de toda significação, os significados de um objeto surgem, então, somente pela inserção em um sistema de operações realizadas real ou potencialmente por um agente, e só existem para esse agente e são relativos à medida e à maneira como ele os insere nesse seu sistema de operações. Nestes termos, segundo Costa (1993) "não há porque recusar a idéia de um processo de interação que resulta na transmissão de uma informação negativa, a qual reduza a informação anteriormente existente no receptor da transmissão". Com estes argumentos Costa (1993) introduz a noção de informação relativa supondo a possibilidade da existência de operações para adicionar e subtrair informações e da operação de identidade de informação.

A partir destas considerações sobre a questão do tempo pautadas na Física em geral e na Teoria da Relatividade em particular, bem como no trabalho de Costa (1993), fica clara a necessidade de que o modelo de mundo utilizado incorpore este conceito de tempo ao contrário daquele utilizado no modelo atual de sistema operacional (em particular) e de computação (em geral), ou seja, o conceito de tempo absoluto demarcado por um componente de hardware denominado: relógio de tempo real.

Isto porque o modelo de tempo atual apresenta as seguintes características:

- Os eventos são associados ao valor absoluto da hora da máquina no momento em que são registrados (tempo explícito);
- A diferença entre passado, presente e futuro é determinada pela diferenciação aritmética entre dois valores absolutos de tempo explícito;
- Um histórico de eventos registrado desta forma demanda manipulação simbólica, uma vez que se restringe à comparação destes valores absolutos;
- Nem o sistema operacional, tampouco a aplicação que manipula o registro histórico, agregam conhecimento a cada operação, nem têm consciência do fato de estarem manipulando o tempo;
- O volume de dados históricos geralmente demanda a necessidade de armazenamento secundário, comprometendo a performance;

- A complexidade no tratamento deste volume de dados históricos remete à necessidade de técnicas especiais para "mineração de dados", uma vez que as mesmas não têm significado a menos que sejam processadas;
- As aplicações não possuem informação de referência de tempo, exceto aquela obtida através de amostragem explicita de tempo absoluto. Este valor de tempo absoluto, quando obtido, já representa o passado, visto que decorre algum tempo entre a aquisição do valor absoluto, conversões de tipo, operações aritméticas, até efetivamente tomar uma decisão sobre o encaminhamento de determinada situação;
- Via de regra, esta preocupação é latente em aplicações tempo-real, onde os *deadlines* são explícitos. Nas demais aplicações, o tempo não existe a não ser para registrar fatos do domínio da aplicação na forma de arquivos de *logs* ou registros em arquivos de dados;
- O procedimento de aquisição da hora atual assemelha-se àquele utilizado nas primeiras gerações de hardware onde o processador necessitava realizar *pooling* nos periféricos para descobrir quando um determinado periférico havia terminado de realizar uma operação de acesso solicitada.

Em função destas questões, faz-se necessária a introdução do conceito de tempo implícito (ou relógio de intervalo), ou seja, uma unidade de tempo que as aplicações possam perceber e referenciar-se imediatamente e que permita à aplicação inferir (justamente pelo fato de poder perceber o tempo) seu estágio atual de execução em relação a execuções anteriores. O conceito de tempo implícito tem sua contrapartida em seres humanos conforme descreve Wright (2002):

> *"Há uma espécie de cronômetro no cérebro, conhecido como relógio de intervalo, que identifica períodos de tempo que vão de segundos a horas. O relógio de intervalo ajuda você a calcular a velocidade com que tem de correr para pegar uma bola num jogo. A marcação dos intervalos ativa as faculdades cognitivas superiores do córtex cerebral, permitindo a uma pessoa que se aproxima de um semáforo calcular quanto tempo ele está em amarelo e decidir se pisa no freio o no acelerador".*

É justamente esta unidade de tempo implícito que deverá reger as decisões fundamentais do sistema relativamente à sua condição em atender adequadamente as solicitações do usuário em função de atributos de qualidade de serviço e adaptação ao ambiente.

A forma de representação do tempo implícito utiliza uma estrutura de dados denominada de sensor lógico, o qual será detalhado na seção 7.4.1.1pág.237. A forma como as aplicações deverão perceber a passagem do tempo será apresentada na seção 7.4.3pág.259.

6.4 O formalismo DEVS

Ziegler e Sarjoughian (2002) afirmam que o formalismo **DEVS** provê uma forma de especificar um objeto matemático denominado "sistema". Basicamente, um sistema possui uma base de tempo, entradas, estados e saídas e funções para determinar os próximos estados e saídas, uma vez fornecidas as entradas e estados atuais.

Dois aspectos importantes relacionados ao tempo e ao formalismo são destacados por Ziegler e Sarjoughian (2002) na seguinte afirmação:

> *"A estrutura de um modelo pode ser expressa em uma linguagem matemática denominada formalismo. O formalismo de eventos discretos concentra-se nas alterações do valor de variáveis e gera segmentos de tempo constantes. Assim, um evento é uma alteração no valor de uma variável que ocorre instantaneamente. Em essência, o formalismo DEVS define como gerar novos valores para variáveis e os tempos em que estes valores devem ser percebidos. Um aspecto importante do formalismo é que os intervalos de tempo entre as ocorrências dos eventos são variáveis contrastando com sistemas a tempo discreto onde o incremento de tempo geralmente é definido por uma constante"*

No formalismo **DEVS**, devem ser especificados (ZIEGLER e SARJOUGHIAN,2002):

- Os modelos básicos a partir dos quais modelos maiores podem ser construídos; e
- Como estes modelos são conectados de modo a formar uma hierarquia.

Segundo Ziegler e Sarjoughian (2002), nas linguagens de simulação tradicionais, deve-se conceber uma visão do modelo como possuindo portas de entrada e saída através das quais toda a interação com o ambiente é mediada. Em **DEVS**, são os eventos que determinam os valores que devem aparecer nas portas de saída. Mais especificamente, quando eventos externos (chegando de fora do modelo) são recebidos em suas portas de entrada, a descrição do modelo deve determinar como responder a eles. Além disso, a ocorrência de eventos internos, além de mudar o estado do modelo, manifesta-se como eventos nas portas de saída, de modo que os efeitos podem ser transmitidos para outros componentes.

Um modelo básico **DEVS** pode ser decomposto em sub-modelos atômicos, os quais são combinados através de modelos compostos (*coupled*).

Um modelo básico DEVS é descrito formalmente da seguinte forma:

$$\mathbf{M = \langle X, S, Y, \delta_{int}, \delta_{ext}, \lambda\ , ta \rangle}$$

onde:

- **X:** representa o conjunto das portas de entrada (ou valores de entrada) através das quais os eventos externos são recebidos;
- **S:** representa o conjunto de estados (variáveis e parâmetros). Duas variáveis estão geralmente presentes, *phase* e sigma, as quais, na ausência de eventos externos permitem ao sistema permanecer na *phase* atual por uma duração de tempo determinada por sigma;
- **Y:** representa o conjunto de portas de saída (ou valores de saída) através dos quais os eventos externos são sentidos;
- δ_{int}**:** é a função de transição interna. Ela especifica qual será o estado para o qual o controle será transferido após o esgotamento do prazo definido na função de avanço de tempo. Formalmente esta afirmação pode ser representada da seguinte forma:
  - $\mathbf{\delta_{int}: S \rightarrow S}$
- δ_{ext}: é a função de transição externa a qual especifica como o sistema muda de estado quando uma entrada é recebida – o efeito é colocar o sistema em uma nova *phase* e *sigma*, escalonando-o para uma próxima transição interna. Neste caso o próximo estado é calculado a partir do estado atual, da porta de entrada, do valor do evento externo e do tempo decorrido no estado atual. Isto pode ser colocado formalmente da seguinte forma:
  - $Q \times X^b \rightarrow S$ onde
    - $\mathbf{Q} = \{(s,\boldsymbol{e}) \mid s \in S, 0 \leq \boldsymbol{e} \leq ta(s)\}$ é o conjunto de estados;
    - $\boldsymbol{e}$ representa o tempo decorrido (*elapsed time*) desde a última transição
    - $\mathbf{X}^b$ denota a coleção de conjuntos (*bags*) sobre X, nos quais alguns elementos podem ocorrer mais que uma vez.
- λ**:** é a função de saída. Ela gera uma saída externa antes da transição interna efetivar-se. Formalmente esta função é definida como:
  - $\mathbf{\lambda: S \rightarrow Y^b}$**.**
- ta: é a função que controla o timing das transições internas. Quando a variável de estado sigma está presente, esta função retorna o valor de sigma. A especificação formal é:
  - $\mathbf{ta: S \rightarrow R^+_{0,\infty}}$

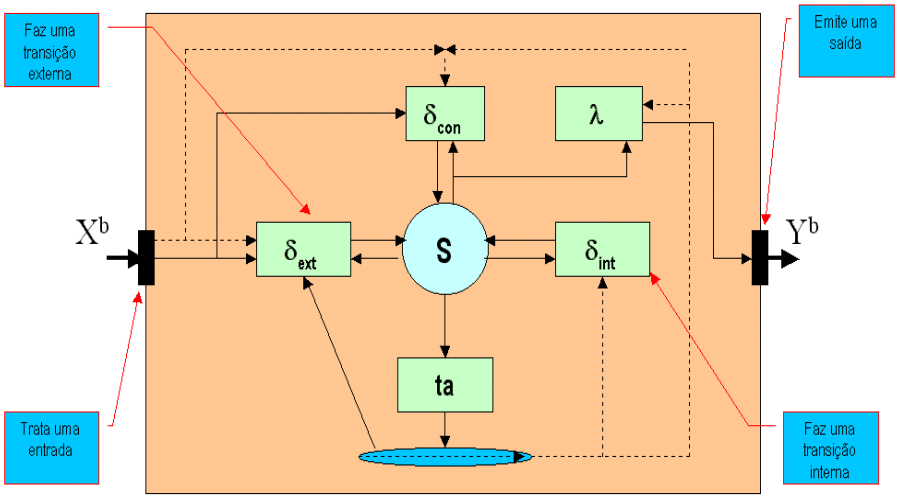


Figura 45 – Modelo DEVS.
FONTE: Adaptado de Ziegler e Sarjoughian (2002).

A interpretação destes elementos, ilustrada na Figura 45, é a seguinte:

- A qualquer tempo o sistema está em algum estado s.
- Se não houver eventos externos, o sistema permanecerá no estado s por ta(s) tempo. Observe-se que ta(s) pode ser um numero real, mas também pode receber valores 0 e ∞. No primeiro caso, diz-se que o modelo está em um estado transitório. No segundo, o sistema permanecerá no estado s para sempre, até que algum evento externo o interrompa. Neste caso o modelo está num estado passivo.
- Quando o tempo de "descanso" (*resting*) termina, ou seja, quando o tempo decorrido (*elapsed time*) e = ta(s), o sistema apresenta as saídas nas portas de saída através da função $\lambda(s)$ e altera o estado através da função δ_{int} (as saídas são disponibilizadas imediatamente antes da transição).
- Se um evento externo $x \in X^b$ ocorrer antes da expiração do tempo, ou seja, quando o sistema está em um estado (s,e) com $\boldsymbol{e} \leq ta(s)$, o sistema executa a transição de estado através da função $\delta_{ext}(s,e,x)$.
- Isto implica que a função de transição interna estabelece o novo estado do sistema quando não ocorreram eventos desde a última transição, ao passo que a função de transição externa estabelece o novo estado do sistema quando um evento externo ocorreu.

Para Barros (2002) como um evento externo $x \in \mathbf{X}^b$ representa o conjunto de elementos de **X**, significa que:

- um ou mais elementos podem ocorrer nas portas de entrada de um modelo;
- ao mesmo tempo o modelo deve ser estendido com uma função extra denominada função de transição de confluência ($\boldsymbol{\delta_{con}}$), a qual é aplicada quando uma entrada é recebida ao mesmo tempo em que uma transição interna está para ocorrer (a situação *default* é aplicar a função de transição interna antes de aplicar a função de transição externa).

O modelo formal que suporta esta facilidade é apresentado a seguir:

$$\mathbf{M = \langle X, S, Y, \delta_{int}, \delta_{ext}, \delta_{con}, \lambda , ta \rangle ,}$$

onde

- $\delta_{con} : \mathbf{Q} \times \mathbf{X}^b \rightarrow S$ e
  - $\mathbf{Q} = \{(s,\boldsymbol{e}) \mid s \in S, 0 \leq \boldsymbol{e} \leq ta(s) \}$ é o conjunto de estados;
  - $\boldsymbol{e}$ representa o tempo decorrido (*elapsed time*) desde a última transição
  - $\mathbf{X}^b$ denota a coleção de conjuntos (*bags*) sobre X, nos quais alguns elementos podem ocorrer mais que uma vez.

Assim é possível estabelecer comportamentos quando as duas condições a seguir ocorrem simultaneamente:

- A chegada de um evento externo; e
- O tempo decorrido do modelo atinge o seu valor máximo.

O aspecto mais importante na adoção do formalismo **DEVS** é que os itens básicos de dados produzidos por um sistema ou modelo no contexto **DEVS** são segmentos de tempo. Estes segmentos de tempo constituem-se em mapeamentos de intervalos de tempo (definidos sobre uma base de tempo especificada) a valores de uma ou mais variáveis. Estas variáveis podem ser ou observadas ou medidas (calculadas). Este modelo, associado à utilização das demais tecnologias descritas neste capítulo, permite a construção de um modelo operacional que é ao mesmo tempo verificável e consistente, conforme será apresentado no próximo capítulo.

6.5 Modelo de mundo – aspectos conceituais

> *"Um programa de computador capaz de agir de forma inteligente no mundo deve ter uma representação do mundo em termos de quais das suas entradas devem ser interpretadas. Projetar tal programa requer soluções de compromisso relativamente ao que significa conhecimento e como ele é obtido. Assim, alguns dos maiores problemas da filosofia são pertinentes em inteligência artificial."*[61] *McCarthy e Hayes (1969).*

Sobre a questão da inteligência e do conhecimento, o texto a seguir apresenta uma compilação do trabalho de McCarthy e Hayes (1969) pela importância histórica e porque suas considerações são pertinentes ao contexto do presente trabalho.

[61] A computer program capable of acting intelligently in the world must have a general representation of the world in terms of which its inputs are interpreted. Designing such a program requires commitments about what knowledge is and how it is obtained. Thus, some of the major traditional problems of philosophy arise in artificial intelligence.

McCarthy e Hayes (1969) estabelecem que:

> *"(...) trabalhos em inteligência artificial, especialmente inteligência geral (*general intelligence*), deverão ser melhorados a partir de uma idéia clara sobre o que é inteligência. Uma forma é apresentar uma definição puramente comportamental ou caixa-preta. Neste caso, nós temos que dizer que uma máquina é inteligente se ela resolve certas classes de problemas que requerem inteligência em humanos, ou sobrevive em um ambiente que demanda intelectualidade. Esta definição parece vaga; talvez ela possa ser apresentada de alguma forma mais precisa sem desconexão dos termos comportamentais, mas nós não tentaremos fazer isto. Ao invés, nós vamos utilizar em nossa definição certas estruturas aparentes para a introspecção tais como o conhecimento de fatos".*

E complementam os autores que,

> *"(...) nós vamos dizer que uma entidade é inteligente se ela tem um modelo adequado de mundo (incluindo o mundo intelectual da matemática, entendimento de suas metas e outros processos mentais), se ela é esperta (*clever*) o suficiente para responder a uma grande variedade de questões baseada neste modelo e pode executar tais tarefas no mundo externo a medida em que suas metas indicam e suas possibilidades físicas permitem" McCarthy e Hayes (1969).*

De acordo com esta definição, inteligência tem duas partes, quais sejam: epistemologia e heurística. A parte epistemológica é a representação do mundo de tal forma que a solução de problemas ocorra em função dos fatos expressos na representação. A parte heurística é o mecanismo que, baseado nas informações, resolve os problemas e decide o que fazer. Segundo McCarthy e Hayes (1969), "a maior parte das pesquisas em inteligência artificial até então estão destinadas à parte heurística do problema".

A partir desta abordagem de inteligência, as seguintes questões devem ser consideradas relativamente à parte epistemológica da inteligência artificial:

- Qual tipo de representação geral do mundo permitirá a incorporação de observações específicas e novas leis científicas à medida que elas são descobertas?
- Além da representação do mundo físico, que outros tipos de entidades devem ser representadas?
- Como as observações serão utilizadas para obter conhecimento sobre o mundo, e como os vários tipos de conhecimento serão obtidos?
- Em particular, quais os tipos de conhecimento sobre o próprio estado do sistema deverão ser considerados?
- Em que tipo de notação interna o conhecimento do sistema é expresso?

6.5.1.1 Representações do Mundo

O primeiro passo no projeto de um sistema de raciocínio é a decisão sobre qual estrutura de mundo será adotada e como as informações sobre o mundo e suas leis de mudanças serão representadas na máquina.

Segundo McCarthy e Hayes (1969),

> *"Do ponto de vista da Inteligência Artificial, nós podemos definir três tipos de adequação para representações do mundo. Uma representação denominada metafisicamente adequada, é utilizada quando a representação não contradiz os fatos e aspectos da realidade que interessam a determinado domínio de aplicação. Este tipo de representação é útil para a construção de teorias gerais onde é possível derivar conseqüências observáveis a partir destas teorias como um passo seguinte no processo. Uma representação é considerada epistemologicamente adequada quando ela pode ser utilizada de forma prática, para expressar os fatos que alguém possuir sobre algum aspecto do mundo.Uma representação é denominada heuristicamente adequada se o processo de raciocínio utilizado na solução do problema pode ser expresso em alguma linguagem."*

Segundo Davidsson (1995), um modelo de mundo é basicamente uma representação interna que um agente faz do mundo exterior. Entretanto, deve-se considerar a seguinte distinção:

- **Modelos ambientais (*environment models*):** são aqueles modelos que descrevem somente o estado atual das cercanias (*surrounding*). Geralmente são apresentados como descrições tridimensionais de objetos físicos no ambiente.
- **Modelos de mundo (*world models*):** são aqueles modelos que incluem conhecimento mais geral sobre possíveis estados e modos de alcançar tais estados. Geralmente este tipo de modelo inclui conhecimento mais estável e geral sobre objetos, suas propriedade, relacionamentos, eventos e processos relacionados.

A realidade é que os agentes autônomos possuem informações incompletas e imprecisas sobre o mundo e não dispõem de tempo suficiente para planejar suas ações. Assim sendo, a solução de compromisso deve ponderar entre (DAVIDSSON ,1995):

- **Enfoque deliberativo**: assumindo-se que o mundo geralmente não muda muito rapidamente, tenta-se executar mais de um plano a cada passo enquanto o processo é monitorado. Quando o monitoramento detecta que algo errado ocorreu, dispara-se um passo de revisão do planejamento anterior, parando-se a execução do plano atual;
- **Enfoque reativo:** tenta-se antecipar cada possível situação categorizando-se cada uma delas. Para cada categoria são estabelecidos os passos a serem executados e estes passos são armazenados sendo disparados somente quando determinada condição ocorrer.

Ambas as situações apresentam pontos positivos e negativos. O enfoque deliberativo tenta obter coerência penalizando a reatividade, uma vez que se baseia nos planos construídos pelo módulo de planejamento. Além disso, a performance é

comprometida porque há uma intercalação entre planejamento e reação, além da dependência de um modelo de mundo mais preciso. Já o enfoque reativo consegue eficiência e reatividade comprometendo a coerência. Estes sistemas geralmente são sujeitos a *dithering* e comportamento inadequado em função de *perceptual aliasing* e por apresentarem estratégias fracas para a solução dos problemas. Além disso, em função da natureza fixa dos planos, geralmente apresentam pouca flexibilidade quando expostos a um ambiente muito dinâmico. Para o programado,r expressar o comportamento desejado em termos de regras de situação-ação é uma tarefa complexa (DAVIDSSON ,1995).

6.5.1.2 Importância do modelo de mundo

O conhecimento *a priori* do mundo pode contribuir significativamente durante o processamento da percepção. Em algumas situações, como no contexto de comportamento motor, o uso apropriado desta forma de conhecimento apresenta os seguintes aspectos positivos:

- Prover expectativas para processos de percepção (o que procurar);
- Mecanismos de foco de atenção (onde procurar);
- Correto seqüenciamento de algoritmos de percepção (quando procurar);
- Configuração inicial de mecanismos para a fusão (junção) de múltiplas fontes de informação em uma representação global.

Além disso, informações sobre o mundo, independentemente de serem representadas ou derivadas, podem auxiliar a restringir a complexidade inerente à interpretação perceptual. Para caracterizar esta situação dois aspectos são analisados a seguir, relativos às expectativas e às suposições sobre continuidade temporal e consistência das informações.

O primeiro refere-se às expectativas (*expectations*) a respeito do mundo, as quais permitem que equipamentos com limitados recursos computacionais possam executar efetivamente em tempo real. As expectativas podem prestar a determinados processos de percepção pelo menos duas informações adicionais:

- Onde procurar por um evento perceptual em particular; e
- Como tal evento se parece.

Ainda quanto às expectativas, é preciso mencionar a questão do foco de atenção, o qual provê informações sobre onde o agente deveria consumir seus recursos perceptuais, orientando o processo de busca em uma imagem; ou determinando para que direção apontar os sensores. Em essência, é uma estratégia de redução do espaço de busca. O problema pode ser resumido como estratégias para podar o espaço de busca dentro do qual um objeto perceptual pode aparecer.

O segundo aspecto refere-se às suposições sobre continuidade temporal e consistência das informações. Isto pode permitir ao sistema restringir a possível localização de um objeto de percepção de um ponto no tempo para o próximo, permitindo que restrições significativas mais simples sejam aplicadas ao processamento sensorial.

Finalmente, uma questão a ser considerada é o que McCarthy e Hayes (1969) denominaram de situação:

> *"Uma situação **s** é um completo estado do universo em algum instante do tempo. Nós denotamos por **Sit** o conjunto de todas as situações. Como o universo é muito grande para uma descrição completa, nós nunca conseguiremos completamente descrever uma situação. No máximo nós teremos fatos sobre situações. Estes fatos serão utilizados para deduzir futuros fatos sobre aquela situação, sobre situações futuras e sobre situações que as pessoas podem trazer sobre aquela situação. Isto requer que nós consideremos não somente situações que efetivamente ocorrem, mas, também situações hipotéticas".*

6.6 O Modelo de Árvores Paralelas Funcionais de Decisão

> *"As pesquisas em agentes situados tem produzido uma série de importantes idéias e princípios nos últimos anos. Pesquisadores de vários campos tem apresentado resultados que vem alterando a forma como nós pensamos sobre comportamento inteligente. Situatedness ou embodiement tornou-se um dos mais importantes princípios para arquiteturas inteligentes e é uma característica que define um agente situado. Utilizando-se o conceito de situatedness como ponto de partida, emergiram várias novas concepções sobre planejamento, comunicação, symbol grounding, percepção e representação e execução de planos. Insitu representa uma tentativa de incorporar algumas das mais importantes destas idéias em um projeto unificado" Schaad (1998, pág59).*

Schaad (1998), em sua tese de doutorado, apresenta um enfoque para a programação de agentes situados reativos baseado em árvores paralelas funcionais de decisão (*Parallel Functional Decision Trees*) cuja estrutura deverá ser utilizada como base para o modelo proposto.

Segundo Schaad (1998), cada agente situado é confrontado com os seguintes problemas fundamentais:

- Está embutido (embedded) em um ambiente que muda dinamicamente, o qual interfere em seus planos e impõe restrições em tempo real na forma de execução de suas computações e processos físicos;
- Não possui conhecimento perfeito sobre o estado do ambiente e sobre seu próprio estado. Geralmente, o conhecimento é incompleto e impreciso;
- Possui limitados recursos computacionais e é incapaz de recalcular planos ótimos a cada unidade de tempo, mesmo que informações completas estivessem disponíveis.

Estes problemas são refletidos nos seguintes requisitos necessários para agentes situados:

- Solução de compromisso entre coerência e reatividade (*coherence-reactivity trade-off*[62]): a área de engenharia está sempre envolvida em encontrar soluções para um conjunto de metas de projeto que são conflitantes. No caso de agentes situados, a coerência temporal das ações e a capacidade de reagir a situações em constante mudança caracterizam estes conflitos. Portanto, uma arquitetura de agentes situados deve prover suporte para adequação e balanceamento destas metas conflitantes;
- Agir para obter informações: o processo de aquisição de informações em agentes situados sempre possui um custo associado. Muitas decisões que o agente precisa tomar dependem de informações que ele não possui. Uma arquitetura para agentes situados deve prover mecanismos especificamente projetados para suportar a aquisição de informações através de ações efetuadas pelo próprio agente;
- Uso adequado de instruções: a sensibilidade no emprego de instruções em uma determinada situação implica verificar se faz sentido aplicá-las (*guarded interpretation*) e de que forma (*context-dependent interpretation*).

As contribuições do trabalho de Schaad que determinaram sua utilização no modelo proposto foram as seguintes:

- Contribuições à questão de balanceamento entre coerência e reatividade: a base do trabalho de Schaad é que uma atividade situada deve estar baseada no principio da improvisação, o qual estabelece que um agente deve considerar a alteração dinâmica do ambiente para constantemente reavaliar as condições que levaram a uma particular decisão no uso de algum plano. Este aspecto, por si, só atende aos requisitos para um agente ser considerado reativo. **Insitu** introduz um mecanismo para capturar e reutilizar planos a partir de tarefas recorrentes através dos chamados *pattern blocks*. O enfoque de *patterns* é adotado em Insitu para permitir a construção de coerência temporal em arquiteturas reativas. Para implementar os planos reativos e os *patterns blocks*, Schaad introduziu uma nova forma de representação de planos denominada de *parallel functional decision trees* (árvores de decisão funcionais paralelas). Apesar de apresentar uma estrutura simples, ainda que extensível, elas permitem a construção de estruturas modulares e em níveis. A partir desta estrutura, é possível

[62] Como consistentemente trabalhar em direção de um conjunto de metas sem sacrificar a capacidade de responder a alterações ambientais.

identificar-se a correspondência entre a estrutura estática e o comportamento dinâmico resultante.

- Contribuições à questão de percepção ativa: as soluções adotadas no projeto sofreram influências de pesquisas na área de visão intermediária, particularmente na construção de modelos para cobrir atenção e processamento visual. Os operadores perceptuais em Insitu podem incluir movimentos físicos em qualquer grau de liberdade do agente. Insitu permite que o uso de representação reativa leve em conta contingências e oportunidades que podem surgir durante o uso dos planos. Além disso, o modelo permite que sejam criadas referências simbólicas entre um objeto no mundo real e um plano reativo sendo adotado. Isto é obtido através de um registro de plano. Registro é um processo associado ao comportamento de, ativamente, localizar um objeto no ambiente e criar uma referencia índice-função para aquele objeto. Estes planos simbólicos ficam *grounded* (baseados) à experiência de interação do agente com o ambiente. Em função da semântica das *parallel functional decision trees*, seu uso é transparente tanto para representar planos reativos, como para representar percepção ativa. Além disso, os mecanismos para suportar coerência temporal são disponíveis a quaisquer tipos de planos, em qualquer nível de abstração.
- Contribuições ao uso de instruções situadas: a maior dificuldade no emprego de instruções situadas está justamente em relacioná-las ao contexto de execução. Insitu demonstra que a integração de execução baseada em improvisação, rotinas perceptuais, registro de planos e representação índice-função, podem produzir instruções altamente sensíveis ao contexto. Apesar do modelo suportar instruções *on-line* e *off-line* (planos), não há diferença na sua forma de representação interna. Assim, as instruções podem ser utilizadas da mesma forma em um ambiente sensível ao contexto associado a ações do agente em atividades de percepção ativa. O modelo utiliza uma forma simples para agregar e separar instruções *on-line* e *off-line* em tempo de execução e interpretá-las de uma forma sensível ao contexto.

6.6.1 Aspectos conceituais

A arquitetura do modelo de Schaad pode ser descrita através de cinco níveis de abstração:

- Planos como comunicação;
- Realidade compartilhada;
- Improvisação;
- Padrões;
- Interpretação como ação.

6.6.1.1 Planos como comunicação

Segundo o autor, tradicionalmente um plano é considerado como uma coleção de primitivas executáveis ou macros que são decompostos em primitivas. Estas primitivas são executadas sem considerar a situação atual, ou seja, sua semântica é livre de contexto.

Agre e Chapman (1990) apud Schaad (1998,pág.61) denominaram esta característica como uma visão de planos como programas (*plan-as-program*). Nesta visão, um plano é como um programa de computador. Um módulo executor independente de domínio simplesmente executa o plano com todos os seus comandos. Além de monitorar condições e predicados que estão contidos no plano, o módulo executor não faz qualquer esforço no sentido analisá-lo semanticamente utilizando sensores de entrada. Se uma contingência aparece, o módulo executor descarta o plano e retorna o controle para o planejador. Este tipo de módulo executor não é condicionado (*grounded*) ao domínio situado porque não leva em conta a situação atual quando interpreta as primitivas do plano. Assim, a responsabilidade de garantir o comportamento apropriado é inteiramente do construtor de planos (programador), o qual não pode utilizar qualquer tipo de "senso-comum" ou inteligência da parte do módulo executor (AGRE e CHAPMAN,1990 apud SCHAAD,1998 ,pág.61).

Agre e Chapman (1990) apud Schaad (1998,pág.61) apresentam uma visão de planos como forma de comunicação (*plan-as-communication*) onde os planos são considerados como um recurso que conduz as ações de um agente ao invés de prescrevê-las. Os autores identificaram três formas que diferenciam a visão de planos-como-programas e planos-como-comunicação:

- Uso de planos: na visão de planos-como-comunicação, a utilização de um plano envolve um processo contínuo de interpretação. Um esforço considerável pode estar envolvido em descobrir qual atividade um plano sugere em uma situação concreta. Este trabalho interpretativo pode, ele mesmo, envolver uma série de ações de sentir e agir (*sensing and acting*). Dessa forma, a visão de plano-como-comunicação sugere que um plano seja um recurso que o agente pode usar (juntamente com os demais recursos disponíveis) quando seleciona suas ações. Sendo um recurso que requer interpretação situada, um agente pode escolher ou ignorar ações sugeridas, se elas não parecem fazer sentido em determinada situação particular. Ou ainda, pode extrair a intenção implícita em um plano e tentar obtê-la de uma forma diferente daquela sugerida por ele.
- Representação de planos: enquanto uma primitiva em um plano-como-programa possui uma semântica simples e sempre produz a mesma seqüência de ações, independentemente da situação, o significado dos elementos de um plano-como-comunicação depende de um grande número de aspectos contextuais. Um agente situado deverá analisar o ambiente, a tarefa corrente e o contexto lingüístico para determinar as ações concretas sugeridas pelo plano. Um plano-como-comunicação pode ser mais simples de gerar que um correspondente plano-como-programa, tendo em vista que o construtor de planos pode lançar mão de conceitos já assimilados através da interação do agente com o ambiente.
- Natureza da atividade: a visão de planos-como-comunicação emergiu a partir de uma nova teoria de atividades situadas (SUCHMAN,1987 apud SCHAAD,1998,pág.61). Nesta visão, a dinâmica autônoma do mundo não é um problema a ser solucionado, mas um recurso a ser considerado. Conseqüentemente, um plano-como-comunicação não tenta rigidamente

> controlar a atividade executada em um ambiente situado. Ao invés disto, um plano forma a base para uma atuação de improvisação dinâmica durante a interação com o ambiente.

Complementa Schaad que a visão de plano-como-comunicação oferece uma perspectiva diferente para o problema de geração e uso de planos, porque enfatiza a capacidade de o agente engajar-se em um processo contínuo de interação com o ambiente e de interpretar os planos passados para ele. Esta característica viabiliza a construção de planos que permitem ao sistema atender à característica de adaptação apresentada na definição de sistema operacional baseado em conhecimento.

6.6.1.2 Realidade compartilhada

A finalidade de passar um plano ou uma instrução a um agente é permitir que ele trabalhe cooperativamente. Contudo, para ser capaz de operar cooperativamente, ele deve compartilhar o mesmo entendimento sobre a situação corrente, a tarefa e seus objetivos. Deve haver uma concordância (*agreement*) sobre qual é a situação e como ela pode ser alterada, ou seja, os agentes envolvidos devem compartilhar uma visão comum da realidade. Contudo, Schaad relaciona os seguintes aspectos sobre a identificação dos recursos necessários para caracterizar esta realidade compartilhada:

- Estrutura compartilhada: a principal fonte de entendimento compartilhado da realidade é a estrutura compartilhada dos agentes. Quanto mais similares os agentes, tanto mais fácil de encontrar uma base comum para a comunicação e mais facilmente suas visões coincidirão em um dado ponto do tempo;
- Interação com um ambiente compartilhado: uma segunda fonte de realidade compartilhada é a interação de agentes com um ambiente comum. Sentindo e atuando em um mesmo ambiente, dois agentes serão capazes de adquirir visões comuns deste ambiente ao longo do tempo;
- Comunicação compartilhada: o terceiro modo de compartilhar as realidades de dois agentes é através da comunicação. Conversando um com o outro, eles podem trocar informações que farão com que suas realidades internas venham a convergir.

Segundo Agre e Chapman (1990) apud Schaad (1998, pág.63), "a comunicação é parte do trabalho de manter uma realidade comum". Schaad complementa que, através da comunicação, os agentes sincronizam suas visões sobre o ambiente, tarefas e metas. A partir desta observação, a visão de planos-como-comunicação implica nas seguintes observações:

- A comunicação facilita o compartilhamento da realidade: comunicação não implica somente em transmitir conteúdo, mas em criar uma visão compartilhada da realidade;

- Realidade compartilhada facilita a comunicação: a comunicação sempre se refere a um contexto compartilhado implícita ou explicitamente. Ele somente pode ser entendido se ambas as partes comunicantes compartilham uma visão da realidade que amarra sua semântica;
- Comunicação como um recurso: a comunicação explícita é única, entre muitas fontes de informação sobre a realidade, para um agente em particular. É responsabilidade do agente interpretar e avaliar esta informação e decidir como utilizá-la.

Esta característica viabiliza o conceito de identidade de máquina apresentado na definição de sistema operacional baseado em conhecimento, na medida em que cada máquina passa a ser concebida como um agente.

6.6.1.3 Improvisação

Este princípio é parcialmente implícito a partir dos dois anteriores. Schaad emprega este termo de forma similar àquela utilizada em Chapman (CHAPMAN,1991,pág.29, apud SCHAAD,1998,pág.64) e designa: "o processo de tomar decisões repetidamente sobre qual ação executar no momento seguinte".

No processo de improvisação um agente usa a situação atual, planos, metas e outros recursos, como meios para decidir a próxima ação no próximo passo (*time-step*). Esta visão contrasta com os enfoques em que um agente segue um pré-determinado curso de ação. Enquanto neste último modelo a tarefa de "pensar" é feita antes da ação, no modelo Insitu as atividades de "pensar" e "agir" ocorrem em paralelo no processo de improvisação.

Segundo Schaad (1998,pág.66), a vantagem fundamental na estratégia de improvisação é que os mais recentes fatos são levados em conta no momento de decidir a próxima ação a ser executada. Esta característica é crucial em ambientes dinâmicos, onde o planejamento antecipado torna-se obsoleto muito rapidamente. Contudo, há também algumas dificuldades que devem ser consideradas:

- Coerência: as atividades de coerência temporal requerem atenção redobrada ao adotarem-se estratégias de improvisação;
- Capacidade computacional: decidir o que fazer a cada passo (*time*-step) consome muitos recursos computacionais. As arquiteturas reativas resolvem este problema adotando planos universais pré-concebidos, ou seja, possuem planos que prescrevem uma ação para cada possível situação.

A característica de improvisação vai viabilizar a implementação do conceito de comportamento inteligente presente na definição de sistema operacional baseado em conhecimento.

6.6.1.4 Interpretação como ação

Na descrição sobre planos-como-comunicação e realidade compartilhada, observa-se que a interpretação como ação pode demandar um considerável esforço para interpretar um plano de comunicação e que uma estrutura comum de agentes comunicantes e os estados percebidos do ambiente executam um papel importante nesta interpretação.

Segundo Schaad "nós podemos resumir isto dizendo que a intenção comunicada é uma função do contexto e do conteúdo comunicado. Contudo, geralmente este contexto não é totalmente instanciado – por exemplo, um agente pode reconhecer que outro agente está referindo a alguma parte do contexto em sua comunicação que ainda não é conhecido por ele. Nestes casos, o agente interpretando poderia ter que tomar ações para aprender esta informação".

De uma forma geral pode-se classificar as ações que um agente poderia executar para aprender sobre uma mensagem em duas categorias (SCHAAD,1998):

- Registro de objetos: refere-se à capacidade de instanciação de referências a objetos externos aos quais os planos se referem;
- Meta de obtenção de conhecimento: refere-se a atividades que são destinadas a responder questões especificas sobre o ambiente – por exemplo, um agente poderia receber a seguinte instrução: se a porta está aberta, envie um alarme de segurança. Após registrar a "porta", ele teria que determinar se a porta está aberta ou não. Então deveria executar uma ação: meta de saber se a porta está aberta. Esta ação refere-se à meta geral de aquisição de informações através da avaliação de funções que se referem ao contexto ambiental em um ou mais aspectos. A parte condicional do fragmento acima requer a avaliação da função: *is_open (the_door*). Sendo o valor de retorno um valor lógico. Em **Insitu**, uma função representando uma meta de conhecimento pode retornar qualquer tipo de dado. Esta função explicitamente relaciona o ambiente através de uma referência *deictic* ao seu ambiente (*the_door*).

Esta característica vai contribuir para a implementação do caráter de aprendizagem que um sistema operacional baseado em conhecimento deve possuir para apresentar um comportamento inteligente.

6.6.2 Visão arquitetônica

Insitu é uma arquitetura de software para desenvolvimento de agentes situados. A princípio, há quatro formas a partir das quais os planos podem ser criados e modificados nesta arquitetura:

- Através de programação *off-line,* utilizando ferramentas como compiladores, mecanismos de aprendizado *off-line* e planejadores (*planners*) *off-line*;
- Através de instrução on-line, por pessoas ou outros agentes;
- Através de planejamento on-line (modificação projetiva de planos);
- Através de aprendizado automático (modificação retrospectiva de planos).

O projeto **Insitu** investiga o uso dos aspectos básicos de instrução e uso de planos. As questões relativas a planejamento e aprendizado, as quais dependem do conceito de uso de planos, não foram consideradas no protótipo construído.

A Figura 46 apresenta o diagrama de fluxo de dados de nível 0, caracterizando que a tarefa é controlar um agente situado. Isto requer o interfaceamento com sensores e atuadores. Insitu processa dois tipos de entrada de planos: planos gerados off-line por um programador e instruções on-line fornecidas a partir de outro agente (o instrutor), o qual é um operador humano no protótipo implementado.

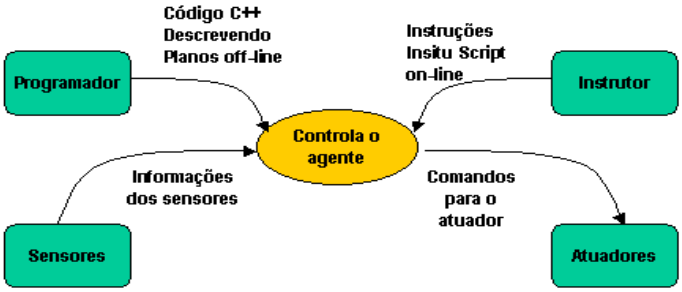


Figura 46 – DFD de nível 0 do projeto InSitu.
FONTE: Adaptado de Schaad (1998).

A Figura 47 apresenta a decomposição do nível 1. Ela mostra a estrutura do Insitu em termos de fluxo de dados. Há dois tipos de planos que guiam as ações do agente:

- Planos *default*: são planos gerados off-line pelo programador. Estes planos são compilados e armazenados no *cache* de planos do sistema antes dele iniciar sua execução;
- Planos adicionados: o instrutor tem a possibilidade de adicionar planos ao sistema, em tempo de execução, e assim expandir os planos *default* e influenciar as ações tomadas pelo módulo executor.

Os planos podem ser removidos do *cache* de planos ativos e devolvidos ao *cache* de planos inativos automaticamente, o que implica que somente os planos ativos influenciam o comportamento do agente.

O princípio de projeto de improvisação é a base do modelo de execução em **InSitu**, o que requer que a situação atual seja completamente reavaliada continuamente para permitir a apropriação das ações geradas. Isto implica que **InSitu** é um sistema reativo. Assim, todos os componentes não reativos são construídos sobre este substrato reativo. O modelo de execução em si não contém qualquer aspecto não reativo.

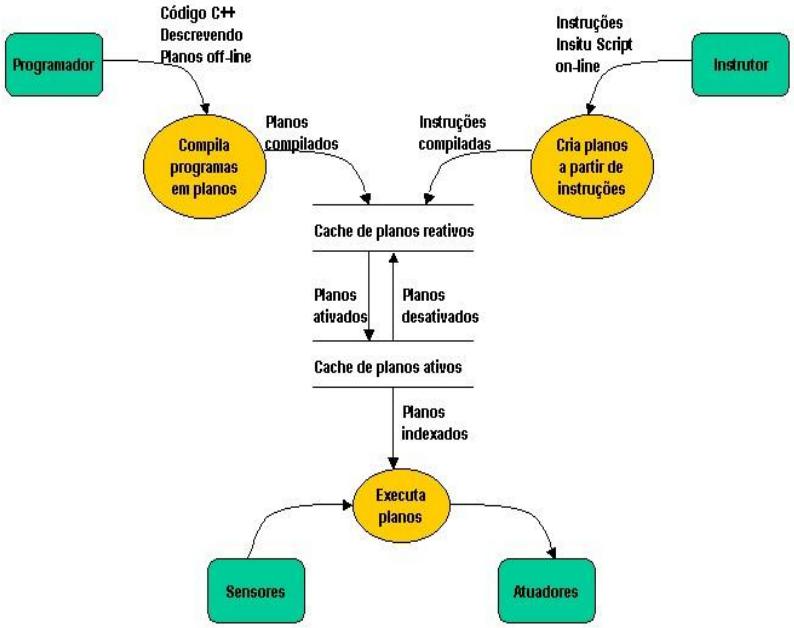


Figura 47 – DFD de nível 1 do Projeto InSitu.
FONTE: Adaptado de Schaad (1998).

Idealmente o ciclo sentir-pensar-agir em um agente com capacidade de improvisação possui duas propriedades:

- Atraso zero (*zero delay*): alteração nos sensores são refletidas nas ações geradas sem atrasos;
- Amostragem continua: a recepção das entradas ocorre continuamente, o que implica numa freqüência de amostragem infinita.

No projeto de um sistema de controle existem dois aspectos importantes a ser considerados:

- A freqüência de atualização: podem-se utilizar períodos de amostragem fixos ou variáveis. Um novo ciclo de atualização pode ser iniciado tão logo o ciclo anterior termine, ou ele pode ser iniciado em pontos fixos no tempo. Dependendo da situação, o atraso de um ciclo de atualização pode variar. Sistemas baseados em períodos de amostragem variável apresentam uma taxa mais alta de amostragem que sistemas baseados em períodos fixos. Contudo, geralmente não é possível predizer o momento em que uma amostragem irá ocorrer. Sistemas baseados em períodos fixos requerem que o programador garanta que nenhum ciclo de atualização demandará mais que o tempo máximo entre duas amostragens. Isto limita a flexibilidade quando combinando processos de longa duração (ex: planejamento ou processamento visual) com processos de curta duração (ex: *servo loops*).
- O processo de arbitragem: Os processos computacionais em sistemas de controle podem ser organizados de duas maneiras: (i) o processo de arbitragem consiste de vários processos paralelos, cada um dos quais produz uma proposta para quais ações produzir através dos sensores (*effectors*). Como estas propostas estão em competição para acesso aos atuadores, elas podem produzir conflitos que devem ser resolvidos por um mecanismo de arbitragem. Este mecanismo seleciona para execução uma das propostas conflitantes geralmente baseado em alguma heurística de arbitragem. Esta estratégia é denominada: arbitragem *ex-post* e aparece em sistemas baseados em comportamento; (ii) o processo de arbitragem é antecipado. Para isto, primeiramente o estado do mundo é classificado em um conjunto de situações (conjuntos de estados do mundo) e então estas situações são mapeadas em suas correspondentes saídas para os atuadores.

Ambos os enfoques de arbitragem apresentam vantagens e desvantagens. O mecanismo *ex-post* torna mais fácil acomodar processos computacionais com possibilidades de variação muito grandes (ex: *servo loops* rápidos e processamento visual lento). Além disso, eles são mais adequados para implementações em hardware paralelo, como redes lógicas ou neuronais. Por outro lado, a coordenação destes processos independentes não é trivial. Garantir coerência temporal nestes casos é uma questão complexa. Como várias propostas são computadas, mas somente uma será utilizada, a arbitragem *ex-post* é ineficiente em hardware seqüencial, como em microprocessadores. Segundo Schaad (1998).

Uma implementação da arquitetura de *Subsumption* (BROOKS, 2002) gasta muitos ciclos de seu processador calculando resultados que serão descartados. Finalmente, estas arquiteturas tendem a dificultar o entendimento tanto em termos de sua estrutura estática como de seu comportamento dinâmico. Isto se deve ao fato de que o relacionamento entre a estrutura estática e o comportamento dinâmico do sistema é altamente não linear e freqüentemente contra-intuitivo, devido à grande quantidade de processos paralelos envolvidos.

A arbitragem *ex-ante,* por outro lado, é eficiente em hardware seqüencial, dado que somente aquelas propostas necessárias são efetivamente computadas. Além disso, a

implementação de um sistema de *run-time* em hardware seqüencial é mais simples porque não precisa tratar com um grande número de processos concorrentes. O mapeamento de situação-ação facilita a compreensão e a associação entre a estrutura estática e o comportamento dinâmico emergente (ex: num processo de depuração, a simples consulta ao plano do agente para a situação considerada explica porque o sistema comporta-se desta ou daquela forma).

A solução adotada em **InSitu** envolve períodos fixos de amostragem e arbitragem *ex-ante*. Em cada período de amostragem do sistema um plano é selecionado e executado até o seu término. Assim, a cada *time tick* um plano é executado. Esta característica é denominada de modelo de execução *stepped* (passo-a-passo). Além das vantagens citadas anteriormente, este modelo de execução é adequado à implementação na forma de árvores de decisão, que é o tipo de representação de planos adotada em **InSitu**.

Este modelo de execução passo-a-passo assemelha-se aos modelos de multitarefa cooperativa adotados em linguagens, como Modula-2, ou sistemas operacionais, como **Windows 3.1** e **MacOS 8**.

Cabe destacar que o modelo **Insitu** requer que o programador do sistema tenha disciplina e evite escrever planos que não terminam dentro de um montante de tempo alocado a cada período de amostragem. Assim, algoritmos de longa duração devem se quebrados em unidades de código menores, as quais terminam dentro do período de amostragem, tendo em vista que o escalonador não pode interromper (preempção) a execução de uma tarefa. Esta restrição deve ser superada no modelo proposto a partir da introdução do conceito de relógio de intervalo. Isto porque um plano, constatando-se "atrasado" em relação ao tempo das execuções anteriores, pode adequar-se dinamicamente às variações ambientais que fazem com que este eventual atraso esteja ocorrendo.

6.6.2.1 Mapeamento entre estados do mundo e ações

Schaad faz uma revisão das diversas formas de representação de planos em sistemas reativos "desde as redes de maquinas de estados finitos de Brooks até os programas Teleo-Reativos de Nilsson". Ele conclui que, por trás das várias estratégias, está a questão do mapeamento entre estados do mundo e ações.

Segundo Schaad (1998), uma arquitetura reativa ideal deveria ser capaz de mapear, em ações, o estado atual do mundo real, incluindo os estados do próprio agente. Na prática, contudo, um agente pode mapear em ações somente seus estados internos e as entradas perceptuais do mundo, uma vez que esta é toda a informação disponível sobre o mundo real. Portanto, as diferentes estratégias de representação deste mapeamento variam em função dos diferentes modos de incorporar expansões em suas funções básicas ou pelo modo que tais funções representam o mapeamento.

Há uma importante observação a ser feita sobre este mapeamento: o número de estados do mundo geralmente é infinito, enquanto o número de atuadores é relativamente pequeno. Assim, como cada estado do mundo deve ser mapeado para algum atuador, muitos estados do mundo vão ser mapeados para as mesmas saídas dos atuadores. A estes conjuntos de estados do mundo, Schaad denominou de situações. Assim, todos os estados em uma situação mapeiam para a mesma saída de um atuador, embora nem todos os estados do mundo que mapeiam para uma particular saída pertençam à mesma situação.

A partir destas informações, pode-se afirmar que uma arquitetura reativa baseada em arbitragem *ex-ante* consiste em um mapeamento de estados do mundo para saídas (atuadores) em dois estágios:

- Estágio um: mapeamento do estado de mundo atual para uma situação;
- Estágio dois: mapeamento de uma situação para uma saída.

Pode-se reduzir o número de situações necessárias e, assim, o número de mapeamentos, reconhecendo-se o fato de que a maior parte dos mapeamentos do estágio dois apresentam uma estrutura muito similar.

Schaad destaca que estes dois estágios estão presentes em qualquer arquitetura reativa e não representam um projeto particular. Contudo, em arquiteturas com arbitragem *ex-post*, este princípio não é claramente visível, principalmente tendo em vista que a ordem do mapeamento é inversa: primeiro calcula-se o número de possíveis funções de saída dos atuadores e, então, seleciona-se uma delas através da classificação do atual estado do mundo.

6.6.3 Representação do mapeamento entre estados do mundo e ações

A adoção de árvores de decisão para representação do mapeamento entre estados do mundo e ações, deve-se ao fato de que a estrutura é adequada para representar funções de decisões, classificação ou mapeamento, de forma simples e eficiente.

Como qualquer estrutura da espécie, as árvores de decisão são constituídas dos seguintes elementos:

- Nodos não-terminais: são associados a funções de decisão, as quais computam um índice de decisão. Este índice é utilizado para selecionar um dos conectores associados com o nodo para efetuar a transição. Cada índice calculado deve corresponder a um dos conectores do nodo. Para tanto, as funções utilizam os estados internos ou entradas (a partir dos sensores) para calcular este índice;
- Conectores: conectam nodos não-terminais a outros nodos não-terminais ou nodos terminais. Eles determinam o conjunto de transições permitidas nodo a nodo. Somente o nodo raiz (*root*) não possui conectores de entrada;
- Nodos terminais: são associados com ações dos atuadores e representam as decisões tomadas. Cada decisão corresponde a uma saída particular para um atuador.

O significado de um nodo em uma árvore de decisão pode ser visto a partir das seguintes perspectivas:

- Visão da situação: a seleção de um nodo em uma árvore indica que o estado do mundo atualmente está em uma situação particular. Esta situação é inequivocamente definida pelo caminho, desde o nodo raiz até o nodo selecionado. Assim, cada nodo é associado a uma situação;
- Visão da ação: as sub-árvores associadas com um nodo descrevem o que deve ser feito naquela situação particular. Assim, cada nodo é também associado às ações chamadas para esta situação.

A seguir são relacionadas as diferentes partes de uma árvore de decisão aos conceitos clássicos das entidades conhecidas como planos:

- Operadores: são comandos associados com um nodo terminal correspondendo aos operadores do plano que pretendem transformar o estado do mundo;
- Macro operadores: cada nodo não-terminal é associado com um determinado número de sub-árvores. Estas sub-árvores podem ser construídas a partir de macro operadores de representação de planos. Cada um destes macro-operadores recursivamente consiste de operadores (nodos terminais) ou outros macro operadores (sub-árvores);
- Pré-condições: uma situação associada com um nodo é parte das pré-condições que devem ser verdadeiras para que o correspondente (macro) operador seja executado;
- Sub-metas: um nodo pode também ser visto como representando uma sub-meta de um agente. Por exemplo, considere um robô móvel que bateu em um obstáculo. Neste caso, um

nodo será selecionado cuja situação associada expressa o fato de que o robô está atualmente parado contra um obstáculo. A sub-árvore associada com aquele nodo conterá código para desviar do obstáculo. Assim se um nodo é selecionado, a meta de suas sub-árvores é tirá-lo desta situação.

Deste modo, um nodo associa uma situação com a próxima sub-meta a atingir, com os operadores a serem executados para esse fim e com as pré-condições necessárias para que tais operadores sejam executados.

A adoção de árvores de decisão como forma de representação do mapeamento entre estados do mundo e ações deveu-se aos seguintes fatores:

- Simplicidade: estrutura de fácil implementação em qualquer linguagem de programação;
- Eficiência: as arvores de decisão, além de executarem eficientemente, permitem o mapeamento do estágio um de forma econômica (em termos de espaço);
- Modelo de execução adotado: a estrutura mostrou-se adequada à implementação do modelo de execução passo-a-passo (*stepped*), do modelo de arbitragem *ex-ante* e da estratégia de improvisação discutidos anteriormente;
- Transparência: a estrutura facilita a depuração porque:
  - O mapeamento do estagio um é facilitado tendo em vista que a categorização ocorre como uma seqüência sistemática de decisões simples, à medida que se navega em direção às folhas da árvore;
  - O mapeamento do estágio dois é fácil de compreender porque toda a informação para entendê-lo está localizada em algum nodo terminal da árvore.
- Níveis (*layering*): o conceito de níveis é importante para representar coerência temporal como persistência, seqüências de ações e reutilização de código.Um nível corresponde à parte da árvore de decisões que executa um estágio no processo de decomposição da meta;
- Predição (*predictability*): a estrutura torna possível a representação gráfica do caminho a ser seguido pela ação, permitindo a predição de características de tempo real (ex: o pior caso de tempo de execução é simplesmente o máximo dos piores casos de todos os caminhos de um nodo raiz a qualquer nodo terminal). O tempo de execução de cada um dos caminhos, por sua vez, é calculado simplesmente somando os piores tempos de execução de todos os nodos naquele caminho.
- Universalidade: em arvores de decisão, dado estado do mundo é associado a uma situação. Neste sentido, as árvores de decisão são inerentemente universais e uma representação adequada para planos universais.

6.6.4 Arquitetura do protótipo InSitu

O modelo de execução da proposta de Schaad (1998) é relativamente simples. Ele consiste de um laço (*loop*) que capta informações dos sensores e transfere-as para *buffers* de entrada. Uma função de mapeamento transfere as entradas para *buffers* de saída e

funções de entrada/saída que transferem as informações dos *buffers* de saída para os atuadores. Todo processamento ocorre na função de mapeamento.

O componente central da arquitetura **InSitu** é sua lista de planos universais ativos (A*ctive Universal Plans* – AUP), descritos através do modelo de árvores de decisão apresentado anteriormente.

A abstração mais importante no modelo denomina-se *Steppable* – objetos que encapsulam algoritmos de mapeamento, os quais são executados de modo passo a passo (*stepwise fashion*) para produzir comportamento seqüencial reativo através do curso de vários passos. Os *Steppables* foram projetados para atender aos requisitos do problema de coerência-reatividade.

Os *steppables* são implementados como classes derivadas da classe abstrata *Steppable*, a qual define a interface de 4 funções membro:

- Data * step (void);
- Void abort (void);
- Void reset (void);
- Fuzz *isDone (void);

O sistema possui uma entidade denominada escalonador, responsável pelo controle da execução do modelo passo-a-passo. A cada *time tick,* o escalonador seleciona um objeto para execução, a partir do *cache* de planos ativos, e ativa o método ***Step***. Este método avança o algoritmo de um passo e produz os efeitos em outros *steppables* conectados, retornando o resultado ao chamador. O método ***step*** implementa a maior parte da funcionalidade do comportamento reativo, além de incluir o algoritmo para navegação na árvore de decisões. Em outras palavras, o algoritmo de escalonamento não é localizado no escalonador, mas implementado de uma forma distribuída como uma seqüência recursiva de chamadas à função ***step***. Cada steppable autonomamente decide qual ramo da sub-árvore deve seguir e, então, o método ***step*** daquele nodo selecionado é ativado. Em função desta arquitetura distribuída, tudo o que o escalonador tem que fazer é chamar o método ***step*** do nodo raiz a cada período de amostragem.

Um aspecto importante destacado por Schaad é que a possibilidade de haver múltiplos AUP em um sistema pode caracterizar uma brecha no modelo de arbitragem *ex-ante*. Isto porque um AUP com maior prioridade, executado posteriormente na

seqüência, poderia sobrescrever valores adquiridos em passos anteriores naquele mesmo *time tick* e comprometer a integridade do modelo. Embora conhecido este problema, Schaad afirma que esta é uma decisão de projeto e que o programador deve encarregar-se de tratar adequadamente a questão.

As funções ***reset*** e ***abort*** estabelecem o progresso do algoritmo para o início ou para o fim do algoritmo.

Um algoritmo passo a passo (*stepped algorithm*) pode realizar qualquer numero de chamadas à função ***step*** para a sua conclusão. Assim, os *steppables* oferecem suporte para combinar aspectos de execução reativa e coerente através da união do modelo de execução com a abstração seqüencial de algoritmos.

O exemplo a seguir descreve a implementação de uma estrutura condicional com controle de *timeout*:

```
Steppable *a = new TimedIF (Steppable *condicao,
                            Steppable *then,
                            Steppable *else,
                            Int timeOut);
```

Esta estrutura encapsula o seguinte comportamento: a cada passo, ou seja, toda a vez que o ambiente de *runtime* chama o passo ***step*** da estrutura *TimedIF*, ocorrem as seguintes ações:

- A estrutura **TimedIF** chama o passo ***step*** da estrutura *steppable* **condição** e retorna um objeto de dados do tipo ***Fuzz*** (um tipo de valor lógico difuso).
  - Se o valor de retorno é avaliado para ***True***, a função ***step*** do steppable ***then*** é chamada;
  - Caso contrário, a função ***step*** do steppable ***else*** é chamada;
  - Após a execução com sucesso da função ***step*** do steppable **condição**, é disparado um contador regressivo a partir do parâmetro *timeOut* milissegundos. Este contador executa em paralelo com a execução do passo ***then*** ou do passo ***else*** da estrutura.

Este exemplo demonstra o uso de *steppables* produzindo resultados através da ativação de outros *steppables*. Esta característica é implementada através de ***Action Steppables***. Contudo, há também os ***Data Steppables***, que são utilizados para armazenar dados. Nestes o passo ***step*** somente retorna o endereço da própria estrutura, tendo em vista manter a integridade do modelo.

A lista a seguir sumariza os *steppables* disponíveis em **InSitu** para ações independentes de tarefas:

- Desvio e coerência temporal: Sequence, Prog, Loop, StickyIf, TimedIf, StateTGraph, PulseTrain, Ramp, Debounce, DoInterval, DoTimes, If, Switch;
- Controle de execução: Every, EveryTime, Exit, Once, RemoveUSP, Parallel;
- Depuração: Debug, Noop, TestAction, TestTester;
- Controle de Seqüência: ResetStep, AbortStep, IsDoneStep, TimeoutReset, TimeoutAbort, ConditionalReset,ConditionalAbort;
- Aritméticas: Abs, Aços, Add, Asin, Atan, Cos, Div, Mul, Sin, Sub, Tan;
- Comparações: About, Eq, Ge, Gt, Le, Lt;
- Lógicas: And, Or, Not;
- Variáveis: Setq;
- Sistema de run-time: Parser, Dock, Scheduler, Opfile, Opprompt e Opremote;
- Rotinas perceptuais: Register.

A lista a seguir sumariza os steppables disponíveis em InSitu para ações específicas:

- Saída para atuadores: Say, SetCam, SetMode, SetVel, SetISize, SetPos, SetSound, AchieveVel;
- Consulta a sensores: Here, AttitudeQ, BlockedQ, BumpedQ, CompassQ, LightACQ, LightQ, ModeQ, PanQ, PirQ, PosxQ, PosyQ, RadarQ, RotvQ, SonarQ, SoundQ, TiltQ, TransvQ;
- Navegação: AvoidForbidden, FolloWall, IterateRoute, LocalMap, PanlARoute, PlanARandomRoute, SonarAvoid, FullCircleQ
- Posicionamento: EdgeDetected, EstimateWall, UpdadeDRPwall, WaypointToFloorpoint;
- Controle de movimento: Moveby, Turnby, Goto, Turnto;
- Operadores de percepção: ExtendedRay, GoAlongRay;
- Processamento visual: MonitorCam, SaccadeTo;
- Saída gráfica: DrawAll, ShowVisual;

Um aspecto importante destacado por Schaad (1998), é que a estratégia de utilizar objetos para encapsular algoritmos permite que múltiplas cópias do algoritmo sejam executadas com diferentes estados locais, ou seja, um tipo de reentrância de código.

A essência do modelo pode ser representada por uma árvore de decisões. Cada *steppable* individualmente pode ser considerado como uma sub-árvore de decisão. Porém, os *steppables* diferem de árvores de decisão puras na medida em que receberam alguns melhoramentos. Por exemplo: as funções de decisão do modelo que permitem a

seleção de um ramo da árvore a ser executado em detrimento de outro, foram modificadas. Em árvores de decisão binárias, esta função é determinada por uma expressão lógica como: (porta = aberta). Dependendo da saída da avaliação (*True* ou *False*) um dos ramos da sub-árvore é selecionado. O problema deste enfoque em agentes situados é que, em gera,l eles não sabem de antemão se a função de decisão (porta = aberta) será avaliada para *True* ou *False*. Assim, é necessário que o agente interaja com o ambiente a fim de obter as informações necessárias para responder a esta questão – percepção ativa. Uma vez que os steppables controlam a interação com o ambiente, a função de decisão propriamente pode ser implementada como uma sub-árvore de decisão, consistindo de outros *steppables*.

A partir destas considerações pode-se destacar as principais características das *Parallel Functional Decision Trees* descritas no trabalho de Schaad (1998):

- Suporte a extensões funcionais: as árvores de decisão são capazes de retornar dados para os métodos que chamam a função ***step*** para servir como função de decisão – compatível com o modelo DEVS;
- Suporte a extensões paralelas: as árvores de decisão são capazes de seguir vários caminhos a cada intervalo de tempo (*time tick*). Por exemplo, o caminho de uma função de decisão e o caminho do desvio selecionado pelo resultado da função de decisão deve ser percorrido no mesmo passo. Isto significa que, mesmo que eles sejam executados seqüencialmente, conceitualmente eles são executados em paralelo – adequado ao modelo de relógio de intervalo;
- Suporte a classificação de situações: quando as árvores de decisão são usadas para controlar um agente, sua finalidade é classificar o estado do mundo como ele se encontra e associar ações programadas para esta classe de estados do mundo. Schaad (1998) chama esta classe de estados do mundo de situações. Com o uso de árvores de decisão, esta classificação é recursiva, isto é, cada ramo na arvore adiciona precisão à classificação e o número de estados na situação resultante diminui a cada ramo. Cada nodo na árvore corresponde a uma situação de mundo em particular. Associando-se uma sub-árvore de decisão a um nodo em particular, confere-se ao sistema uma estratégia para executar naquela situação em particular. Assim, quando um ramo deixa de ser utilizado, outro mais apropriado pode ser adotado – adequado para aquisição de conhecimento procedural;
- Suporte a árvores *default*: esta estratégia permite a definição de uma classificação padrão e, portanto, de um comportamento padrão. Ma medida em que ocorrem as interações do agente com o mundo, este comportamento pode ser incrementalmente alterado através da adição ou subtração de sub-árvores, tendo em vista adequar-se a situações diferentes. Esta possibilidade de alteração dinâmica das árvores é uma das mais importantes características do modelo – adequado para a implementação de adaptação;
- Suporte à percepção ativa: como as sub-árvores são capazes de retornar dados para os métodos chamadores (extensões funcionais), é possível às funções de decisão adquirir informações através das ações e utilizá-las para selecionar sub-ramos a serrem executados. Este tipo de percepção é geralmente referido na literatura como buscar atingir metas, ou seja,

a meta do agente é adquirir conhecimento – adequado para a implementação de aprendizagem;

- Suporte à marcação: estratégia que permite o registro de símbolos como referência a objetos no mundo real. Schaad (1998) cita como exemplo um objeto que armazena coordenadas de um cachorro obtidas através de uma câmera instalada no robô. O modelo suporta tanto a marcação como o acompanhamento dos movimentos do objeto marcado. Assim, os marcadores passam a fazer parte das metas a serem perseguidas – adequado para implementação de aprendizagem.

6.7 Considerações finais

Este capítulo apresentou os fundamentos conceituais que dão sustentação ao modelo proposto, quais sejam:

- O conceito de conhecimento e comportamento inteligente segundo a visão naturalista da inteligência artificial proposta por Costa (1993), bem como a necessidade de aprendizagem como forma de produção de comportamento inteligente;
- A questão do tempo implícito a partir do conceito de relógio de intervalo;
- O formalismo DEVS como forma de implementar não somente o conceito de relógio de intervalo, como um modelo de execução que permita as aplicações perceber a passagem do tempo durante a execução;
- O modelo de árvores paralelas funcionais de decisão, as quais, associadas ao formalismo DEVS como modelo de *run-time*, possibilitam acrescentar determinismo ao sistema de escalonamento distribuído dos *steppables*.

A partir destes fundamentos, o próximo capítulo apresenta o modelo proposto.

7 SISTEMA OPERACIONAL BASEADO EM CONHECIMENTO - SOBC

7.1 Introdução

O modelo proposto neste capítulo busca uma melhor solução para os problemas apresentados nos capítulos anteriores, a partir da agregação de conhecimentos desenvolvidos em outras áreas de pesquisa, particularmente em Robótica, Inteligência Artificial, Engenharia do Conhecimento, Computação Gráfica, Física e Filosofia, não como módulos agregados (ou *kernel implants*), mas como parte da especificação funcional de tais sistemas. O capítulo inicia com o estabelecimento dos conceitos que fundamentam a proposta e, posteriormente, é apresentada a definição de Sistema Operacional Baseado em Conhecimento. A partir desta definição são identificados os requisitos funcionais que este modelo de sistema operacional deve atender e, finalmente, é apresentado o detalhamento do modelo a partir da nova base conceitual estabelecida.

Conforme afirmado na seção 1.1.2 (Problema de Pesquisa$_{Pág.28}$), o tripé conceitual caracterizado pela virtualização do operador, pela virtualização do hardware e pelo conceito de programa, caracterizam os fundamentos básicos do atual modelo de construção de sistema operacional, em particular, e de soluções em software em geral. Conforme apresentado nos capítulos precedentes, problemas identificados já há quase 3 décadas persistem através das várias gerações de tecnologias desenvolvidas ao longo deste período de tempo. Portanto, a iniciativa da construção de um novo modelo deve atacar esta base.

Como discutido na seção 2.4.12 (Estruturas Baseadas em Conhecimento$_{Pág79}$), a idéia de sistema operacional baseado em conhecimento não é recente. No entanto, invariavelmente as propostas foram concebidas a partir do modelo atual, o qual foi resumido na Figura 44, sem atacar diretamente a questão do tripé conceitual que o caracteriza. Em conseqüência, as abordagens acabaram desviando-se dos objetivos iniciais e derivando os esforços no sentido de aplicar técnicas da Inteligência Artificial sobre algum tipo de núcleo (adaptado ou não) que suporta multitarefa (multiprocessado ou não, distribuído ou não), contribuindo, sem dúvida, para facilitar a utilização da máquina (através de interfaces em linguagem natural, planejamento de escalonamento de

tarefas e aquisição de informações sobre o ambiente), mas efetivamente distanciando-se do que deveria ser um Sistema Operacional Baseado em Conhecimento.

Piaget (1977,pág.19) apud Costa (1993,pág.131) afirma que: "nenhum trabalho científico pode avançar metodicamente sem que seu objeto de estudo esteja definido, o que se impõe, sem dúvida, para delimitar o domínio de que ele se ocupa".

Assim, como os trabalhos anteriores em sistemas operacionais baseados em conhecimento não se apoiaram em uma definição adequada, em verdade seu domínio de interesse, até aqui, não ficou perfeitamente delimitado. Por conseguinte, o emprego de alguma ou várias técnicas de inteligência artificial na área de sistemas operacionais, segundo o paradigma atual, passou a ser erroneamente caracterizada como um sistema operacional baseado em conhecimento.

Definir o que é um sistema operacional baseado em conhecimento, portanto, é um passo preliminar necessário para esclarecer o objeto do estudo em questão. Assim sendo, e tendo esta concepção bem clara, faz-se imprescindível caracterizar o que se entende por um sistema operacional baseado em conhecimento.

7.2 Definição

Um Sistema Operacional Baseado em Conhecimento é:

Um sistema de software incorporado, localizado, adaptável e autônomo, baseado em conhecimento, que possui identidade e conduta inteligente quando executado.

A partir deste enunciado, faz-se necessária uma análise detalhada sobre o real significado do conceito apresentado. Para tanto, cada trecho da definição é discutido a seguir:

- **Um sistema**: é um sistema[63] cujos componentes se organizam em funções sistêmicas realizadas por subsistemas cujo conjunto suporta funções gerais, em especial a organização, a adaptação e a regulação que se realizam de forma endógena;

[63] A palavra sistema tem muitas conotações, mas no contexto do presente trabalho será considerada a definição apresentada em Chiavenato (1993), ou seja: "o conjunto de elementos interdependentes e interagentes; um grupo de unidades combinadas que formam um todo organizado e cujo o resultado é maior do que o resultado que as unidades poderiam ter se funcionassem independentemente." Chiavenato (1993) identifica os seguintes componentes dos sistemas em geral: (i) Entrada (insumo, impulso ou input) – é a força de arranque ou de partida do sistema que fornece o material ou energia para a operação do sistema; (ii) Saída (produto ou resultado ou output) é a finalidade para a qual se reuniram elementos e relações do sistema; (iii) Processamento (processador ou transformador) é o fenômeno que

- **De software**: é um programa de computador que, como qualquer outro, pode apresentar os mesmos problemas de qualidade e produtividade tão amplamente discutidos pela área de Engenharia de Software. Portanto, um projeto de sistema operacional está sujeito às mesmas considerações relativas às atividades de criação, entrega e manutenção de sistemas de software;
- **Incorporado:** o modelo é um construto teórico que determina um jogo de propriedades operatórias para as quais se estabelecem, no hardware real, correspondentes reais tanto da estrutura material[64] como da estrutura operacional[65] que provêm suporte operacional[66] à máquina;
- **Localizado:** o sistema tem que possuir meios de localizar-se no ambiente onde está inserida a máquina. Indica, também, que o sistema não funciona isoladamente, mas sim está inserido em um ambiente e funciona em interação com ele;
- **Adaptável:** a adaptação do sistema ao ambiente é a construção, por assimilação ou acomodação[67] correlativas, de uma estrutura funcional englobando o hardware e a parte do ambiente com a qual o sistema interage, de um modo que possibilita a continuidade do funcionamento do sistema no ambiente cada vez que este muda[68];
- **Autônomo:** o termo autonomia indica que as formas de funcionamento do sistema devem ser determinadas por ele mesmo, e não por um agente externo qualquer. O sistema deve ser capaz de interagir independente e efetivamente com o ambiente através dos sensores e atuadores lógicos e físicos, de forma a executar as tarefas que lhe são submetidas;
- **Baseado em conhecimento:** considera-se neste contexto a definição apresentada em Costa(1993), isto é, um sistema baseado em um conjunto de estruturas operacionais e operatórias que organizam o seu funcionamento, atuando umas sobre as outras segundo leis de interconexão e de funcionamento essencialmente operatórias, isto é, lógico-algébricas;
- **Que possui identidade:** a identidade é uma decorrência imediata da capacidade de adaptação e incorporação e das condições da estrutura e do suporte operacional do sistema em determinado momento, fazendo com que os eventos sejam percebidos de forma absolutamente individualizada;

---

produz mudanças, ou seja, é o mecanismo de conversão das entradas em saídas. O processador caracteriza a ação dos sistemas e define-se pela totalidade dos elementos (tanto elementos como relações) empenhados na produção de um resultado. E é geralmente representado por uma caixa negra; (iv) Retroação (retroalimentação ou alimentação de retorno ou *feedback*) é a função de sistema que visa comparar a saída com um critério ou padrão previamente estabelecido tendo por objetivo o controle e,(v) Ambiente – é o meio que envolve externamente o sistema. Também pode ser o contexto em que se está inserido.

[64] Conforme Costa (1993) estrutura material é "a coleção de componentes materiais da máquina e suas interconexões".

[65] Conforme Costa (1993) estrutura operacional é "a estrutura de processos dentro da qual se dá o funcionamento do hardware conforme ele é determinado pela estrutura material da máquina em função da interação com um dado ambiente".

[66] Conforme Costa (1993) suporte operacional é "a correspondência entre funções da estrutura funcional e componentes materiais da estrutura material, e entre ligações funcionais da estrutura funcional e processos da estrutura operacional, de modo que, em cada momento, para cada função exista exatamente um componente material, ou conjunto de componentes materiais, que a realiza, e para cada ligação funcional existe um processo na estrutura operacional que a produza".

[67] Conforme Costa (1993), assimilação é o processo pelo qual o sistema incorpora estruturas do ambiente à sua estrutura funcional enquanto acomodação é o processo pelo qual a estrutura funcional é modificada para possibilitar a assimilação de estruturas apresentadas pelo ambiente, quando alguma resistência é encontrada.

[68] Conforme Costa(1993), é o estado de desenvolvimento do sistema em que toda a estrutura que ele encontra no meio pode ser assimilada, e toda a acomodação necessária pode ser realizada.

- **E conduta inteligente:** é determinada pelo conjunto e pelo entrelaçamento de ações que o sistema realiza com o ambiente, isto é, pela participação do sistema nas trocas funcionais que ele estabelece com o ambiente[69] e nas interações funcionais [70] entre o ambiente e o sistema;
- **Quando executado:** em se tratando de uma máquina e de um software, estão sujeitas a restrições físicas insuperáveis. Enquanto a máquina estiver desligada, nenhum evento externo ocorrido no ambiente poderá ser percebido, da mesma forma que nenhum evento interno poderá acontecer. Portanto, o sistema somente poderá comportar-se inteligentemente enquanto a máquina estiver ligada.

Algumas considerações fazem-se necessárias neste ponto objetivando esclarecer determinadas questões pertinentes à definição apresentada.

Daquela definição infere-se a noção de inteligência de máquina apresentada por Costa(1993,pág.135):

> *"Inteligência de máquina refere-se exatamente à estrutura (operacional e funcional) da máquina encarregada de realizar a regulação da interação funcional que ela mantém com o ambiente".*

Decorrente disto, a última parte da definição remete ao problema do frame, já que, a máquina terá períodos intermitentes de funcionamento e desligamento ao longo do tempo. E a cada nova "inicialização", as condições do ambiente podem ser completamente diferentes daquelas encontradas durante a última execução do sistema.

A definição apresentada difere do conceito anterior, mencionado no início da seção 2.2$_{pág.55}$, na medida em que aquele expressa um sistema com características exógenas, enquanto que o modelo proposto reconhece no sistema operacional uma camada de software, pressupondo, portanto, que outras camadas devem existir – e efetivamente existem. Isto porque, no presente contexto, o sistema operacional é um todo e não mais uma camada ou parte de um todo. À medida que se relaciona com o mundo e com cada usuário em particular, o sistema vai adquirindo conhecimento e aprimorando sua capacidade de auxiliar na solução de problemas relativos ao domínio da aplicação de cada usuário individualmente. Trata-se, portanto, de um sistema endógeno.

Um aspecto importante a destacar é que a característica de identidade do sistema remete à área de sistemas multiagentes (SMA), "os quais têm por objetivo o estudo da coletividade e não de um único indivíduo. Assim sendo, deixam de ter atenção as

---

[69] Conforme Costa(1993), "o termo conduta adotado por Piaget significa mais que comportamento, não só porque diz respeito a processos internos, que a visão comportamentalista esquece, como também porque diz respeito à totalidade da ação do indivíduo e não só à seqüência de atos que ele realiza numa situação determinada."

[70] Costa(1993,pág.135) define interação funcional de uma máquina com seu ambiente como a interação em que se dá o estabelecimento recíproco de funções entre a máquina e o ambiente.

iniciativas, sejam mental (IA simbolista) ou neural (IA conexionista), de compreender e simular o comportamento humano isoladamente, passando o foco da atenção para a forma de interação entre as entidades que formam o sistema e para a sua organização." (HÜBNER,2003). Este paradigma é motivado pela observação de alguns sistemas naturais, nos quais se percebe o surgimento de um comportamento inteligente a partir da interação de seus elementos (JOHNSON,2001 apud HÜBNER,2003).

Tomando-se o sistema baseado em conhecimento como um indivíduo, os avanços nas pesquisas em SMA, no sentido de estudar o comportamento de um grupo organizado de indivíduos (agentes autônomos), podem ser aplicados para especificar como um subconjunto de tais indivíduos pode cooperar na resolução de problemas que estão além da capacidade de solução de cada indivíduo em particular. Cabe destacar, portanto, que o estudo de Hübner (2003) na elucidação do problema de reorganização em sistemas multiagentes, adéqua-se perfeitamente aos propósitos do presente trabalho. Como exemplo pode-se utilizar uma situação que atualmente é corriqueira: o compartilhamento de arquivos em rede – no modelo proposto, esta operação é substituída pela troca de conhecimento entre os indivíduos envolvidos, através da troca de planos de ação e meta-planos conceituais, estrutura que será apresentada posteriormente.

O modelo proposto deve permitir que tanto as aplicações, como o sistema, estejam sempre cientes de sua atual situação. Uma máquina de inferência, que avalie as transições de estado, pode identificar alterações de comportamento de forma relativamente simples em comparação ao modelo atual, onde estas questões passam pela geração de *logs* contendo registros de comportamento para posterior análise. Portanto, o aspecto de localização é um requisito importante a ser considerado em qualquer projeto de um novo sistema operacional. Neste sentido Schaad (1998) afirma que:

> *"Estar localizado significa estar em contínua interação com um ambiente autonomamente ativo, dinâmico e possivelmente hostil. Isto significa trocar informações, forças ou energia com este ambiente. Isto também implica em que o ambiente é afetado pelo agente e o agente é afetado pelo ambiente.*

O problema do frame remete à condição de produtividade do sistema, a qual é formalmente apresentada a seguir:

Sendo **A** um ambiente de funcionamento e **A'** outro, o sistema será tão mais eficiente quanto maior for o grau de similaridade entre **A** e **A'**. Isto porque, em um

ambiente estável, todas as atividades de "localização e adaptação" a mudanças ambientais são minimizadas, permitindo ao sistema dedicar-se a outras atividades de interação com o meio. A medida em que **A** for diferenciando-se de **A'**, as atividades de localização passam a requerer mais atenção do sistema, levando-o a uma situação de absoluto "stress", isto é, uma parcela significativa da capacidade computacional será destinada à "adaptação" do sistema ao novo meio.

Cabe destacar que a definição apresentada deixa em aberto dois aspectos importantes:

- A questão da autonomia axiológica (Costa, 1993, pág.33), isto é, a questão de cada máquina poder ou não desenvolver valores e normas particulares para decidir sobre a conveniência e a finalidade de suas ações; e
- A questão da inteligência de máquina. Segundo Costa (1993, pág.133), "a medida em que a inteligência tem uma definição funcional, toda estrutura de qualquer sistema que realize uma função correspondente a essa nesse sistema, pode também, em princípio, ser chamada de inteligência desse sistema. Quer dizer, é o aspecto funcional da inteligência, e não seu aspecto estrutural, que justifica falar simultaneamente em inteligência humana, animal, vegetal ou de máquina, independentemente das diferenças de estrutura que estas inteligências possam apresentar. (...) Contudo, se a inteligência é uma estrutura cuja função é regular as interações funcionais do indivíduo com o meio, nem toda a estrutura que cumpre esta função é uma estrutura de inteligência. Caso contrário, os reflexos inatos e os instintos seriam estruturas de inteligência, e a noção de inteligência perderia sua especificidade. A inteligência é a estrutura que regula as interações funcionais que são, ao mesmo tempo, adquiridas e intencionais. (...) Quer dizer, no estágio atual do desenvolvimento da ciência da computação, o ponto de vista realista em inteligência artificial deve contentar-se com uma situação intermediária, em que a possibilidade de máquinas capazes de condutas adquiridas já pode ser vislumbrada, mas a intencionalidade que permite especificar as estruturas propriamente de inteligência nem sequer dá indicações de ser possível".

Em função destes argumentos, Costa(1993,pág.134) afirma que "a noção de inteligência de máquina só se mostra, no presente, ao nível do que podemos chamar de pré-inteligência e o trabalho em inteligência de máquina tem necessariamente de restringir-se, por enquanto, ao problema da aquisição não intencional de condutas".

É justamente em função desta constatação que se procurou caracterizar, na definição, um sistema que apresente conduta inteligente e não um sistema que seja inteligente.

A definição apresentada e as considerações precedentes permitem que sejam estabelecidos os requisitos funcionais deste sistema, os quais são discutidos na seção a seguir.

7.3 Identificação dos requisitos do modelo proposto

Nesta seção são apresentados os aspectos relativos aos requisitos funcionais e requisitos não funcionais do modelo proposto.

7.3.1 Requisitos funcionais da nova geração de sistemas operacionais

> *"(...) quanto mais um computador sabe sobre um usuário, melhor ele pode servir àquele usuário. Contudo, há diferentes estilos e filosofias que pretendem esclarecer qual o melhor método para ensinar nossos computadores sobre nós, sobre nossos hábitos, interesses, padrões e preferências. (...)" Orvant (1996).*

Segundo Sztajnberg (1999), a separação de interesses divide o projeto de uma aplicação em especificações de requisitos funcionais e não-funcionais. Aspectos relacionados com as computações básicas de uma aplicação podem idealmente ser tratados independentemente de aspectos não-funcionais, como a coordenação, comunicação e distribuição. Aspectos não funcionais, como protocolos de comunicação, QoS e gerência de recursos do ambiente de execução, considerados operacionais, também podem ser tratados separadamente. Robertson e Robertson (1999) destacam que os "requisitos são coisas a descobrir antes de começar a construir um produto."

Em Mattos e Pacheco (2002) foram relacionados alguns requisitos funcionais[71], os quais são a seguir ampliados em função do atual estágio do trabalho:

RF1 O sistema deve ser descrito a partir de um modelo conceitual formalmente verificável, capaz de descrever e dirigir sua dinâmica. Desvios de comportamento, em relação aos requisitos estabelecidos, devem ser detectáveis para garantir a consistência do sistema;

RF2 As aplicações devem ser orientadas pelo modelo dinâmico fornecido ao sistema operacional como condição para a execução das mesmas. Desvios de comportamento relacionados com o modelo da aplicação, também devem ser detectáveis para garantir a consistência do sistema;

RF3 O sistema deve construir associações entre os dados de uma forma que lhe permita traçar um perfil do usuário e auxiliá-lo nas tarefas diárias;

RF4 O sistema deve possuir uma interface de aprendizagem que viabilize tanto a introdução de conhecimento factual (dados), como de conhecimento procedimental (planos);

RF5 A interface de aprendizagem deve permitir quatro tipos de interação:

RF5.1 Interação com o usuário final, onde o processo de assimilação de conhecimento tem muito mais a ver com o perfil de uso da máquina, através de inferências sobre conhecimentos procedimentais previamente assimilados (programas no modelo

[71] Segundo Gause e Weinberg (pág 186,1991) , "para testar se um requisito é realmente uma função, ponha a frase 'Queremos o produto para...' na frente do mesmo. Podemos dizer, de forma alternativa: 'O produto deve...' ".

atual), relativos a área de domínio de uma determinada aplicação. Neste caso a interação deve facilitar as operações do usuário;

RF5.2 Interação com o conhecimento codificado, fornecido pelo desenvolvedor (programador no modelo atual) da base de conhecimento procedimental sobre determinada área de aplicação (procedimento de *setup* de aplicações no modelo atual). Neste caso a interação deve consistir de um processo criterioso de assimilação e verificação do conhecimento que está sendo introduzido contra as regras de autoproteção do sistema;

RF5.3 Interação com outros sistemas baseados em conhecimento, através da interface de aprendizagem;

RF5.4 Utilização de dados armazenados em arquivos de outros sistemas baseados no modelo atual (FAT, NTFS, HPFS, etc.)

RF6 Deve haver uma forma de representação do conhecimento procedimental[72] (atualmente embutido no código binário) em lugar do código executável, de forma a permitir que, através da interface de aprendizagem, o sistema possa assimilar este tipo de conhecimento e ampliar a sua base;

RF7 O sistema deve possuir conhecimento suficiente para converter especificações funcionais em código binário (nativo da máquina alvo), quando necessário, tendo em vista aumentar a performance de determinadas situações;

RF8 O sistema deve possuir conhecimento suficiente para armazenar dados factuais segundo alguma estratégia que preserve sua privacidade;

RF9 A estratégia de privacidade dos dados armazenados deve ser tal que duas máquinas, executando o mesmo sistema operacional baseado em conhecimento, tenham estratégias diferentes de armazenamento e localização física dos dados por elas gerenciados;

RF10 O sistema deve suportar a separação entre formato de arquivo e conteúdo;

RF11 Somente quando necessário, o sistema deve transformar conhecimento armazenado em arquivos, segundo a concepção atual de arquivo, de acordo com algum layout previamente assimilado. Isto implica em que, internamente, só há conhecimento, não existem arquivos. O conceito explícito de arquivo deverá existir somente em situações onde o usuário precisa transferir o conhecimento para outro usuário ou, por exemplo, gerar uma representação escrita em papel;

RF12 O sistema deve possuir o conceito equivalente ao de "relógio biológico" implícito[73], a fim de poder inferir sobre o seu estado geral de funcionamento e controlar mais naturalmente o escalonamento das tarefas a serem executadas;

RF13 O sistema deve possuir uma rede de sensores (físicos e lógicos) que lhe permitam sentir instantaneamente quaisquer alterações ambientais e executar ações apropriadas.

Como requisitos funcionais secundários, que contribuirão para que o sistema apresente uma abstração de comportamento inteligente, estão as seguintes características compiladas a partir de Orvant (1996) e Patki e Raghunathan (1996):

RF14 Utilizar técnicas de personalização para identificação de perfil de usuário e adequação do ambiente computacional a este perfil.

[72] Atualmente na forma de código binário, o qual pode ser executado diretamente pela máquina ou interpretado através de um interpretador de máquina virtual.

[73] O hardware continua tendo o relógio de tempo-real explícito segundo o modelo atual.

RF15 Capacidade de tratar informações parciais e/ou imprecisas;

RF16 Capacidade de extração de conceitos;

RF17 Capacidade de raciocínio aproximado e aprendizado;

RF18 Facilidades de utilização por usuários sem experiência com computadores;

RF19 Capacidade de autodiagnóstico do sistema para facilitar as atividades de administração e interação.

Alguns requisitos não funcionais podem ser identificados já nesta fase do projeto, os quais são elencados na próxima seção.

7.3.2 Requisitos Não Funcionais

Tomando-se por base a relação de requisitos funcionais, de imediato é possível identificar uma primeira relação de requisitos não funcionais, quais sejam:

RNF1. Modelo de mundo baseado em máquina de estados;

RNF2. Modelo de representação de conhecimento procedimental através de especificação executável na forma de máquina de estados estendida;

RNF3. Especificação da máquina de inferência, em substituição ao núcleo de sistema operacional, de acordo com os requisitos RNF1 e RNF2;

RNF4. Estratégia de representação de planos conforme descrita em Schaad (1998);

RNF5. Modelo de execução baseado no formalismo DEVS (*Discrete Event System Specification*).

A seção 7.3.4 apresenta uma avaliação da viabilidade dos requisitos funcionais e não funcionais identificados, onde são discutidos os aspectos mais importantes que levaram ao modelo proposto.

7.3.3 Comparativo entre requisitos funcionais e não funcionais

A tabela abaixo apresenta um comparativo entre os requisitos funcionais identificados nos capítulos anteriores e os requisitos apresentados no capítulo atual, permitindo uma antecipação de como tais requisitos estão relacionados ao contexto.

Tabela 25 - Comparativo entre requisitos funcionais e não funcionais

Requisitos de Sistemas Operacionais	Requisitos de Computação Ubíqua, Teletrabalho e Comércio Eletrônico.	Requisitos de Robótica	Sistema operacional baseado em conhecimento: Requisitos Funcionais	Sistema operacional baseado em conhecimento: Requisitos Não Funcionais
• Estrutura organizacional (monolítico, camadas, etc).		• Modularidade e flexibilidade.	• 1-Modelo formalmente verificável.	• Máquinas de estados.
• Performance do núcleo/ serviços do sistema.				• Planos.
• Modelo de aplicação com código compilado estático com suporte a: (i)*Late binding*; (ii)Código interpretado; (iii) Pesquisas em paralelismo de instruções (*pre-fetch*).				• Especificação executável (planos).
• Políticas de segurança: permissões, restrição de acesso, *capabilities*.			• 8-Privacidade.	• Planos.
• Primitivas de sincronização e intercomunicação de processos (*threads*).				• Máquina de inferência.
• Suporte a uma ou mais políticas de escalonamento.				• Planos.
• Suporte a 1+ estratégias de gerenciamento de memória.				• SASOS (Single Address Space Operating Systems).
• Suporte a tratamento de exceções.				• Planos.
• Suporte a gerência de meios de armazenamento.	• Sistema de arquivos com maior nível de abstração.			• Máquina de inferência e estratégia de representação de conhecimento.
• Interface do sistema com as aplicações através de APIs.			• 4-interface de aprendizagem.	• Interface com a máquina de inferência.
• Facilidade de extensão dos	• Facilidades para usuários	• Adaptabilidade.	• 5-Tipos de interação.	

serviços do sistema para permitir a adaptação por parte das aplicações.	especializados aumentarem as bibliotecas do sistema operacional sem comprometimento dos aspectos de segurança e confiabilidade.	• Facilidade de expansão.		
• Suporte à multitarefa.				Máquina de inferência.
• Compatibilidade (ou não) com versões anteriores.				Planos estabelecendo estratégia de conversão.
• Intercomunicação e interligação com outros equipamentos.			• 5-Tipos de interação.	
• Interfaces de interação com o usuário gráficas (e interação por voz).	• Interação com o usuário.	• Interação entre Agente-Agente e Agente-Humanos.	• 5-Tipos de interação.	
• Controle de recursos.	• Controle de recursos com QoS.			Planos.
• Robustez.	• Robustez.	• Robustez e confiabilidade.	• 2-Aplicações dirigidas pelo modelo dinâmico.	
• Política de segurança.	• Segurança.		• 8-Privacidade. • 11-Transformação de conhecimento em arquivo.	
• Administração do sistema.			• 4-interface de aprendizagem.	
	• Mobilidade de código estático e dinâmico.	• Mobilidade.	• 5-Tipos de interação.	• Planos de interação com outras entidades inteligentes ou não.
	• Capacidade de tratar info. parciais e/ou imprecisas.	• Resolução de múltiplos objetivos.	• 4-Interface de aprendizagem.	• Máquina de inferência.
	• Capac. de extração de conceitos.		• 4-Interface de aprendizagem.	• Máquina de inferência.
	• Capac. de raciocínio aprox. (difuso) e de aprendizado.	• Raciocínio global.	• 4-Interface de aprendizagem.	• Máquina de inferência.
	• Ident.o perfil de usuário e adequação do ambiente ao mesmo.		• 3-Construção de associações.	
		• Reatividade ao ambiente.	• 13-Rede de sensores.	
		• Integração de múltiplos sensores.	• 13-Rede de sensores.	
		• Autonomia.		
		• Comportamento inteligente.		
	• Privacidade de informações sobre transações.		• 9-Estratégia de privacidade.	
	• Confidencialidade das informações.		• 10-Separação entre conhecimento e formato de arquivos.	
	• Integridade das informações.		• 9-Estratégia de privacidade.	• Planos.
			• 6-Substituição de código executável.	• Planos.
			• Otimização dos planos.	• Gerador de código para o processador alvo.
			• Relógio de referência.	• Formalismo DEVS.

7.3.4 Avaliação da Viabilidade dos Requisitos Identificados

Uma análise menos criteriosa do conjunto de requisitos funcionais e não funcionais poderia classificar o projeto atual como um modelo de máquina de inferência semelhante àquelas existentes e amplamente descritas na literatura de Sistemas Especialistas, em particular, e de Inteligência Artificial, em geral. Ao confundir-se o sistema proposto com uma máquina de inferência tradicional, corre-se o risco de retornar ao tripé conceitual discutido anteriormente e, dessa forma, reeditar o circulo vicioso que o caracteriza.

Uma tentativa de contornar este problema foi apresentada pelo projeto **Genera**[74] (GENERA,1990), que descreve um sistema operacional baseado na implementação da máquina LISP diretamente sobre o hardware, através de microprogramação. Embora a máquina *Symbolics Common Lisp* em particular tenha sido concebida como um produto comercial, a relação de problemas relatados demonstra efetivamente que não basta implementá-la sem atacar a questão do tripé conceitual.

Em particular, o uso de máquinas de inferência remete ao problema de *symbol grounding* e a representação de ontologias, as quais representam significações e associações de conceitos que não foram experimentados nem pelo usuário nem pelo sistema operacional. Portanto não houve apropriação deste conhecimento de forma procedimental, causal e temporal. Para não remeter a questão da necessidade de criação de grandes ontologias, como aquela desenvolvida no projeto **CYC**[75], deve-se esclarecer que o conceito de programa utilizado nos modelos anteriores transforma-se agora no conceito de plano, conforme utilizado nas áreas de Inteligência Artificial e Robótica. Um programa que anteriormente incorporava o conhecimento procedimental de forma absolutamente incompreensível, agora passa a representar o mesmo conhecimento na forma de planos de execução. Portanto, as informações ontológicas relativas a cada aplicação devem ser explicitamente informadas ao sistema, de tal forma que ele possa assimilá-las e utilizá-las para auxiliar o usuário no desenvolvimento de suas atividades específicas naquele domínio.

---

[74] Seção 2.4.12 (Pág.79).

[75] Segundo Bittencourt (1998), a base de conhecimento contém 100 milhões de asserções divididas em 1 milhão de quadros. O projeto pretende construir o primeiro sistema artificial que apresente inteligência geral e senso comum (www.cyc.com).

Tendo em vista esta questão, o modelo de mundo proposto é apresentado no decorrer da próxima seção.

7.4 O Modelo Proposto

O conceito fundamental sobre o qual assenta-se esta proposta de sistema operacional baseado em conhecimento, é o modelo de mundo.

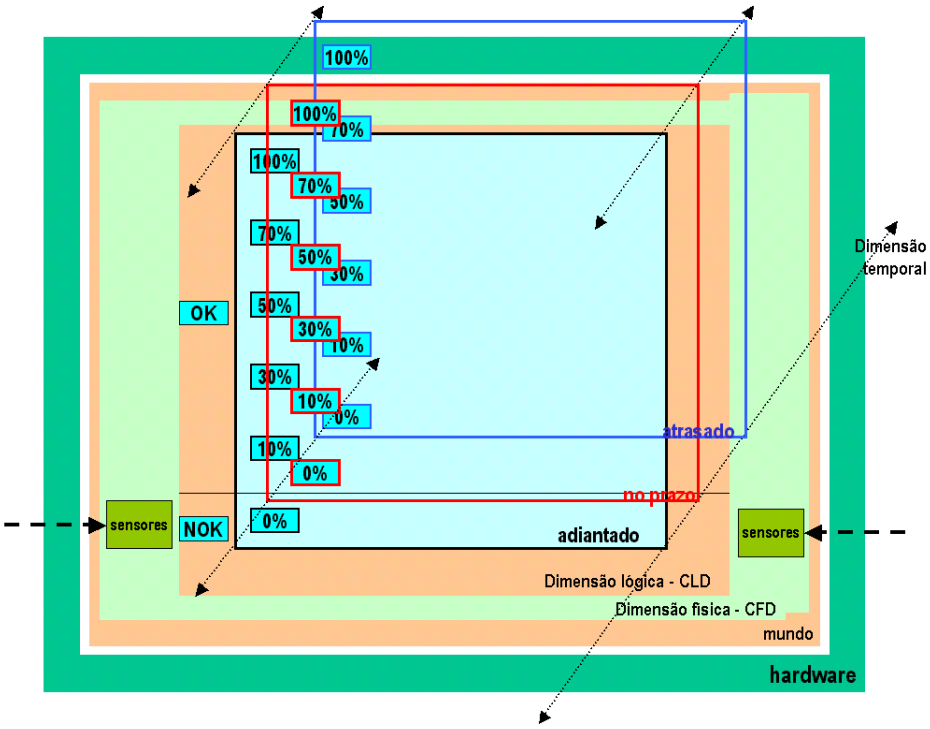


Figura 48 – Modelo de mundo hiperdimensional

Na Figura 48 é destacado o caráter endógeno do sistema na medida em que o modelo de mundo é envolvido pelo hardware, isto é, qualquer forma de acesso ao modelo de mundo ocorre através do hardware. Dentro desta camada está inserido o modelo de mundo, o qual pode ser caracterizado como a membrana envoltória de uma célula inserida no meio circundante (hardware). Portanto, a membrana faz as vezes de interface entre o meio externo e o meio interno.

Utilizando-se a analogia da célula, não há como romper a membrana para acessar as partes internas da célula. Assim, qualquer forma de influência no comportamento da célula deverá ocorrer num processo semelhante ao da osmose, ou seja, fornecem-se informações à interface e esta as traduz para a representação interna da célula.

Na parte interna do modelo de mundo estão:

- O modelo de tempo;
- A máquina virtual **InSitu**;
- O escalonador de tarefas **DEVS**;
- O modelo de aplicações.

Nas próximas seções estes componentes são detalhados.

7.4.1 O modelo de tempo

A dimensão temporal está relacionada aos aspectos relativos ao conceito de tempo que o sistema incorporado (*embodied*) consegue perceber como entidade inteligente. As considerações que sustentam a necessidade desta dimensão ser tratada explicitamente no modelo de mundo proposto foram apresentadas no capítulo anterior (seção 6.3$_{\text{pág.194}}$).

Contudo, em função dos requisitos do modelo proposto, houve necessidade de conceber uma nova forma de representação tempo, a qual buscou inspiração na especificação das estruturas ***quadtrees***. À estrutura concebida, denominou-se: ***sensores lógicos, que é detalhado a seguir***.

7.4.1.1 O modelo de sensores lógicos

Ao introduzir o conceito de tempo implícito, impõem-se alguns requisitos adicionais às aplicações, já que elas, mesmas sentindo o tempo passar, poderão alterar a estratégia de solução de determinado problema, adequando-se a flutuações de carga.

A solução proposta para a questão dos sensores lógicos, que traduzem eventos do mundo real para eventos do modelo de mundo, segue a filosofia de sensores/atuadores lógicos (seção 4.2.3 -Lógica difusa$_{\text{Pág.116}}$) descrito no trabalho de Reed (2001). No entanto, no presente trabalho adotou-se a estrutura *quadtree*[76] para representar eventos instantâneos e históricos, como será descrito a seguir.

Segundo Freitas (1998), a ***quadtree*** foi desenvolvida para permitir , por exemplo, representação de regiões num plano. A estrutura consiste de uma árvore onde a raiz representa uma região retangular que contém o objeto. Cada um de seus quatro nodos

[76] Klinger e Alexandridis(1978) propuseram uma estrutura de dados em árvore mais conhecida como quadtree para as aplicações que exigiam decomposição de figuras. Contudo, estruturas análogas à quadtrees foram empregadas por outros autores em problemas diversos daquele de decomposição de figuras, como apresentado em Samet (1984).

filhos representa um quadrante desse retângulo e a divisão prossegue sucessivamente até ser atingida a resolução de representação adequada. Cada nodo da árvore traz a indicação de ocupação total, parcial ou ausência do objeto no quadrante que o nodo representa (Figura 49).

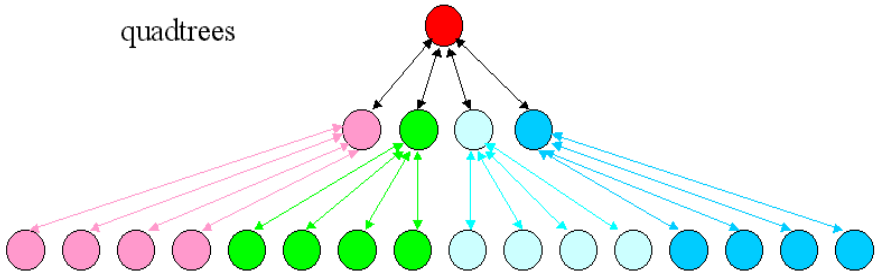


Figura 49 – Estrutura de uma quadtree

O mesmo tipo de estrutura, estendido para o espaço 3D, constitui o que é conhecido por ***octree***. Estas estruturas são utilizadas, também, para organizar o espaço dos objetos representados segundo outras formas de modelagem, de modo a acelerar procedimentos (FREITAS,1998).

Esta estrutura tem sido utilizada em várias áreas de aplicação geralmente associadas à Computação Gráfica. No entanto, esta mesma estrutura mostrou-se adequada à construção dos sensores lógicos do modelo proposto.

As próximas seções apresentam casos de uso do modelo de sensores lógicos, tendo em vista caracterizar o seu funcionamento.

7.4.1.1.1 *O uso de sensores lógicos para indicação de eventos*

Retomando a afirmação de Einstein (2001,pág.14), de que toda a descrição do lugar onde ocorreu um evento ou onde se encontra um objeto, baseia-se na indicação do ponto de um corpo rígido (corpo de referência) com o qual o evento coincide, pode-se verificar que os eventos que ocorrem em um sistema operacional, via de regra, são dissociados de corpo de referencia. Eles não ocorrem em lugar algum e, no máximo, ocorrem em um tempo absoluto (hora atual) sem qualquer relação com pelo menos uma base de referência.

A estrutura *quadtree* é adequada porque permite a divisão de um intervalo em sub-intervalos, caracterizando uma estrutura homogênea. Tomando-se um intervalo de 24

horas e utilizando-se a *quadtree* para representar este intervalo de tempo, ter-se-ia 4 divisões de 6 horas cada. Descendo-se um nível da estrutura, ter-se-ia 16 divisões das mesmas 24 horas em intervalos de 1,5 horas (1h30min). Uma nova sub-divisão remeteria as mesmas 24 horas a um intervalo de 24/64 horas e assim sucessivamente.

Definindo-se uma estrutura de sensor lógico e estabelecendo-se a seguinte semântica: todo evento sinalizado na base da árvore deverá sinalizar também o seu nodo raiz correspondente e assim sucessivamente até atingir o nodo raiz – obtém-se uma estrutura de sinalização instantânea de eventos com base de referência implícita e com granularidade ajustável à demanda de cada aplicação em particular. A Figura 50 demonstra o uso do modelo de sensores lógicos para representar eventos localizados no tempo.

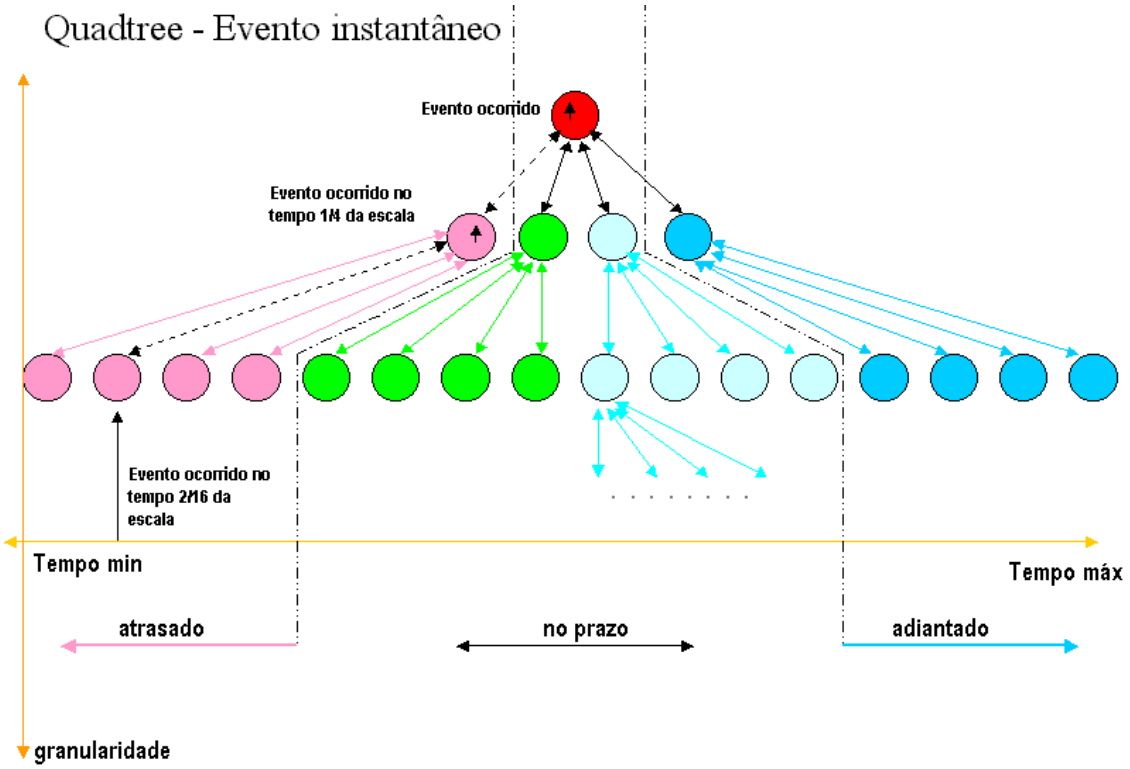


Figura 50 – Representação de evento instantâneo localizado numa base de tempo

Uma função **f(x)** de mapeamento converte o instante da ocorrência do evento para uma base de representação considerando o nível de granularidade estabelecido da seguinte forma:

Posicao_na_estrutura = f(instante_ocorrencia_evento,base_referencia_temporal).

Esta função de mapeamento não precisa necessariamente trabalhar com intervalos de base de tamanho fixo. Enquanto para uma aplicação específica os intervalos podem ser de tamanho fixo, para outras pode utilizada uma escala exponencial para a seleção do nodo da árvore a ser marcado.

Aos nodos da árvore podem ser associadas variáveis lingüísticas, de forma a facilitar o uso em sistemas de inferência de forma mais natural. Deve-se destacar que, embora tratando-se de variáveis lingüísticas, sua representação interna não é simbólica. Assim sendo, elas devem ser implementadas na forma de *alias* sobre a estrutura original.

Outro aspecto a ser destacado é o fato de que o termo variável lingüística toma emprestada da lógica difusa a idéia de que os termos denotam intervalos não muito precisos em relação à realidade. De fato, à medida que novos sub-níveis são agregados à estrutura, uma maior precisão da informação representada pode ser obtida.

Deve-se salientar que esta estrutura não somente permite a sinalização da ocorrência de um evento, mas simultaneamente o faz dentro de uma base de referência com limites conhecidos. A estrutura hierárquica permite a realização de inferências com maior ou menor grau de precisão a partir da seleção do nível ou sub-nível da estrutura a ser utilizado.

Tomando-se o contexto representado pela Figura 50, poder-se-ia considerar a seguinte situação:

- Base de referência: 24 horas
    - Tempo min: hora 0:00 do dia;
    - Tempo max: hora 23:59 do dia;

Neste caso, a ocorrência de algum evento poderia ser sinalizada da seguinte forma:

- **Ocorreu**: se analisada a raiz da estrutura;
- **Ocorreu no primeiro quarto do dia**: se analisado o segundo nível da estrutura
- **Ocorreu aproximadamente 1:30 após o início da contagem da base de referência**: se analisado o terceiro nível da estrutura.

Neste caso a granularidade da precisão da informação pode ser tão grande ou tão pequena quanto seja necessário para uma aplicação em determinado momento. No exemplo acima, o próximo nível de resolução seria de 24h/64 segmentos de precisão. O ajuste de granularidade pode ser realizado identificando-se no nodo raiz o número de níveis da estrutura e a unidade de representação subdividida: dia, mês, ano, hora, minuto, segundo ou outras unidades relacionadas ao domínio da aplicação.

O mesmo contexto representado pela Figura 50 poderia ser analisado sob uma diferente base de referência, sem contudo alterar a estrutura ou a mecânica de funcionamento:

- Base de referência: 1 mês
    - Tempo min: 1º dia do mês;
    - Tempo max: 30º dia do mês;

Neste caso, a ocorrência de algum evento poderia ser sinalizada da seguinte forma:

- **Ocorreu**: se analisada a raiz da estrutura;
- **Ocorreu na primeira semana**: se analisado o segundo nível da estrutura
- **Ocorreu aproximadamente no segundo dia do mês**: se analisado o terceiro nível da estrutura.

Novamente a granularidade da precisão da informação pode ser tão grande ou tão pequena quanto seja necessário para uma aplicação em determinado momento. No exemplo acima, o próximo nível de resolução seria de 30 dias/64 segmentos de precisão.

7.4.1.1.2 *O uso de sensores lógicos para indicação de histórico de eventos*

Esta mesma estrutura pode ser utilizada no registro de eventos históricos, respeitando-se a mesma semântica de sinalização descrita anteriormente.

A questão dos históricos remete à geração de *logs* de eventos ocorridos[77] para posterior análise. Esta tarefa geralmente consome muitos recursos computacionais tendo em vista que, embora a aquisição possa ser relativamente eficiente, conforme descrito na seção 2.5 - Técnicas para obtenção de dados do núcleo$_{\text{pág.83}}$, o armazenamento (geralmente arquivos texto) gera demandas extras de acesso a disco e o posterior

[77] Exemplos deste tipo de arquivo podem ser consultados nos apêndices: 10.3$_{\text{pág.363}}$, 10.4$_{\text{pág.365}}$ e 10.5$_{\text{pág.368}}$.

processamento, além de consumir tempo de processador, também gera novas demandas de acesso a disco, comprometendo a performance de ambientes que já estão carregados.

Vias de regra, arquivos de *logs* são gerados sempre que algum problema ocorreu. Não sendo possível gerá-los de forma isolada, isto é, sem comprometer as estruturas que podem ter sido danificadas em função do próprio problema em análise e sem comprometer a performance de um sistema já carregado, a regra prática utilizada é:

- Ativar a opção de *log* por pouco tempo;
- Coletar os dados; e
- Enquanto estiver sendo realizada uma análise dos dados gerados, desligar a opção de *log*.

A estrutura de sensores lógicos pode ser expandida para incluir o seguinte comportamento:

- Nodos sinalizados permanecem sinalizados e,
- A propagação do sinal pode obedecer a regras de limiar (threshold) (ex: sinaliza o nível superior quando 50% dos nodos do nível inferior houverm sido sinalizados).

Com este comportamento semântico, poder-se-ia observar o histórico de uso de determinado programa, simplesmente sinalizando os sensores lógicos no momento de disparo do referido programa. Dependendo da base de referência e preservando-se os sinais anteriores, pode-se obter um longo histórico com uma estrutura relativamente eficiente e compacta.

Neste sentido, o uso de sensores lógicos para representar eventos localizados pode ser caracterizado como mostrado na Figura 51. Tomando-se o contexto representado pela figura, poder-se-ia considerar a seguinte situação:

- Base de referência: 24 horas
    - Tempo min: hora 0:00 do dia;
    - Tempo max: hora 23:59 do dia;

Neste caso, a ocorrência de algum evento poderia ser sinalizada da seguinte forma:

- **Ocorreu**: se analisada a raiz da árvore;
- **Ocorreu no primeiro quarto do dia**: se analisado o segundo nível da árvore
- **Ocorreu aproximadamente 1:30 após o início da contagem da base de referência**: se analisado o terceiro nível da estrutura;

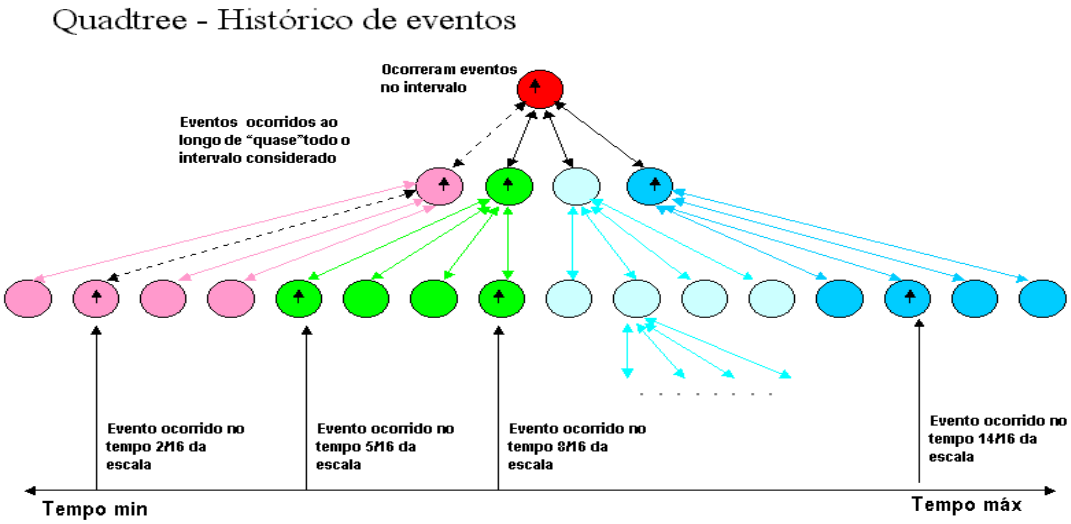


Figura 51 – Sensores lógicos representando histórico de eventos

Neste caso a granularidade da precisão da informação pode ser tão grande (ou pequena) quanto seja necessário para uma aplicação em determinado momento. No exemplo acima, o próximo nível de resolução seria de 24h/64 segmentos de precisão.

Por outro lado, o mesmo contexto representado pela Figura 51, poderia ser analisado sob uma diferente base de referência sem contudo alterar a estrutura ou a mecânica de funcionamento:

- Base de referência: 1 mês
  - Tempo min: 1º dia do mês;
  - Tempo max: 30º dia do mês;

Neste caso, a ocorrência de algum evento poderia ser sinalizada da seguinte forma:

- **Ocorreu**: se analisada a raiz da estrutura;
- **Ocorreu na primeira semana**: se analisado o segundo nível da estrutura
- **Ocorreu aproximadamente no segundo dia do mês**: se analisado o terceiro nível da estrutura;

Novamente a granularidade da precisão da informação pode ser tão grande (ou pequena) quanto seja necessário para uma aplicação em determinado momento. No exemplo acima, o próximo nível de resolução seria de 30 dias/64 segmentos de precisão.

7.4.1.1.3 *O uso de sensores lógicos para representação de espaço*

Uma decorrência da generalidade da estrutura de sensores lógicos é que a ela pode ser utilizada não apenas para a representação de eventos temporais, mas também de eventos espaciais. A Figura 52 demonstra a utilização da estrutura de sensores lógicos para representar o estado lógico do espaço em disco ocupado em determinado momento. Uma vez que a alocação/liberação de cada setor (ou *cluster*) é sinalizada na estrutura, a base de referência, neste caso, passa a ser uma unidade de medida de espaço máximo disponível de determinado disco.

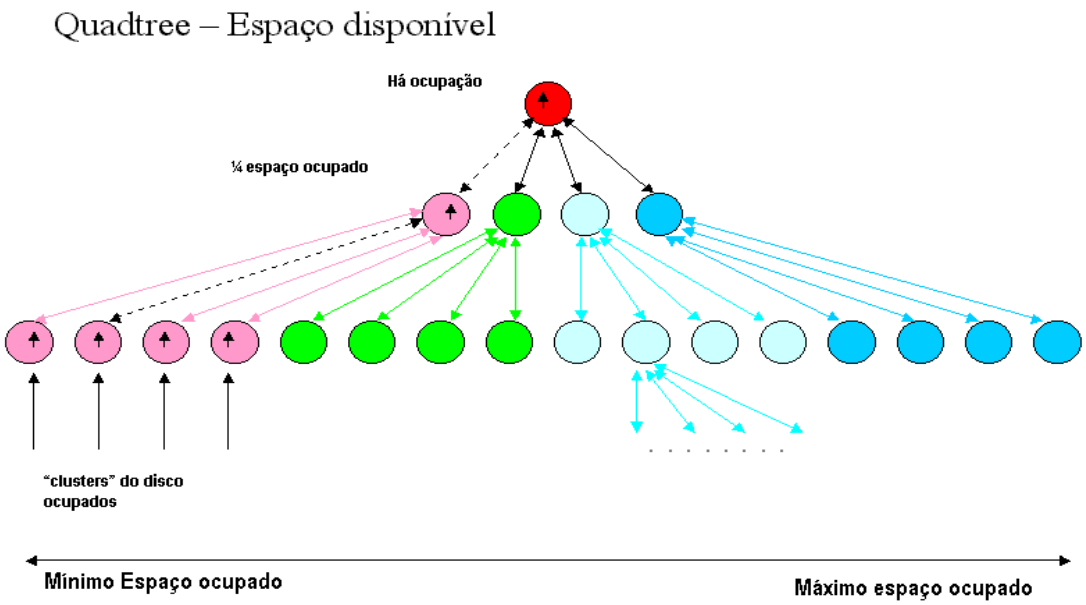


Figura 52 – Sensores lógicos representando espaço disponível em um disco

Novamente é possível realizar-se uma macro-análise prévia, antes de efetivamente recorrer a uma análise detalhada sobre os setores ou *clusters* utilizados. É esta representação que permite a implementação do modelo de dispositivos lógicos de forma eficiente, já que qualquer acesso ao disco é indicado imediatamente nesta estrutura. Utilizando as informações codificadas na estrutura como índice de acesso a uma tabela de seções do código da aplicação, implementa-se o conceito de instantaneidade referido anteriormente.

A Figura 53 representa outro cenário: a representação de fragmentação do disco. Neste caso, uma derivação do comportamento original da *quadtree*, através da introdução do conceito de limiar (*threshold*), permite que as significações sejam propagadas na árvore somente depois de determinado limiar ter sido atingido.

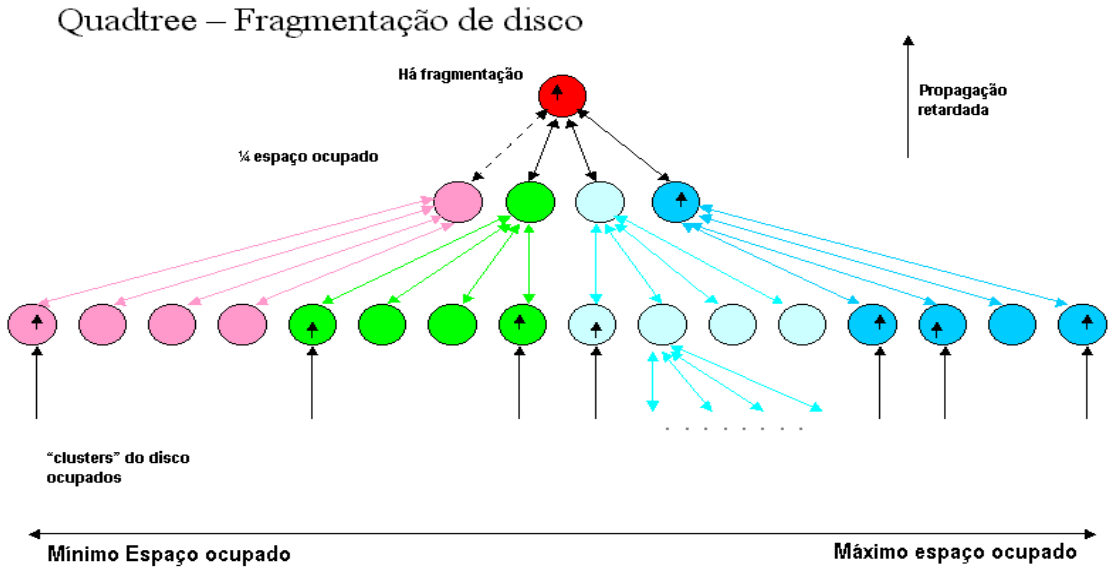


Figura 53 – Quadtree representando fragmentação de disco

7.4.1.2 Outras considerações

O uso de sensores lógicos, segundo a estrutura apresentada nesta seção, permite a identificação intuitiva de relacionamentos como aqueles apresentados em Rich e Knight (1991,pág.615). Os autores indicam que há exatamente 13 maneiras segundo as quais dois intervalos de tempo não-vazios podem estar relacionados entre si.

A Figura 54 evidencia que, na verdade, há apenas sete relacionamentos distintos: o relacionamento de igualdade mais seis outros que têm seus próprios inversos. Tomando-se 2 estruturas de sensores lógicos com a semântica descrita anteriormente e considerando-se a mesma base de referência de tempo, é possível a identificação destes relacionamentos a partir das operações lógicas apresentadas na Tabela 26.

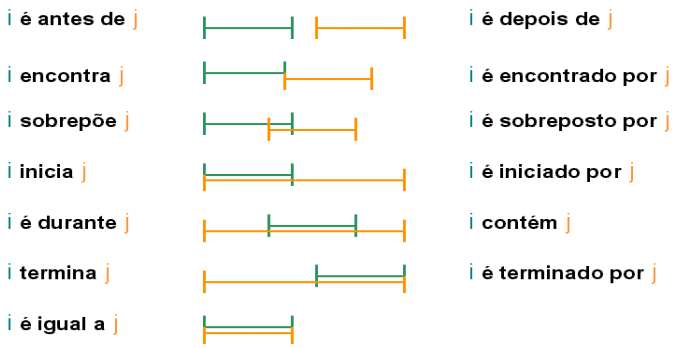


Figura 54 - Treze relacionamentos entre dois intervalos de tempo.

FONTE: Adaptado de Rich e Knight (1991, pág.615).

A partir das informações apresentadas na Tabela 26, constata-se a facilidade na expressão de eventos compostos, ou seja, a combinação da ocorrência de eventos distribuídos mais simples que são relacionados na forma de expressões de eventos do tipo: **(E_1 ^ E_2)** ou **(E_1 v E_2),** onde cada E_x é representado por uma estrutura (sensor lógico) independente.

Tabela 26 - Relacionamento entre intervalos usando quadtrees

Intervalo1<intervalo2	É antes de	É depois de
Max(Intervalo1) = min(intervalo2)	Encontra	É encontrado por
Max(intervalo1) > min(intervalo2)	Sobrepõe	É sobreposto por
((Intervalo1 or intervalo2) > intervalo1) and (min(intervalo1) = min(intervalo2))	Inicia	É iniciado por
((intervalo1 or intervalo2) = intervalo1) and (min(intervalo1)< min(intervalo2))	É durante	É iniciado por
((intervalo1 or intervalo2) = intervalo1) and (Max(intervalo1) = Max(intervalo2))	Termina	É terminado por
Intervalo1 and intervalo2 = intervalo1	É igual	

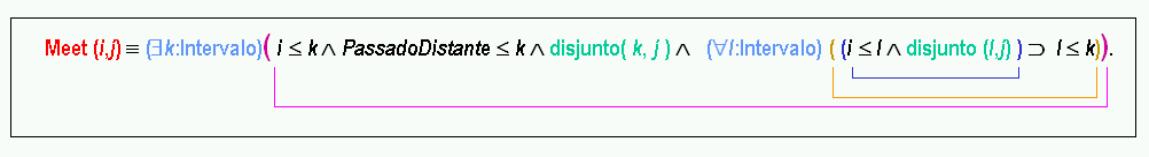
As relações definidas na Tabela 26, encontram sustentação formal e base científica nos axiomas apresentados a seguir, conforme apresentado em Sowa (2000). Segundo Sowa (2000,pág.114), em Inteligência Artificial a ontologia mais adotada para representação do tempo está baseada nos axiomas definidos por James Allen em 1984. Ainda segundo Sowa:

> *"(...)em 1985, Allen e Hayes simplificaram aqueles axiomas assumindo um único operador primitivo denominado **meet(i,j)**, o qual significa que o intervalo **i** imediatamente precede o intervalo **j.**Em termos da mereologia[78] e da geometria de Tarski, os axiomas de Allen e Hayes podem ser provados como teoremas, porém, algumas suposições adicionais são necessárias para distinguir passado e futuro. Um enfoque é assumir um intervalo especial denominado **passado distante**, o qual inclui todo o tempo que passou antes do período de interesse. Esta suposição é neutra sobre a questão se o passado distante estende-se infinitamente para trás ou se há um ponto inicial de contagem de tempo. Um outro enfoque é mapear os intervalos de Allen-Hayes para diâmetros de esferas na geometria de Tarski " Sowa (2000, pág 114).*

Segundo Sowa (2000, pág.114), a relação ***meet (i,j)*** pode ser definida, em termos de mereologia e um intervalo especial chamado ***PassadoDistante***, da seguinte forma:

- Meet: o intervalo ***i*** encontra (*meets*) o intervalo ***j*** se ambos ***i*** e ***PassadoDistante*** são parte de algum intervalo ***k*** que é disjunto de ***j***, e se ***i*** é parte de qualquer intervalo ***l*** que é disjunto de ***j***, então ***l*** é parte de ***k***.

[78] Mereology: Segundo Sowa (2000,pág 103), Lesniewski (1916) insatisfeito com a teoria de conjuntos como fundação matemática, buscou na álgebra booleana pré-Cantor a base para uma teoria mais simples e que fosse livre de paradoxos. Como aplicou-a ao estudo de partes e todos, ele denominou-a de mereologia – do grego meros=parte. Whitehead (1919), de forma independente, desenvolveu teoria similar que aplicou na descrição de eventos espaço-temporais. A primitiva básica da mereologia é ***parteDe***, a qual corresponde ao operador $\subset$ da teoria de conjuntos. Contudo, a mereologia não possui um operador separado para indicar $\in$.



Segundo Sowa (2000, pág.115) a relação ***meet(i,j)*** "é análoga às esferas externamente tangentes de Tarski. A condição adicionada sobre o passado distante garante que ***i*** precede ***j***". Allen e Hayes definiram todas as outras relações entre intervalos em termos da relação ***meet***, mas algumas de suas definições são mais simples em termos das seguintes relações mereológicas:

- Antes de (*before*): o intervalo **i** está antes do intervalo **j** se há algum intervalo **k** tal que **i** encontra (*meets*) **k** e **k** encontra (*meets*) **j**;
- Igual (*equal*): o intervalo **i** é igual ao intervalo **j** se **i** é parte de **j** e **j** é parte de **i**;
- Sobrepõe (*overlap*): o intervalo **i** sobrepõe o intervalo **j** se há três intervalos **a**,**b** e **c** tais que **i** = **a** ∪ **b**, **j**= **b** ∪ **c**, **a** encontra (*meets*) **b** e **b** encontra (*meets*) **c**;
- Durante (*during*): o intervalo **i** ocorre durante o intervalo **j** se existem dois intervalos **a** e **b** tais que **j**= **a** ∪ **i** ∪ **b**, **a** encontra (*meets*) **i**, e **i** encontra (*meets*) **b**;
- Inicia (*starts*): o intervalo i inicia o intervalo **j** se **i** é uma parte (*proper part*) de **j** e algum intervalo **k** encontra (*meets*) tanto **i** como **j**;
- Termina (*finishes*): o intervalo **i** termina o intervalo **j** se **i** é uma parte (*proper part*) de **j** e ambos **i** e **j** encontram (*meet*) algum intervalo **k**.

Além destas relações, outras podem ser definidas como inversas ou suas combinações das mesmas. Por exemplo, ***depois(i,j)*** é definida como ***antes(j,i).***

Um outro aspecto importante destacado por Sowa (2000), é que:

> *"Tendo em vista que é possível realizar-se o mapeamento entre a linha de tempo de Allen-Hayes, a geometria de Tarski e a geometria Euclidiana, todas as construções da matemática Euclidiana correspondem a construções equivalentes nas outras duas propostas. As operações no modelo Euclidiano, as quais são geralmente mais simples, são válidas para os três modelos. Além disso, as construções de Tarski podem também ser adaptadas para geometrias não Euclidianas utilizadas na teoria da Relatividade.O axioma 1 de Tarski, o qual assume que as construções de pontos a partir de esferas satisfaz os axiomas Euclidianos usuais, podem ser substituídos por axiomas que assumem qualquer versão da geometria 4-dimensional usada para a relatividade. Em todas estas variações, os intervalos de tempo são considerados primários e os instantes são considerados como ficção computacional, não realmente constituintes do tempo" Sowa (2000, pág.116).*

A utilização da estrutura de sensores lógicos é mais eficiente que a geração de arquivos de *logs*, tendo em vista que a primeira opera na ordem de microssegundos ($\mathbf{10^{-6}}$) enquanto a segunda opera na ordem de milissegundos ($\mathbf{10^{-3}}$). Neste contexto deve-se

considerar que uma operação de acesso a disco consome em média **500ms**, sem contar os tempos necessários para a formatação dos *strings*, a gerência dos *buffers* de saída e o escalonamento da cabeça de escrita/gravação, funções estas executadas pelo subsistema de E/S do sistema operacional. Além disso, deve-se destacar o fato de que o registro armazenado não agrega conhecimento ao sistema operacional. Via de regra, esta informação deverá ser utilizada pelo Operador Virtual(Pág29) para que o ele tome as providências necessárias para resolver a questão.

Outro aspecto a ser destacado é que esta estrutura apresenta as seguintes propriedades identificadas em Rich e Knight (1991, pág.130):

- Adequação representacional uma vez que os eventos são sinalizados instantaneamente, relacionados à base de referência;
- Adequação inferencial. A estrutura *quadtree* permite a identificação de relacionamentos intervalares de forma eficiente e eficaz;
- Eficácia Inferencial. Como os eventos sinalizados são relacionados imediatamente à base de referência, os mecanismos de inferência podem identificar direções promissoras e tendências a partir da manipulação de bits da estrutura; e
- Eficácia aquisitiva uma vez que o custo de aquisição da informação refere-se à lógica para ligar os bits correspondentes à posição relativa da ocorrência do evento, levando-se em conta a base de referência de tempo especificada.

A partir da especificação do modelo de sensores lógicos, retoma-se novamente a questão da dimensão temporal no modelo proposto.

7.4.1.3 O relógio big-ben

Conforme registrado no início desta seção, o modelo proposto necessita de alguma estrutura que viabilize ao sistema incorporado (*embodied*) perceber a passagem do tempo de forma implícita, tal como os humanos a percebem, isto é, sem a constante necessidade de consultar um relógio, qualquer que seja ele: de pulso, de parede, etc.

Dessa forma, o modelo de mundo proposto possui um componente denominado relógio de intervalo, batizado de big-ben[79]. Este componente é, na realidade, composto por uma família de sensores lógicos (seção 7.4.1.1Pág.237) com várias bases de referência de tempo com as seguintes granularidades:

- Base: década – utilizada para demarcar a história com intervalos de logo prazo;

[79] Como referência ao relógio de Londres que por muito tempo foi utilizado como referência em nossa sociedade.

- Base: anual – utilizada para demarcar eventos de médio prazo;
- Base: mensal – utilizada para demarcar eventos recentes;
- Base: diária – utilizada para demarcar eventos muito recentes;
- Base: horária – utilizada para demarcar eventos de duração muito curta;
- Base: minuto – utilizada para demarcar eventos de hardware.

Como todos os sub-relógios utilizam a mesma estrutura de representação, na realidade eles podem ser conceitualmente implementados na forma de estruturas superpostas, conforme apresentado na Figura 55.

```
Var
  Bigben           : array [0.. (a+me+d+h+mi+s) ] of bits;
  Relogio_ano      : array[0..a] of bits absolute bigben[ano_atual];
  Relogio_mês      : array[0..me] of bits absolute relogio_ano[mês_atual];
  Relogio_dia      : array[0..d] of bits absolute relogio_mês[dia_atual];
  Relogio_hora     : array[0..h] of bits absolute relogio_dia[hora_atual];
  Relogio_minuto   : array[0..m] of bits absolute relogio_hora[minuto_atual];
  Relogio_segundo : array[0..s] of bits absolute relogio_minuto[segundo_atual];
```

Figura 55 - Especificação do relógio de intervalo big-ben.

Utilizando-se a semântica de funcionamento dos sensores lógicos, o big-ben implementa a base de referência temporal necessária para o funcionamento do sistema como um todo, resultando em uma estrutura que viabiliza a implementação de uma lógica temporal simples[80] da seguinte forma:

- Como cada sensor lógico está sempre associado a uma base de referência,
  - O símbolo □ significa sempre se avaliada a raiz da árvore. Porém,
    - pode significar sempre se todos os nodos da sub-árvore considerada estiverem marcados;
    - pode significar nunca se todos os nodos da sub-árvore considerada estiverem desmarcados;
  - O símbolo ◊ significa eventualmente se somente alguns nodos da sub-árvore considerada estiverem ligados e o número de nodos ligados for menor que o número total de nodos daquele nível.

Uma característica importante do big-ben é que, mesmo que um "usuário" deseje alterar a data do sistema para uma hora passada ou futura, o mecanismo de registro da passagem de tempo continua o inalterado, isto é, a função de conversão dos *ticks* do relógio de tempo real do hardware continua sempre avançando para frente e demarcando

[80] Conforme Bittencourt (1998), a principal característica de uma lógica temporal é o fato de uma determinada fórmula lógica poder apresentar valores verdade distintos em instantes diferentes do tempo.

a passagem do tempo a partir da última marcação. Isto garante integridade dos eventos históricos e, portanto, a integridade dos fatos historicamente assimilados pelo sistema.

7.4.1.4 O conceito de tempo implícito

A partir desta regra de integridade temporal, tomando-se o big-ben como base de referência de tempo de fluxo contínuo para todas as atividades realizadas pelo sistema, é possível implementar o conceito de tempo implícito, ou seja, uma unidade de tempo que cada plano, em particular, utiliza para "saber" se está atrasado ou adiantado em relação ao tempo de execuções anteriores. Isto é, cada plano de execução, sendo parte integrante do "organismo", tem de perceber os efeitos da execução de outras atividades paralelas (outros planos) em cada unidade de tempo do relógio de intervalo da máquina. A forma como isto pode ser percebido dá-se justamente pela constatação de que, em execuções anteriores, determinado ponto da lógica do plano foi executada mais ou menos rapidamente do que a execução atual[81]. Um exemplo de funcionamento do conceito de tempo implícito será apresentado na seção 7.4.3$_{pág.259}$.

7.4.2 Modelo de Mundo

> *"Recentemente tem havido um acúmulo de evidências no sentido de que, para exibir um comportamento inteligente, sistemas artificiais devem ter acesso ao ambiente que os cerca. Parece não ser mais suficiente possuir elaboradas capacidades de raciocínio, aprendizado e planejamento. Um sistema inteligente deve ser capaz de autonomamente adquirir as informações necessárias através da percepção e executar as ações contempladas." Schaad (1998).*

Nesta seção o modelo de mundo proposto é detalhado. No contexto do presente trabalho e doravante denominado de **MM**, ele é definido a partir das seguintes dimensões:

- A dimensão física:
  - Modelo Arquitetônico;
- A dimensão lógica:
  - Modelo de Dispositivos Físicos
  - Modelo de Dispositivo Lógicos;
  - Modelo de Aplicação.

[81] Este conceito, se implementado no modelo atual, inviabilizaria qualquer solução computacional, visto que, via de regra, o conceito de tempo é implementado através da ativação de chamadas da primitiva **getTime()** (ou similar) e pela diferenciação entre valores numéricos para constatar-se o atraso ($t_{atual} > t_{anterior}$), adiantamento ($t_{atual} < t_{anterior}$) ou estabilidade ($t_{atual} = t_{anterior}$).

A seguir, cada uma das dimensões acima é detalhada.

7.4.2.1 A dimensão física

Segundo Einstein (pág.14, 2001),

> *" a descrição do lugar onde ocorreu um evento ou onde se encontra um objeto se baseia em indicarmos o ponto de um corpo rígido (corpo de referência) com o qual o evento coincide. Este procedimento se aplica não apenas à descrição científica, mas também para a vida diária. Nas aplicações práticas, aquelas paredes rígidas que formam o sistema de coordenadas quase nunca existem na realidade; também as coordenadas não são determinadas através de construções com estacas rígidas, mas sim, indiretamente. Porém, o sentido físico das especificações de posição sempre deve ser buscado conforme as discussões precedentes, se não quisermos que os resultados da física e da astronomia se percam na imprecisão. Chegamos assim ao seguinte resultado:* ***toda descrição de eventos no espaço necessita de um corpo rígido (sistema de referência) com o qual os eventos são espacialmente relacionados****. Toda e qualquer relação pressupõe que, para os "segmentos de reta", valem as leis da geometria euclidiana, sendo o "segmento" representado por duas marcas sobre um corpo rígido Toda descrição do lugar (ou posição) onde ocorreu um evento ou onde se encontra um objeto se baseia em indicarmos o ponto de um corpo rígido (ou corpo de referência) com o qual aquele evento coincide. Na verdade há dois aspectos que devem ser considerados em favor da teoria da relatividade: Mesmo que a mecânica clássica não possa fornecer uma base suficientemente ampla para a apresentação teórica de todos os fenômenos físicos, deve-se no entanto atribuir-lhe uma parcela muito importante da verdade; porque ela fornece com muita precisão os movimentos reais dos corpos celestes. A priori, é pouquíssimo provável que um princípio de tão grande generalidade, que se aplica com tamanha exatidão a um tipo de fenômeno, venha a falhar em um outro domínio ".*

A Dimensão Física corresponde ao conceito de estrutura material apresentada em Costa (1993), isto é, "a coleção de componentes materiais da máquina e suas interconexões". Em outras palavras, é um construto teórico que serve como corpo de referência e que permite descrever o local onde os eventos acontecem. A dimensão física viabiliza o conceito de incorporação (*embodiment*) do sistema à sua representação física no mundo real.

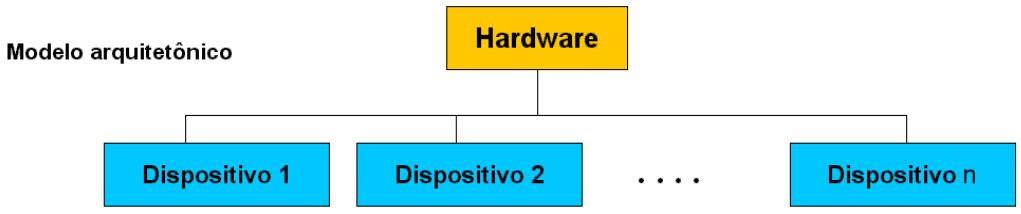


Figura 56 – Dimensão física do Modelo de Mundo proposto

A dimensão física é composta basicamente pelo modelo arquitetônico (um diagrama de classes - Figura 56). Este modelo descreve os componentes físicos em termos de suas características, interconexões e relacionamentos permitindo uma

descrição formal sobre o que constitui a parte física do sistema. O objetivo deste componente do modelo de mundo é permitir ao sistema saber o que ele é, do que é composto e como suas partes funcionam.

Além disso, a existência da dimensão física vai permitir que:

- Tarefas (ou sub-tarefas) associadas a recursos físicos não disponíveis por algum motivo não sejam executadas ou sejam tomadas providencias imediatas quanto à indisponibilidade do recurso;
- A descrição de configuração atual e do status dos dispositivos sejam mapeados para uma palavra de status, a qual, sendo OR-ed com os requisitos da tarefa, determina o (ou um dos) estado(s) de execução.

Se esta dimensão está associada ao modelo de execução, toda a vez que o monitor for "dormir", todo acesso ao vídeo é desabilitado, economizando energia e recursos computacionais. Dessa forma:

  - As aplicações instantaneamente devem saber que o monitor está em descanso, o que lhes permite adaptarem-se a esta realidade;
  - Quando ocorre o evento monitor acordado, dispara-se a tarefa de atualizar o monitor.

Esta dimensão está de acordo também com a noção de computação como regulação (COSTA,1993,pág.81), no sentido de que, "um processo computacional é regulativo se e somente se o efeito das interações que o constituem é o de manter a informação existente no objeto que ele constrói o mais próximo possível de um marco referencial, estabelecido pela máquina ou pelo ambiente". Neste contexto, um plano pode ser considerado com o um elemento regulador de uma computação, em sentido mais amplo do que o de simplesmente ordenar operações: "o programa de uma computação é a estrutura que regula a informação contida no objeto construído por aquela computação".

7.4.2.2 A dimensão lógica

A Dimensão Lógica está relacionada aos aspectos que tem por finalidade implementar, através de estruturas computacionais (linguagens de programação e estruturas de dados), a conexão entre o modelo físico e o modelo que exporta a abstração de uma entidade que possui comportamento inteligente, isto é, que permite a implementação do conceito de suporte operacional conforme apresentado em Costa (1993,pág.246).

7.4.2.2.1 *Modelo de Dispositivos Físicos*

O Modelo de Dispositivos Físicos envolve os sensores lógicos que obtém informações sobre o hardware e os atuadores que nele operam. Esta dimensão corresponde, grosseiramente, ao nível de inicialização dos *device drivers* nos modelos tradicionais. Uma alteração no estado físico de qualquer dispositivo imediatamente afeta o mundo como um todo e, portanto, todas as entidades que nele ocorrem.

Neste sentido, os sensores deste nível detectam, basicamente, a presença ou não e a disponibilidade ou não de determinado recurso físico, caracterizando, desta forma, o contexto físico do dispositivo (**CFD**). Cabe destacar que o conceito de presença ou ausência tem conotação semântica, porque o estado associado ao nível físico pode auxiliar no processo de inferência de alto nível e indicar características de forma, peso e possibilidades de ação e interação do sistema com o meio externo.

Um CFD é descrito através dos seguintes estados (Figura 57):

- Não existe;
- Desligado – se existe, mas está efetivamente desligado;
- Inicializando – existe e está sendo inicializado;
- Disponível – foi inicializado e está pronto para ser utilizado;
- Não disponível – existe, mas, por algum motivo, não pode ser utilizado neste momento.

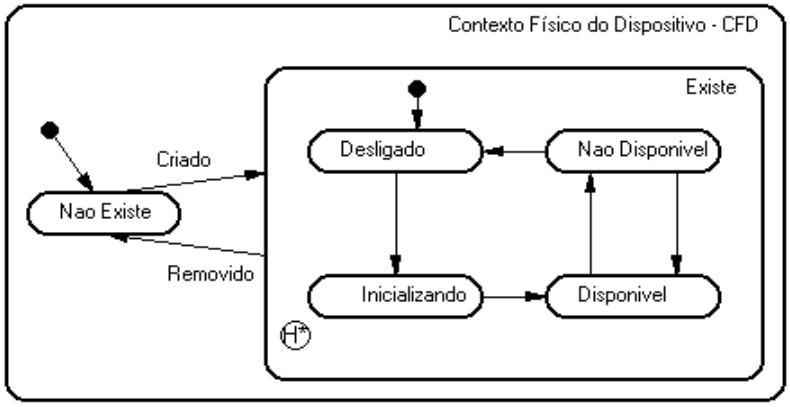


Figura 57– Estados do Contexto Físico do Dispositivo

A junção de todos os **CFD** determina o contexto físico do mundo (**CFM**). Esta afirmação pode ser representada da seguinte forma:

$$CFM = f(CFD_1, CFD_2, CFD_3, CFD_n)$$

A função que agrega os valores de cada **CFD** consiste, basicamente, de uma operação de concatenação dos vários **CFDs** individuais. Assim, o **CFM** pode ser definido da seguinte forma:

$$\mathbf{CFM = CFD_1 \oplus CFD_2 \oplus \ldots \oplus CFD_n}$$

A Figura 58 apresenta a configuração de um subespaço de possibilidades para um **MM** contendo 5 dispositivos (o modelo foi produzido a partir de um protótipo de software para validação do conceito). Como cada **CFD** é representada por um sensor lógico, toda alteração no estado deste dispositivo imediatamente altera o valor de **CFM**. Sendo o **CFM** uma "variável global *read-only*" para o **MM**, qualquer alteração é imediatamente refletida no **MM** como um todo.

996 PHY [nao disponivel,nao disponivel,disponivel,desligado,nao disponivel]
997 PHY [nao disponivel,nao disponivel,disponivel,inicializando,desligado]
998 PHY [nao disponivel,nao disponivel,disponivel,inicializando,inicializando]
999 PHY [nao disponivel,nao disponivel,disponivel,inicializando,disponivel]
1000 PHY [nao disponivel,nao disponivel,disponivel,inicializando,nao disponivel]
1001 PHY [nao disponivel,nao disponivel,disponivel,disponivel,desligado]
1002 PHY [nao disponivel,nao disponivel,disponivel,disponivel,inicializando]
1003 PHY [nao disponivel,nao disponivel,disponivel,disponivel,disponivel]
1004 PHY [nao disponivel,nao disponivel,disponivel,disponivel,nao disponivel]
1005 PHY [nao disponivel,nao disponivel,disponivel,nao disponivel,desligado]
1006 PHY [nao disponivel,nao disponivel,disponivel,nao disponivel,inicializando]
1007 PHY [nao disponivel,nao disponivel,disponivel,nao disponivel,disponivel]
1008 PHY [nao disponivel,nao disponivel,disponivel,nao disponivel,nao disponivel]
1009 PHY [nao disponivel,nao disponivel,nao disponivel,desligado,desligado]
1010 PHY [nao disponivel,nao disponivel,nao disponivel,desligado,inicializando]
1011 PHY [nao disponivel,nao disponivel,nao disponivel,desligado,disponivel]
1012 PHY [nao disponivel,nao disponivel,nao disponivel,desligado,nao disponivel]
1013 PHY [nao disponivel,nao disponivel,nao disponivel,inicializando,desligado]
1014 PHY [nao disponivel,nao disponivel,nao disponivel,inicializando,inicializando]
1015 PHY [nao disponivel,nao disponivel,nao disponivel,inicializando,disponivel]
1016 PHY [nao disponivel,nao disponivel,nao disponivel,inicializando,nao disponivel]
1017 PHY [nao disponivel,nao disponivel,nao disponivel,disponivel,desligado]
1018 PHY [nao disponivel,nao disponivel,nao disponivel,disponivel,inicializando]
1019 PHY [nao disponivel,nao disponivel,nao disponivel,disponivel,disponivel]
1020 PHY [nao disponivel,nao disponivel,nao disponivel,disponivel,nao disponivel]
1021 PHY [nao disponivel,nao disponivel,nao disponivel,nao disponivel,desligado]
1022 PHY [nao disponivel,nao disponivel,nao disponivel,nao disponivel,inicializando]
1023 PHY [nao disponivel,nao disponivel,nao disponivel,nao disponivel,disponivel]
1024 PHY [nao disponivel,nao disponivel,nao disponivel,nao disponivel,nao disponivel]

Figura 58- Exemplo de representação de Modelo Físico do Mundo

A possibilidade de sentir instantaneamente qualquer alteração no nível de **CFM** indica que o **MM** deve sustentar a representação de comportamentos para cada uma das possibilidades de variação de cada **CFD** individual.

7.4.2.2.2 *Modelo de Dispositivo Lógicos*

Uma vez que o contexto físico do mundo (**CFM**) descreve a disponibilidade física de cada recurso declarado no Modelo Arquitetônico, o próximo nível refere-se ao contexto lógico do dispositivo (**CLD**).

Esta dimensão está associada à lógica de controle dos dispositivos físicos, indicando a disponibilidade dos dispositivos lógicos em função das respectivas

flutuações de carga - espaço em disco, espaço em memória, banda de transmissão. Esta dimensão corresponde ao nível da lógica de controle presente nos *device drivers* dos modelos tradicionais. Alterações nesta dimensão imediatamente afetam o mundo como um todo e, portanto, todas as entidades que nele se verificam. É neste nível que são implementados os serviços lógicos que podem ser executados sobre os dispositivos físicos.

Um **CLD** pode ser descrito a partir dos seguintes estados:

- Não disponível – estado correspondente ao do modelo **CFD**;
- NegativeBig – baixa utilização, pouco espaço ocupado, baixa taxa de transferência;
- NegativeSmall – baixa-média utilização do dispositivo;
- Zero – estado intermediário de recursos lógicos do dispositivo;
- PositiveSmall – média-alta utilização do dispositivo;
- PositiveBig – alta utilização, muito espaço ocupado, alta taxa de transferência.

A Figura 59 apresenta o relacionamento dos **CLDs** através de uma máquina de estados.

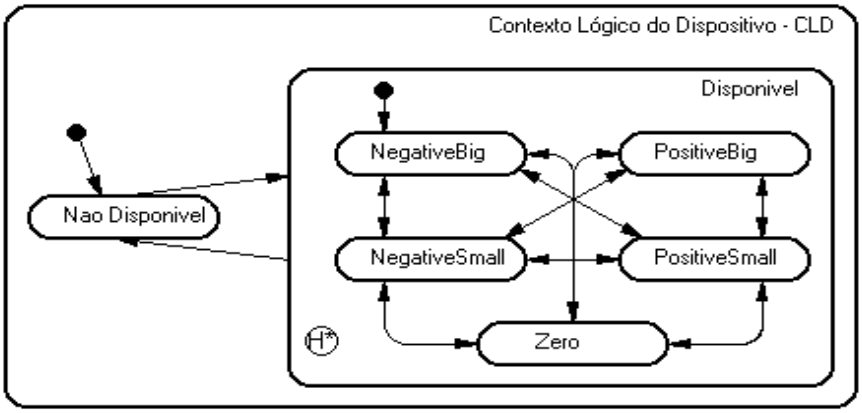


Figura 59 - Estados do Contexto Lógico do Dispositivo

Cabe destacar que foi adotada a mesma terminologia utilizada nos exemplos de variáveis lingüísticas de lógica difusa, tendo em vista caracterizar um comportamento genérico deste modelo. Ainda mais porque a granularidade da leitura dos sensores lógicos, que é ajustável, permite uma maior ou menor precisão na qualidade da informação apresentada.. De alguma forma, esta "imprecisão" controlada permite uma analogia com o conceito de variáveis lingüísticas da lógica difusa.

Por outro lado, a referência aos estados pode ser feita através de *aliases* conforme o exemplo apresentado na Tabela 27.

Neste nível os sensores e atuadores também são componentes lógicos. Diferentemente dos modelos tradicionais, o **CLD** faz parte do mundo e é afetado por ele. Assim, sua lógica de funcionamento é implementada segundo o modelo de aplicação que será apresentado na seção 7.4.3Pág259.

Tabela 27 - Aliases para variáveis lingüísticas

Estados	Aliases p/disco	Aliases p/rede	Aliases p/ memória	Aliases p/CPU
Negative big	Disco vazio	Sem colisões	Memória disponível	Idle
Negative small	Disco pouco ocupado	Baixa taxa de colisões	Baixa ocupação	Pouco usada
Zero	Disco mais ou menos ocupado	Media taxa de colisões	Media ocupação	Media utilização
Positive small	Disco mais ou menos cheio	Media/alta taxa de colisões	Media alta ocupação	Media alta utilização
Positive big	Disco cheio	Alta taxa de colisões	Muito ocupada	Alta utilização

A junção de todos os **CLD** determina o contexto lógico do mundo (**CLM**). Esta afirmação pode ser representada da seguinte forma:

$$\mathbf{CLM = f(CLD_1, CLD_2, CLD_3, CLD_n)}$$

A função que agrega os valores de cada **CLD** consiste, basicamente, de uma operação de concatenação dos vários **CLDs** individuais. Portanto, o **CLM** pode ser definido da seguinte forma:

$$\mathbf{CLM = CLD_1 \oplus CLD_2 \oplus \ldots \oplus CLD_n}$$

Da mesma forma, uma consulta ao **CLM** a partir dos sensores de estado de cada um dos **CLDs** descritos no Modelo de Dispositivo Físico, permite uma visão instantânea do estado lógico do sistema, ou seja, da sua disponibilidade dos recursos.

Como cada **CLD** é representado por um sensor lógico, toda alteração no estado de um dispositivo lógico imediatamente altera o **CLM**. Sendo um **CLD** uma "variável global *read-only*" para o **MM**, qualquer alteração é imediatamente refletido no **MM** como um todo. Da mesma forma, a possibilidade de sentir instantaneamente qualquer

alteração no nível de **CLM** indica que o **MM** deve sustentar uma representação de comportamento para cada possibilidade de variação de cada **CLD** individual.

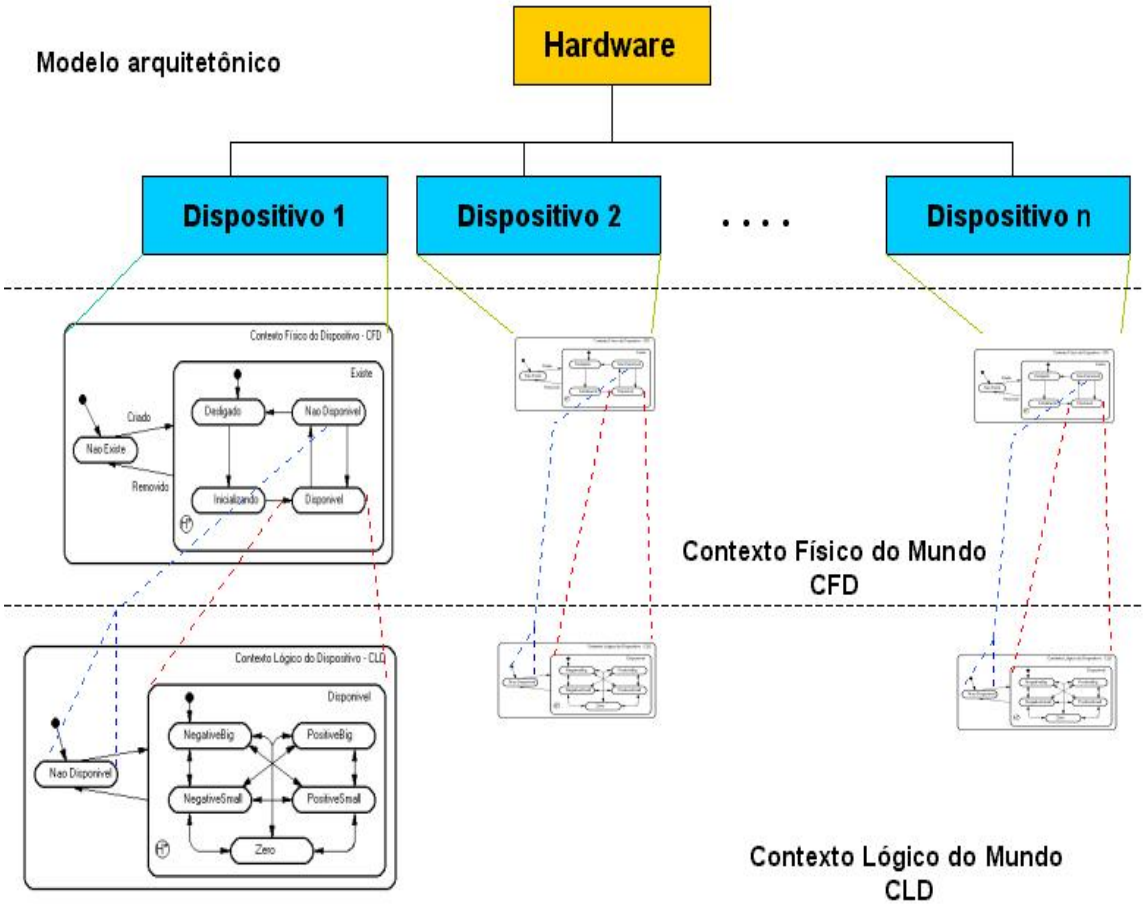


Figura 60 – Contexto Lógico do Mundo - CLM

Deve ser destacado que o contexto lógico do dispositivo não indica o que ele está fazendo. O objetivo desta informação de baixo nível é indicar instantaneamente o estado lógico de cada periférico. Por exemplo, um controlador de rede, neste nível, deve indicar o estado da rede em termos de variáveis lingüísticas como: ótimo, bom, regular, péssimo; e não em termos de taxa de transferência, número de pacotes e número de colisões[82].

A Figura 61 apresenta a configuração de um subespaço de possibilidades para um **MM** contendo 5 dispositivos (o modelo foi produzido a partir de um protótipo de software para validação do conceito).

A partir das considerações acima, surge a questão acerca da forma de tradução deste modelo conceitual em uma implementação computacional.

[82] Embora estas informações sejam úteis, o use delas dificulta uma reação imediata a alterações ambientais, tendo em vista que há a necessidade de interpretação a cada momento, o que demanda tempo e recursos computacionais.

Figura 61– Exemplo de representação do Contexto Lógico do Mundo

O modelo de mundo proposto, embora conceitualmente hiperdimensional, pode ter sua dimensionalidade reduzida para uma estrutura unidimensional conforme apresentado na Figura 62.

Esta figura descreve o mapeamento entre os sensores lógicos para um padrão de bits, o qual é tomado em sua totalidade como um deslocamento em uma tabela de uma dimensão. Como esta tabela contempla todas as possibilidades de combinações de estados do mundo, ela certamente teria um tamanho intratável para os padrões de programação do modelo atual. Além disso, haveria provavelmente um grande número de entradas vazias para as quais não foi previsto antecipadamente um comportamento adequado.

Contudo, utilizando-se os recursos de um hardware **MMU** (gerenciador de memória virtual) ou associando-se uma estrutura de *extendible-hashing*, não se faz necessária a prévia inicialização da tabela toda, mas somente daqueles estados de mundo que serão utilizados, o que se sabe de antemão À medida que o sistema vai sendo utilizado, novos estados vão sendo endereçados e, para os mapeados para um endereço não inicializado, o hardware da **MMU** causará uma interrupção de *page-fault*. Este é o indicativo de que o sistema derivou para uma situação desconhecida. A partir deste indicativo, o sistema chaveia-se para um modo de aprendizagem onde ele tentará aprender a tratar aquela nova situação a partir de sua base de planos.

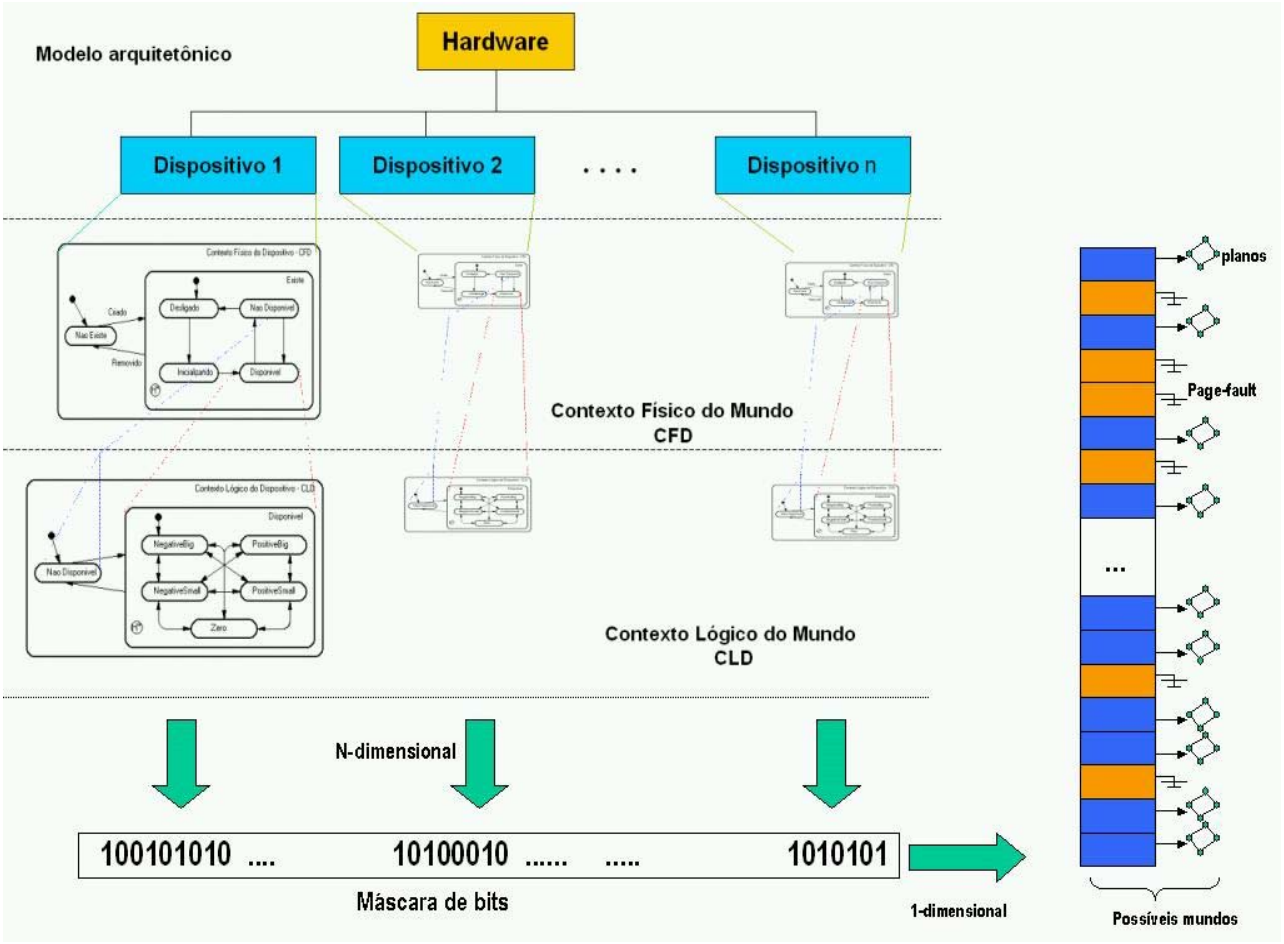


Figura 62 - Transformação do modelo hiperdimensional em modelo unidimensional.

Por outro lado, o uso de uma estrutura de *extendible-hashing* permite que o próprio sistema controle a granularidade do fator de agrupamento (clusterização), dependendo do contexto do mundo como um todo, ou seja, considerando também as tarefas em execução naquele momento.

7.4.3 Modelo de Aplicação

Envolve a especificação do modelo de aplicação que será executada no modelo de sistema proposto.

Um modelo de mundo não é suficiente para tornar um sistema inteligente. Obrigar que as aplicações explicitamente considerem a adequação ao ambiente, também não é suficiente. Isto porque tanto o sistema como as aplicações permanecem "cegas" às ações que estão sendo executadas pelas demais aplicações em operação[83].

[83] Ver a questão da virtualização do hardware na página 30.

Conforme apresentado em Mattos e Pacheco (2002), grande parte do esforço destinado ao detalhamento de um projeto de software, já considerando a fase de projeto, envolve a especificação do comportamento dinâmico da aplicação. A UML apresenta uma série de diagramas destinados a este fim (ERIKSSON e PENKER,1998). Contudo, apesar desta fase demandar muita energia para sua realização, uma vez concluída o resultado do trabalho passa a fazer parte da documentação do projeto e geralmente é arquivado (Figura 63a).

Independentemente de se estar usando ferramentas CASE, que geram código automaticamente ou não, o código fonte produzido é submetido ao compilador que traduz as especificações para um padrão binário compatível com o processador alvo denominado formato de arquivo executável (.COM,.EXE,.PE, .COFF,.a.out,etc).

O modelo proposto estabelece as seguintes alterações no processo de desenvolvimento de aplicações:

- O modelo dinâmico deve ser expresso em um modelo de representação de planos, conforme definido no trabalho de Schaad (seção 6.6 pág 205);
- Cada nodo componente do plano deve ser descrito segundo um modelo DEVS (atômico ou composto) seguindo a notação Insitu;
- O modelo dinâmico, expresso em planos de execução, descreve um meta espaço conceitual da aplicação, de tal forma que a execução da aplicação é direcionada pelo meta-modelo dinâmico baseado numa máquina de estados DEVS;
- Sobre o modelo dinâmico são aplicadas as regras do relógio de intervalo, de tal forma que o modelo dinâmico deve perceber instantaneamente a passagem do tempo por ele medida e considerar as variações de deadline estabelecidas pela aplicação;

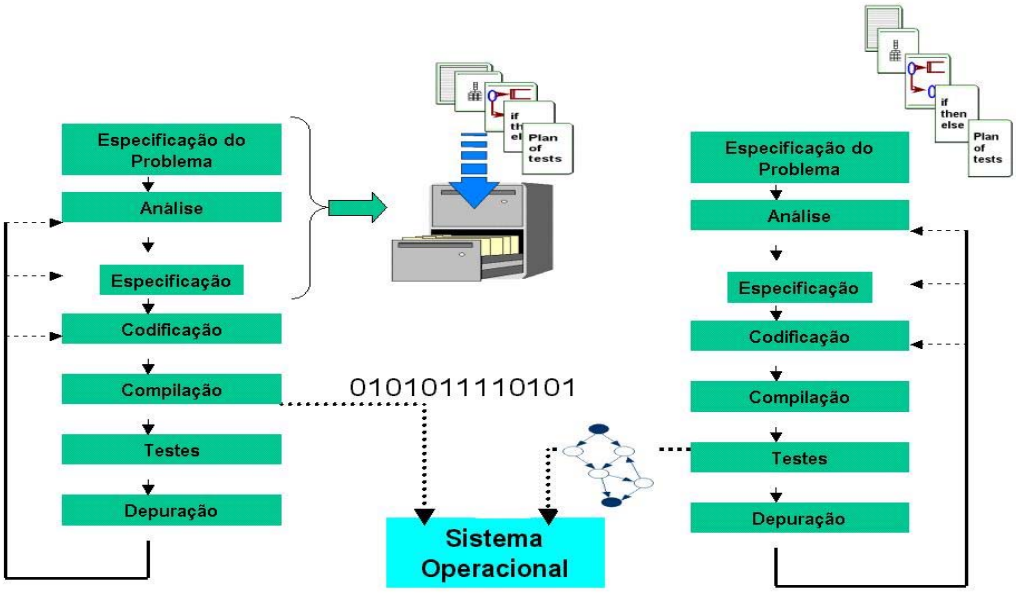


Figura 63 – Processo de Desenvolvimento de Aplicações (a) tradicional (b) proposto.

O conjunto de modelos dinâmicos deve ser integrado em um todo executável na forma de planos, segundo especificado em Schaad (1998).

O modelo proposto, ao utilizar planos em lugar de código binário (Figura 63b), permite que o resultado da fase de testes seja utilizada como conhecimento pelo sistema operacional acerca, por exemplo, daqueles caminhos da solução que foram "mais testados", "menos testados", ou mesmo que não foram testados porque, por algum motivo, os dados utilizados não exercitaram todas as possibilidades. A conseqüência imediata deste modelo é que a fase de testes passa a ter uma importância maior, contribuindo sobremaneira tanto para a melhoria na qualidade dos artefatos de software, como para que o sistema operacional, de posse do modelo de execução, possa tomar providências quando um ramo de uma árvore de decisões é escalonado para execução, mas está marcado como "não tendo sido testado".

Além disso, a partir deste modelo dinâmico é possível ao sistema identificar o que uma determinada aplicação está fazendo. Utilizando técnicas de interação mais adequadas, é possível informar ao usuário, de uma forma também mais adequada, o que está ocorrendo no mundo em determinado momento. Em outras palavras, é possível perguntar ao sistema: "o que você esta fazendo agora?" e obter uma resposta mais razoável do que uma lista de processos em execução.

A Figura 64 apresenta o modelo geral de desenvolvimento de aplicações. A estratégia utiliza uma interface conversacional para adquirir as informações necessárias à montagem de um perfil de caso que solucione o problema em foco.

A partir deste perfil, um módulo de Raciocínio Baseado em Casos acessa uma base de planos já desenvolvidos e recupera os mais adequados ao contexto do problema. O melhor perfil de caso é selecionado e o sistema reconfigurado para orientar a interação com o desenvolvedor de forma produzir/adequar/adaptar os planos existentes ou não, caso se trate de alguma estratégia de solução ainda não empregada.

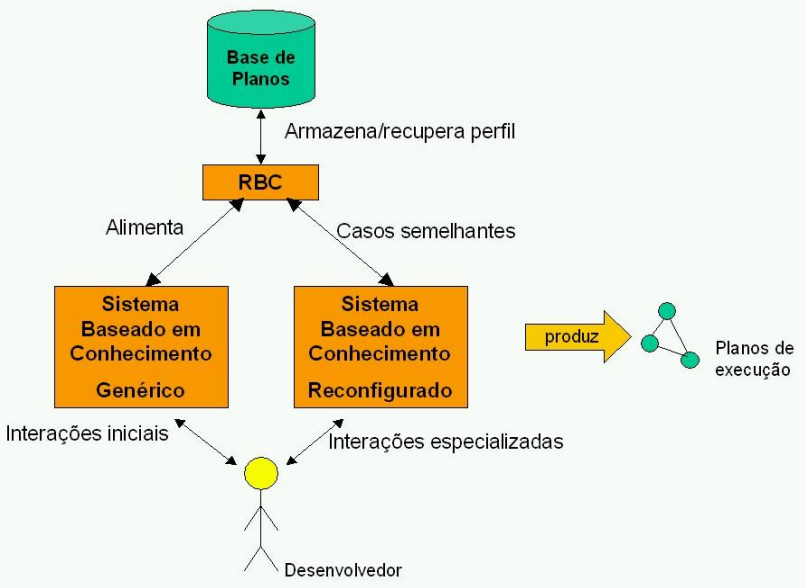


Figura 64 – Modelo de desenvolvimento de aplicações

A base de planos contém não somente os planos anteriormente desenvolvidos, mas as práticas conhecidas em termos de padrões (*design patterns*) de projeto, as quais vão sendo dinamicamente selecionadas à medida que o sistema reconfigurado vai sendo utilizado e o contexto da aplicação delineado. O resultado deste processo é uma especificação em alto nível, a qual é compilada na forma de planos de execução, segundo especificado em Schaad (1998).

7.4.3.1 Validação do modelo de desenvolvimento proposto

Tendo em vista validar o modelo de desenvolvimento apresentado na seção anterior, foram desenvolvidos alguns protótipos em software para avaliar a forma de interação e a viabilidade da utilização do modelo no desenvolvimento de aplicações. O

cenário escolhido foi o de ensino de lógica de programação a partir dos seguintes aspectos:

- A especificação de um enunciado de problemas de lógica de programação assemelha-se, respeitadas as proporções, a uma especificação de sistema (ou módulo, ou programa, ou componente, ou qualquer denominação que designe algum artefato de software a ser produzido);
- O processo de identificação dos requisitos necessários ao desenvolvimento de um algoritmo assemelha-se ao processo de análise de requisitos de um sistema, novamente respeitadas as devidas proporções;
- Uma boa análise de requisitos de problemas de lógica de programação conduz, via de regra, à solução dos problemas propostos. As boas práticas da Engenharia de Software estabelecem que uma boa análise de requisitos é fator determinante para o sucesso de um projeto de software;
- A solução de problemas de lógica de programação conduz ao desenvolvimento de um pseudocódigo que, posteriormente, pode ser convertido diretamente para uma linguagem de programação ou para uma ferramenta de diagramação a partir da qual podem ser realizadas animações e simulações da execução do código produzido. A premissa é verdadeira também para o escopo de desenvolvimento de sistemas;
- O pseudocódigo produzido pode ser submetido a ferramentas de verificação formal, a fim de validar a especificação em função dos requisitos identificados na fase de análise de requisitos.

Descrições dos protótipos foram relatadas em Mattos, Fernandes e López (1999), Gubler (2002), Heinzen (2002), e em Mattos(2002).

Basicamente, o modelo utiliza uma estrutura baseada em árvores de decisões que guia a interação com o aluno, induzindo-o a descobrir o caminho em direção à solução, enquanto o módulo de raciocínio baseado em casos generaliza um enunciado extraindo as informações necessárias à identificação, na base de casos, daqueles que podem reconfigurar a base de regras do sistema, eliminando as não pertinentes ao contexto da aplicação em desenvolvimento.

No caso específico do problema de ensino de lógica de programação, observa-se com freqüência que, a partir da solução de um determinado número de exercícios, alguns alunos "descobrem" o mecanismo de como construir as soluções de lógica de programação. Entretanto, outros, enquanto não se apropriam desta nova forma de pensar, permanecem estagnados e assumem uma postura de tentativa-e-erro. Naturalmente, isto conduz ao insucesso e ao desânimo, levando muitas vezes à reprovação na disciplina. Este ato de descobrir, se analisado do ponto de vista do aluno, poderia ser considerado

como um conhecimento tácito[84], uma vez que, mesmo aqueles que aprenderam como fazer, têm dificuldades de explicar aos colegas como organizaram o raciocínio que os levou à solução do problema. Por outro lado, analisado-se do ponto de vista dos professores, o conhecimento é procedimental, ou seja, sabe-se que há uma seqüência de passos a serem executados no sentido de construir-se uma solução lógica. Naturalmente, há ainda um outro tipo de conhecimento envolvido no caso em questão, o qual pode ser classificado como declarativo – ou seja, analisando-se a solução, verificar-se se a mesma funciona ou não, baseando-se apenas em experiência prévia. Por exemplo: sabe-se que determinada seqüência de passos, para determinados problemas, provavelmente conduzirá a uma solução incorreta para determinadas situações.

Uma característica fundamental do protótipo construído é que o conhecimento do especialista é armazenado através de regras organizadas na forma de árvores de decisão. Regras, neste contexto, são construções simples na forma: Se alguma condição é verdadeira, Então faça alguma coisa. O uso de regras é interessante em algumas situações, tendo em vista que permite a construção da definição de um determinado comportamento de maneira compacta. Analisado em sua forma mais simples, um determinado comportamento caracteriza-se por um conjunto de ações e um conjunto de condições sob as quais as ações devem acontecer.

O ponto de partida da análise foi: de um lado, o enunciado do problema; de outro, o esboço da solução para cada um dos problemas conforme caracterizado na Figura 94[85] pág.372. Uma vez estabelecidos os limites, passou-se a responder às questões e analisar-se o relacionamento entre as respostas e a proposta de esboço de solução, no sentido de identificar-se padrões de comportamento. Este procedimento foi realizado para um conjunto de aproximadamente 30 exemplos, os quais englobavam desde a solução de problemas simples (entrada-equação-saída), problemas de complexidade média baixa (entrada-várias equações-saída), problemas de média complexidade (entrada-decisões-equações-saída) e problemas de complexidade média-alta (repetições-entrada-decisões-equações-saída). Com base nestas informações, o passo seguinte foi a construção uma

[84] Segundo Giarratano e Riley (1994), o conhecimento pode ser classificado em: (i)Procedimental: saber como fazer – ex: como esquentar água; (ii) Declarativo: saber que algo é verdadeiro/falso – ex: não tocar em chaleira quente e (iii)Tácito ou 'inconsciente': sabe o que ocorre, mas não sabe como faz – ex: mover a mão.

[85] Apêndice 8 – Planilha para identificação de regras.

estrutura de representação do conhecimento identificado através da análise dos problemas resolvidos, o que será descrito a seguir.

7.4.3.2 A representação do conhecimento

Durante o processo de análise do conhecimento representado na planilha, verificou-se que havia questões que exigiam respostas do tipo sim/não; questões que exigiam respostas textuais; situações em que era necessária uma orientação ao aluno no sentido de guiá-lo ao passo seguinte e, finalmente, situações onde se fazia necessário uma realimentação sobre as decisões tomadas anteriormente, a fim de posicioná-lo no contexto da solução em andamento.

A partir destas informações, construiu-se uma árvore de decisões[86], a qual possui 4 tipos de nodos, a seguir relacionados:

- Nodos de decisão – ex: "Há alguma operação lógica ou aritmética?";
- Nodos de ação – ex: "Descreva a operação";
- Nodos de status – ex: "Até o momento você identificou os seguintes passos":
- Nodos de ajuda – ex: "Uma vez analisadas as pré-condições, podemos identificar a operação a ser realizada!".

A Figura 92[87] pág.370, apresenta a configuração desta estrutura, que representa o conhecimento do especialista em termos de conhecimento procedural. A estratégia adotada para orientar o aluno foi a filosofia de desenvolvimento *top-down*, onde o desenvolvimento da aplicação dá-se por refinamentos sucessivos. Sempre que houver necessidade, um novo passo de refinamento vai sendo realizado até a obtenção do nível desejado de especificação que efetivamente solucione o problema.

Para tanto, foram construídas regras que permitem a navegação através dos nodos da árvore de decisões. Assim sendo, à medida que os nodos vão sendo repetidamente visitados, novos fatos vão sendo gerados na memória de trabalho. Estes novos fatos disparam regras que, executadas, geram uma nova árvore auxiliar (denominada árvore de *log*), a qual registra as respostas do usuário e estabelece a seqüência temporal em que as respostas vão sendo cadastradas.

---

[86] A primeira versão foi construída em Clips pelo autor e posteriormente convertida para Delphi no trabalho de conclusão de curso de Gubler(2002).

[87] Apêndice 6 – Modelo de árvore de decisões para algoritmos.

Em momentos estratégicos, apresenta-se ao aluno um feedback do contexto por ele delineado até o momento e, dependendo do nível de refinamento necessário, automaticamente o sistema inicia um novo refinamento de alguma estrutura que ainda requeira informações complementares. A Figura 65 apresenta uma tela onde é solicitado ao aluno que descreva sinteticamente o problema a ser resolvido. Neste momento já se procura induzi-lo a pensar em termos do contexto do problema. Repetidamente vão sendo apresentadas questões ao aluno para que ele vá, aos poucos, delimitando o escopo do problema e aprofundando-se na busca de uma solução para o mesmo.

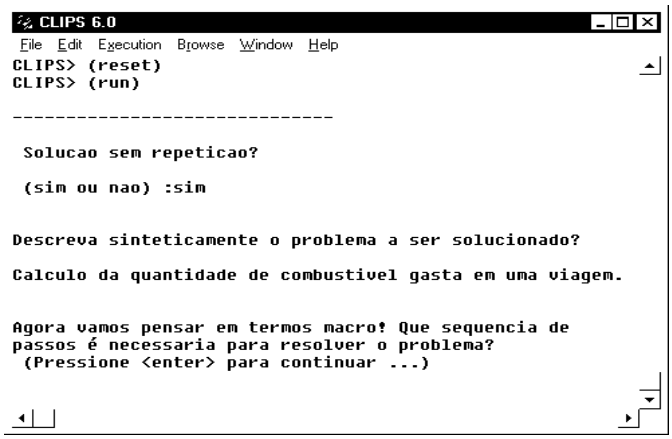


Figura 65 – Contextualização do problema

Após cada rodada na árvore de decisões, apresenta-se ao aluno um feedback do contexto por ele delineado até o momento e, dependendo do nível de refinamento necessário, automaticamente o sistema inicia um novo refinamento de alguma estrutura que ainda requeira informações complementares (Figura 66).

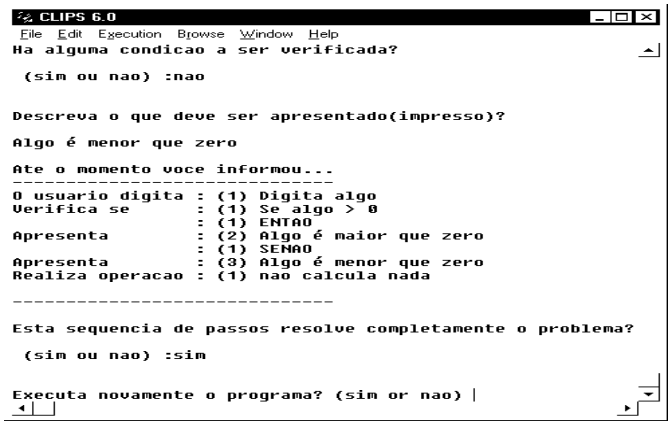


Figura 66 –Feedback ao aluno

A Figura 67 apresenta uma versão do protótipo convertida de *Clips* para *Delphi* descrita em Gubler (2002).

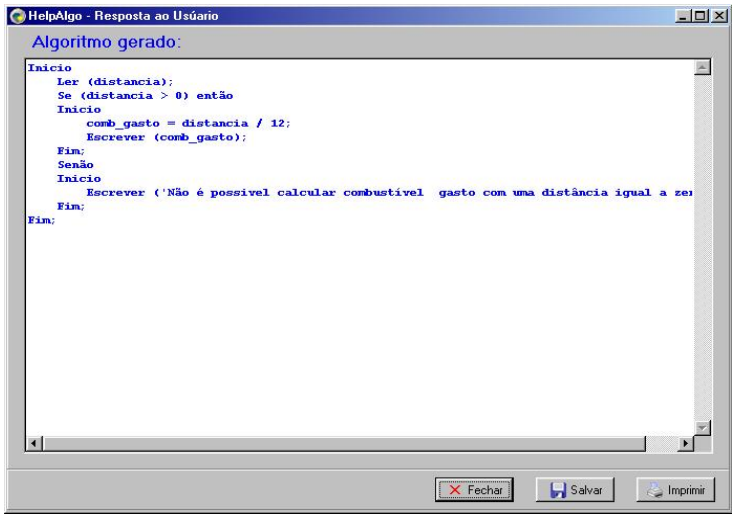


Figura 67 – Pseudo-código resultante da execução do sistema

A adoção desta estratégia permitiu que fosse adquirido o conhecimento tácito do aluno, a partir do conhecimento declarativo, ou seja, o aluno sabe responder sim e não às perguntas que vão sendo realizadas. A ordem das perguntas está estabelecida na árvore de decisões. O que o aluno não sabe fazer que é explicar como cada passo para resolver o problema vai sendo armazenado na árvore auxiliar (*log*), de tal forma que, ao final do processo, é possível inferir a solução global, através da conjugação dos vários tipos de conhecimento envolvidos no processo.

Um outro aspecto destacado na Figura 64 é o emprego de técnicas de raciocínio baseado em casos, o qual é delineado a seguir.

7.4.3.3 O uso da técnica de raciocínio baseado em casos.

A área de raciocínio baseado em casos é relativamente recente. Contudo, os pesquisadores em RBC não divergem muito sobre sua definição. A maioria dos autores concorda que o RBC é um método de raciocínio baseado na proposta de utilizar experiências passadas, encapsuladas em estruturas de dados, como base para lidar com novas situações similares.

A abordagem parece ser intuitiva: quando uma nova situação acontece, deve-se tentar alguma coisa que já foi utilizada com sucesso. De um modo geral, as soluções

utilizadas em situações similares devem ajudar na solução do novo problema. É importante notar que o raciocínio pode funcionar também por contra-exemplos. Neste caso, dada uma nova situação, deve-se procurar descartar aquelas soluções utilizadas em situações similares e que resultaram em insucessos.

Os sistemas RBC utilizam um processo interativo constituído genericamente por: identificação da situação atual, busca da experiência mais semelhante na memória e aplicação do conhecimento desta experiência na situação atual. Entretanto, a literatura usualmente não considera a identificação da situação atual como parte do processo RBC, adotando um modelo genérico baseado em quatro etapas: recuperar, reutilizar, revisar e reter.

A principal parte do conhecimento nos sistemas RBC está representada através de seus casos. Um caso pode ser entendido como a abstração de uma experiência descrita em termos de seu conteúdo e contexto, podendo assumir diferentes formas de representação. A representação dos casos é uma tarefa complexa e importante para o sucesso do sistema RBC. O conhecimento, neste contexto, pode ser interpretado como sendo um conjunto de métodos que modelam um conhecimento especializado para disponibilizá-lo em um sistema inteligente.

Apesar de um sistema RBC ser extremamente dependente da estrutura e do conteúdo de sua base de casos, seu conhecimento não está presente apenas nos seus casos (memória), mas também nas suas etapas de desenvolvimento. Estas etapas são:

- Representação dos casos: Um caso neste contexto é definido através de um perfil de solução, o qual é formado pelos seguintes componentes: Palavras-Chave, Generalização, Número-de-passos, Número-de-variáveis, Número-de-constantes e Solução. As palavras-chave são obtidas tomando-se por base o texto do enunciado e a aplicação de um conjunto de heurísticas importadas da solução anterior. A generalização constitui-se no resultado do processo de abstração da solução do caso em questão. Número-de-passos, Número-de-variáveis e Número-de-constantes são métricas utilizadas para melhor definir o perfil deste caso e usadas na fase de seleção dos melhores casos para um determinado contexto. A Solução contém a seqüência detalhada dos passos necessários para a solução do problema. Esta solução é desenvolvida utilizando-se a metodologia descrita anteriormente;
- Indexação: A indexação é um problema central em raciocínio baseado em casos, envolvendo a determinação dos tipos de índices que serão utilizados pela etapa de recuperação. Da mesma forma que índices podem acelerar a busca em bases de dados, eles são utilizados em RBC para acelerar a recuperação de casos. No protótipo construído há basicamente 2 índices que são utilizados para nortear o processo de recuperação: um índice montado a partir das generalizações e um índice montado a partir das palavras-chave. Cada entrada no índice de generalização aponta para o conjunto de casos que descrevem os perfis de soluções associadas. Cada entrada no índice de palavras-chave aponta para o conjunto de

generalizações em cujas soluções está a referida chave. Como este é um protótipo de validação da idéia, e a base de casos não é muito grande, não há a necessidade de uma estrutura de armazenamento mais elaborada. Certamente esta será uma questão a ser melhor considerada quando da implementação real do modelo completo;

- Recuperação dos casos: Dado o enunciado de um problema a ser resolvido, esta etapa realiza a busca na base de casos e seleciona os que podem ser aproveitados. O processo é feito por algoritmos que selecionam casos da base que apresentam um determinado grau de similaridade com o proposto. No contexto do protótipo, este processo ocorre em duas fases: uma fase de pré-seleção, dirigido por palavras-chave, e uma fase de seleção propriamente dita, dirigida por abstrações. Na fase de pré-seleção, o processo de montagem do caso proposto envolve uma fase de análise do texto do enunciado, onde se busca identificar, através de uma série de heurísticas, um conjunto de palavras-chave que nortearão o processo de busca dos possíveis casos enquadrados neste perfil. O índice de similaridade é calculado a partir do número de palavras-chave encontradas. Quanto maior o número delas, melhor a qualidade da seleção. A fase de seleção propriamente dita, onde será selecionado o caso que melhor se enquadra à solução, ocorre da seguinte forma: uma vez recuperados os casos julgados pertinentes, dispara-se o processo de análise de requisitos descrita em Mattos, Fernandes e López (1999). A medida em que o aluno vai desenvolvendo a lógica da solução, o conjunto de casos selecionado vai sendo depurado e um perfil genérico vai sendo montado, de tal forma que, a partir de um determinado momento, o melhor caso passa a ser usado como referência para nortear as questões que devem ser apresentadas ao aluno;
- Adaptação: Quando problemas são resolvidos usando-se RBC, o tipo principal de conhecimento usado durante a solução é aquele fornecido pelo conhecimento específico contido nos casos. Contudo, em muitas situações este conhecimento não é suficiente ou apropriado para atender a todos os requisitos da aplicação. Freqüentemente são utilizados conhecimentos de domínio, os quais podem contribuir para a identificação de dependências entre certas características de um caso e auxiliar na inferência de características adicionais. No caso do protótipo, as regras de adaptação são usadas para modificar a estrutura de uma solução apresentada pelo aluno. Por exemplo: a regra "seqüências if's podem ser escritas através de uma estrutura case", poder-se-ia enquadrar-se neste caso.
- Aprendizagem: Uma característica muito interessante que os sistemas de RBC podem apresentar é a capacidade de aprendizagem. Este processo pode ser aplicado a um caso específico ou à base toda. No protótipo, este processo é disparado em três situações: (i) toda vez que, após um enunciado ter sido digitado, o processo de seleção não consegue identificar nenhuma palavra-chave, ou o conjunto de palavras-chave recuperado é muito pequeno, ou ainda produz ambigüidades. Neste momento, dispara-se um processo onde o usuário, provavelmente o professor, deverá identificar quais são as palavras-chave importantes que vão nortear o processo da solução; (ii) num segundo momento, o processo de aprendizagem é disparado, quando o processo de recuperação encontra palavras-chave e não consegue recuperar um conjunto de casos. Nesta situação, o processo de aprendizagem consiste em generalizar a solução apresentada e cadastrá-la na base de casos; e (iii) num terceiro momento, dito off-line, quando se solicita explicitamente a execução do processo de análise e generalização dos casos da base.

A adoção desta estratégia permitiu que fosse adquirido o conhecimento tácito do aluno, a partir do conhecimento declarativo, ou seja, o aluno sabe responder sim e não às perguntas que vão sendo realizadas. A ordem das perguntas está estabelecida na árvore de decisões. O que ele não sabe fazer é explicar como cada passo para resolver o problema vai sendo armazenado na árvore auxiliar (*log*), de tal forma que, ao final do

processo, é possível inferir a solução global através da conjugação dos vários tipos de conhecimento envolvidos no processo.

7.4.4 O ambiente de execução

Um aspecto do modelo **InSitu** que precisa ser modificado é a estrutura de ativação dos planos a partir da base de planos inativos para o cachê de planos ativos. Isto porque a concepção da arquitetura **InSitu**, embora inovadora, também está baseada no modelo atual de construção de sistemas operacionais, ou seja, a estratégia de improvisação permite que o sistema adapte-se dinamicamente às situações novas. Porém o próprio sistema não aprende a partir delas. Daí a necessidade de um operador e do programador externos. Além disso, a estrutura de sub-planos, quando em execução, também segue o modelo de processo, apesar de implementar o conceito de multitarefa cooperativa e não preemptiva.

Deve-se destacar que a questão chave não é se o modelo preemptivo é melhor ou pior que o modelo cooperativo. A questão é que, em nenhum dos casos, as aplicações conseguem perceber a interferência das demais, tanto em termos de consumo de recursos como em termos de passagem do tempo.

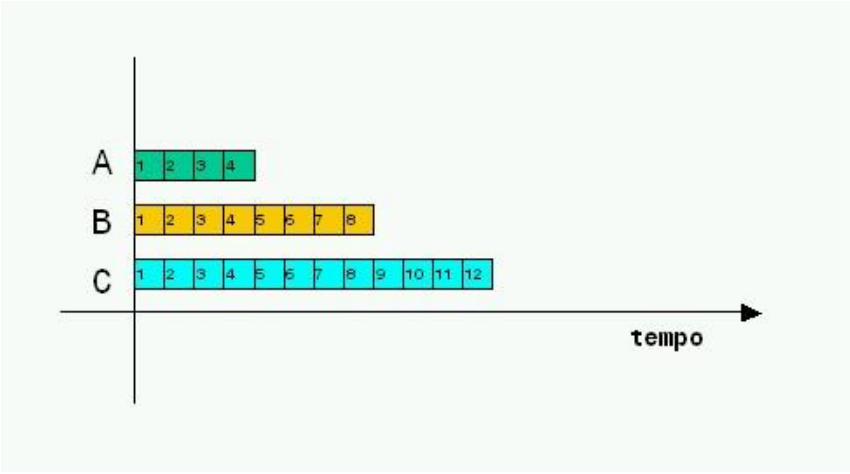


Figura 68 – Tempo nominal de execução das aplicações

Supondo a configuração inicial apresentada na Figura 68, onde A, B e C representam 3 programas a serem executados e t1,t2 e t3, respectivamente, a medida do tempo necessário para a execução de cada um deles. Para facilitar esta análise, supõe-se que todos o programas são CPU-Bound.

Conceitualmente, num ambiente multitarefa preemptiva tradicional, onde os 3 programas possuam a mesma prioridade, um dos possíveis caminhos de execução dos programas poderia ser aquele apresentado na Figura 69[88]. A cada marcação de t, ocorre uma troca de contexto.

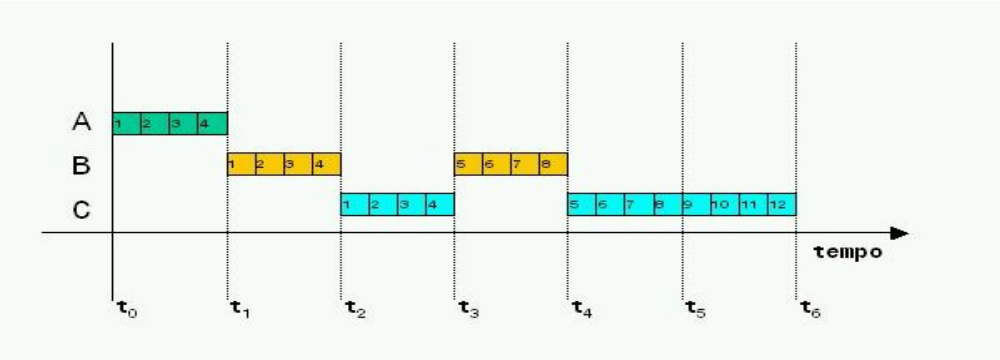


Figura 69 – Modelo preemptivo conceitual

Este é um modelo conceitual porque o que ocorre, na realidade, é que, entre as interrupções de relógio, há outras inúmeras possibilidades de interrupções do hardware que interferem na configuração final do modelo. Além disso, não está sendo considerado o tempo de troca de contexto, que não é constante.

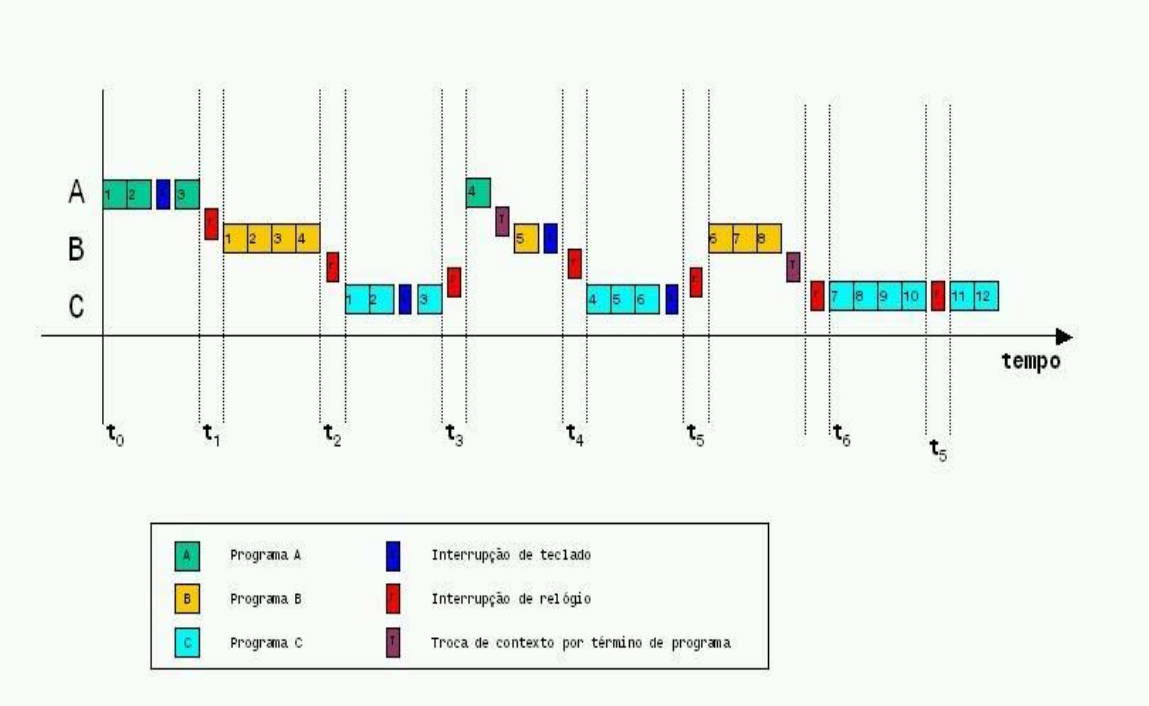


Figura 70 – Exemplo de modelo preemptivo real

[88] Reproduzida aqui para caracterizar o contexto da explicação.

A Figura 70 exemplifica uma situação real onde, embora os programas sejam CPU-bound, o sistema operacional é eminentemente IO-bound, ou seja, o sistema operacional tem que atender a todas as interrupções de hardware mesmo que, como no exemplo, as teclas pressionadas não sejam destinadas a algum dos programas em execução (o usuário poderia,por exemplo, estar "brincando" com o teclado).

A Figura 71[89] caracteriza como o substrato reativo é construído no paradigma atual.

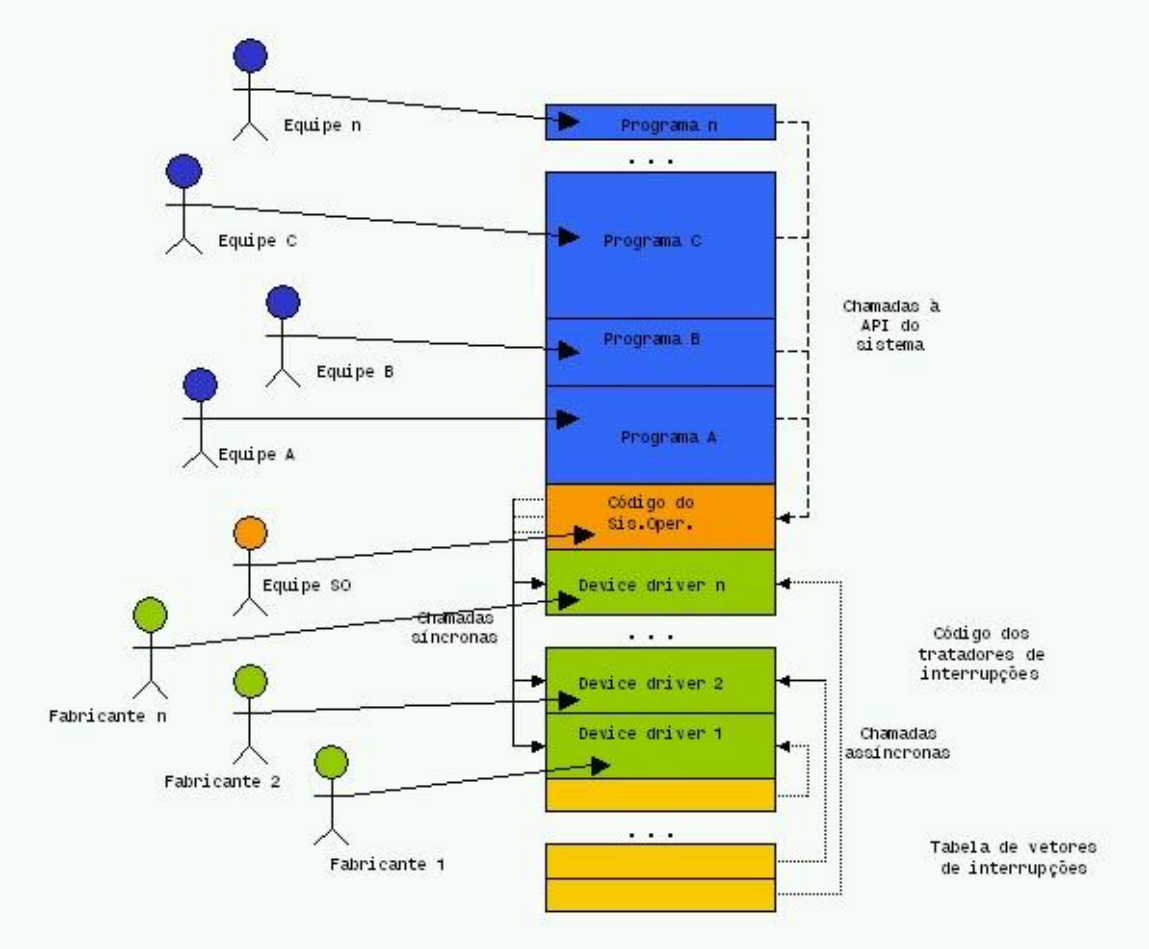


Figura 71 – Modelo de substrato reativo no paradigma atual

Como cada equipe é responsável pelo desenvolvimento de uma parcela específica do suporte operacional[90], não há como cada uma delas, individualmente, prever todas as possibilidades de combinações de eventos que ocorrem (e programas que estarão

[89] Reproduzida aqui para caracterizar o contexto.
[90] Segundo a premissa de *information hiding* (Eng.Software) a qual estabelece que se deve expor somente a interface dos módulos para que os programadores não tenham que preocupar-se com detalhes de implementação.

executando em determinado instante) no momento da concepção e construção de um módulo específico. A conseqüência imediata é relatada no capítulo 5(Pág.142).

Objetivando propor uma solução para esta questão é que surge a integração entre o formalismo DEVS e o modelo Insitu.

Conforme apresentado na seção 6.4, o principal aspecto que viabiliza a integração do modelo **InSitu** com o formalismo **DEVS** é que ambos estão baseados na passagem do tempo. Em **InSitu**, o intervalo de tempo é fator determinante para que o modelo execute todos os planos entre operações de *pooling* dos sensores. No modelo **DEVS**, o intervalo de tempo determina quais ações de simulação devem ser executadas "instantaneamente" antes que a função de avanço do relógio de simulação seja executada.

Como o modelo proposto não é um ambiente de simulação e, portanto, o relógio não pode ser avançado mecanicamente para antecipar eventos que ainda não ocorreram, o emprego do formalismo pode parecer antagônico.

Por outro lado, não se pretende que o modelo de sistema operacional baseado em conhecimento tenha que fazer *pooling* de sensores para estabelecer seu estado de mundo atual, exceto para determinados periféricos que assim o exijam.

Assim sendo, a solução proposta passa pelo reconhecimento dos seguintes aspectos:

- Via de regra, as aplicações são construídas a partir de seqüências de outras construções (atribuições, decisões, desvios, repetições, etc.);
- Estas construções podem ser mapeadas para construções *steppable* de forma direta;
- Os *steppables* compostos incorporam outros *steppables* mantendo a coerência lógica do algoritmo;
- O fato de que os dados também são *steppables* garante proteção de memória, visto que, do ponto de vista das aplicações, a memória deixa de ser endereçável diretamente – assim, *steppables* de outros planos não tem como sobrepor seus *steppables*;
- A característica anterior permite que 2 seqüências de *steppables* sejam executadas em paralelo sem que haja necessidade da implementação do conceito de pilha por processo – basta utilizar a construção *steppable parallel*;
- A construção *parallel* suporta qualquer quantidade de *steppables* simples ou compostos;

- A máquina DEVS[91] pode ser implementada no núcleo dos *steppables*, de tal forma que a função ***step***, ao invés de simplesmente avançar para o próximo nodo da árvore ao término da execução do nodo atual, execute funções de transição interna, externa e confluente sempre que necessário.

A partir destas considerações tem-se a seguinte situação:

- Já que os planos são concebidos, via de regra, como seqüências de instruções; e
- A coerência lógica das instruções é garantida pelo ambiente de *run-time* **InSitu;** e
- A interrupção na seqüência, por algum tempo, não afeta o resultado final, exceto em situações com deadline explícito,

então:

- É possível agregar, através de um *steppable parallel*, vários sub-planos que precisam ser executados em paralelo em determinado momento; e
- Utilizando-se o formalismo DEVS, é possível fazer a carga de planos que precisam ser executados em paralelo e considerá-los como o '*bag'* de entrada para este formalismo.

Como a seqüência lógica determina qual é a próxima construção a ser executada, então, tomando-se as instruções como eventos de simulação, o modelo **DEVS** tem como saber de antemão qual é o próximo evento a ocorrer. Portanto, o passo de carga de uma aplicação no modelo atual transforma-se num processo de escalonamento de instruções para execução. Desse modo, os *steppables* passam a ser vistos como modelos atômicos ou *coupled* no formalismo **DEVS**.

Como o escalonamento é realizado ao nível de instrução e não mais ao nível de processo (ou *thread*), o intervalo de tempo utilizado é fator determinante para a coerência do modelo. Esta restrição caracteriza, portanto, a necessidade do relógio de intervalo, ou seja, um intervalo de tempo dentro do qual as operações executadas possam ser percebidas pela entidade inteligente como tendo sido executadas instantaneamente.

O relógio de intervalo permite também que seja estabelecida e conhecida a capacidade de trabalho da máquina não mais em *megaflops* ou *mips*, mas em unidades de trabalho *steppable*. Sabendo isto, a entidade inteligente poderá antecipar situações, pois terá "consciência" de seu rendimento.

[91] Durante a apresentação do trabalho: Dynamic models as Knowledge in Operating Systems Kernels. no evento AIS'2002 houve a oportunidade de discutir o modelo proposto com o Prof. Ziegler, pai do formalismo DEVS. O Prof. Ziegler achou a proposta de utilização do modelo DEVS como run-time para um sistema operacional muito interessante e apresentou-a em uma mesa redonda onde eram discutidos aspectos de padronização das várias versões do modelo por ele definido.

A partir destas considerações, a próxima seção descreve os passos necessários para a integração dos modelos DEVS e Insitu.

7.4.4.1 O ambiente de execução: DEVS + Insitu

Os principais conceitos do formalismo DEVS foram apresentados na seção 6.4$_{\text{pág.198.}}$ e os principais aspectos da arquitetura da especificação do modelo Insitu (SCHAAD,1998) foram apresentadas na seção 6.6$_{\text{(pág.205).}}$ Nesta seção são apresentadas as especificações necessárias para viabilizar a união dos dois modelos em um que preserve as principais características individuais de cada um deles.

Para tanto, as seguintes fases deverão ser superadas:

1. Conversão de cada objeto *steppable* do modelo **Insitu** para um correspondente objeto atômico ou *coupled* do modelo DEVS;
2. Mapeamento do método ***step*** da classe *steppable*, que controla o avanço para o próximo nodo, para a função de transição interna (δ_{int}), de acordo com a configuração do modelo de mundo naquele instante;
3. Implementar o conceito de tempo implícito, através da configuração da função de *time advance* do modelo DEVS, para uma unidade de tempo tal que não seja extremamente pequena, a ponto de permitir a execução de um nodo de cada árvore de decisão de cada vez, mas que também não seja longo o suficiente para inviabilizar os ajustes na árvore, toda a vez que isto for exigido por alterações; esta função de *time advance* é que viabiliza a implementação do conceito de relógio de intervalo, visto que, durante um intervalo, todos os eventos executados são considerados como tendo ocorrido "simultaneamente";
4. Implementação da função de transição externa (δ_{ext}), que transfere os nodos da árvore que não puderam ser executados num intervalo de relógio de intervalo, marcando-os como "empurrados";
5. Implementação da função de transição de confluência (δ_{con}) para inserção de planos reativos que devem ser ativados em função da notificação de ocorrências no ambiente externo ou de suporte material que requerem atenção naquele momento.

Portanto, o modelo de execução passa a funcionar com a semântica do ambiente de simulação DEVS, porém com uma estrutura de improvisação descrita na especificação **Insitu**.

Sabendo-se que todo evento de hardware é mapeado para o modelo de mundo, a máquina **InSitu** deve ser adaptada com construções de acesso ao hardware, onde o sistema será executado, na forma de sensores (leitura à portas de E/S) e atuadores (escrita em portas de E/S)[92]. Das demais construções especificadas por Schaad devem ser eliminadas aquelas destinadas ao controle de movimentação do projeto do robô, permanecendo aquelas construções tradicionais da lógica de programação.

A partir da experiência na construção do protótipo de validação, é possível generalizar o modelo para o desenvolvimento de aplicações em geral, tomando-se por base a enorme quantidade de conhecimento procedural sobre como realizar o desenvolvimento de aplicações, disponível na forma de livros, métodos e metodologias de desenvolvimento de sistemas.

Assim sendo, neste ponto da especificação já é possível vislumbrar o contexto do suporte operacional que está sendo proposto:

- Há um novo modelo de aplicação representado na forma de planos de execução; e
- Os planos são desenvolvidos através de um ambiente que implementa uma interface interativa cujo objetivo é transformar o conhecimento do especialista em planos (Figura 72).

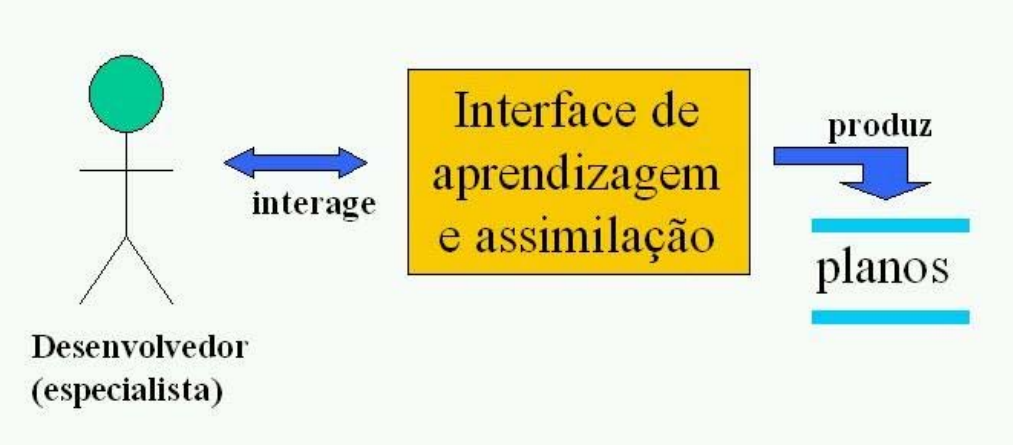


Figura 72 – Interação do especialista desenvolvedor.

[92] Cabe destacar que estas construções existem na especificação Insitu original, porém são específicas para o hardware do robô Rufus Firefly utilizado no trabalho de Schaad.

A Figura 72 caracteriza o fato de que não há mais o conceito de código executável, nem as operações tradicionais de instalação, ou a diferença entre *device driver* e aplicação: neste sistema "tudo são planos"[93].

Assim, os especialistas em cada área submetem (e/ou interagem) com a interface de aprendizagem, no sentido de "instruir o sistema sobre como resolver o problema".

Esta interface de aprendizagem, a partir de heurísticas próprias, transforma o produto das interações com o especialista em planos, os quais são adicionados à base de conhecimento do sistema. A própria interface de aprendizagem é escrita na forma de planos, conforme descrito no protótipo de validação apresentado anteriormente, o que torna o sistema homogêneo. A única estrutura a ser efetivamente implementada em código nativo do processador alvo é o código que implementa as árvores paralelas de decisão (Insitu) associadas ao modelo DEVS de execução (Figura 73).

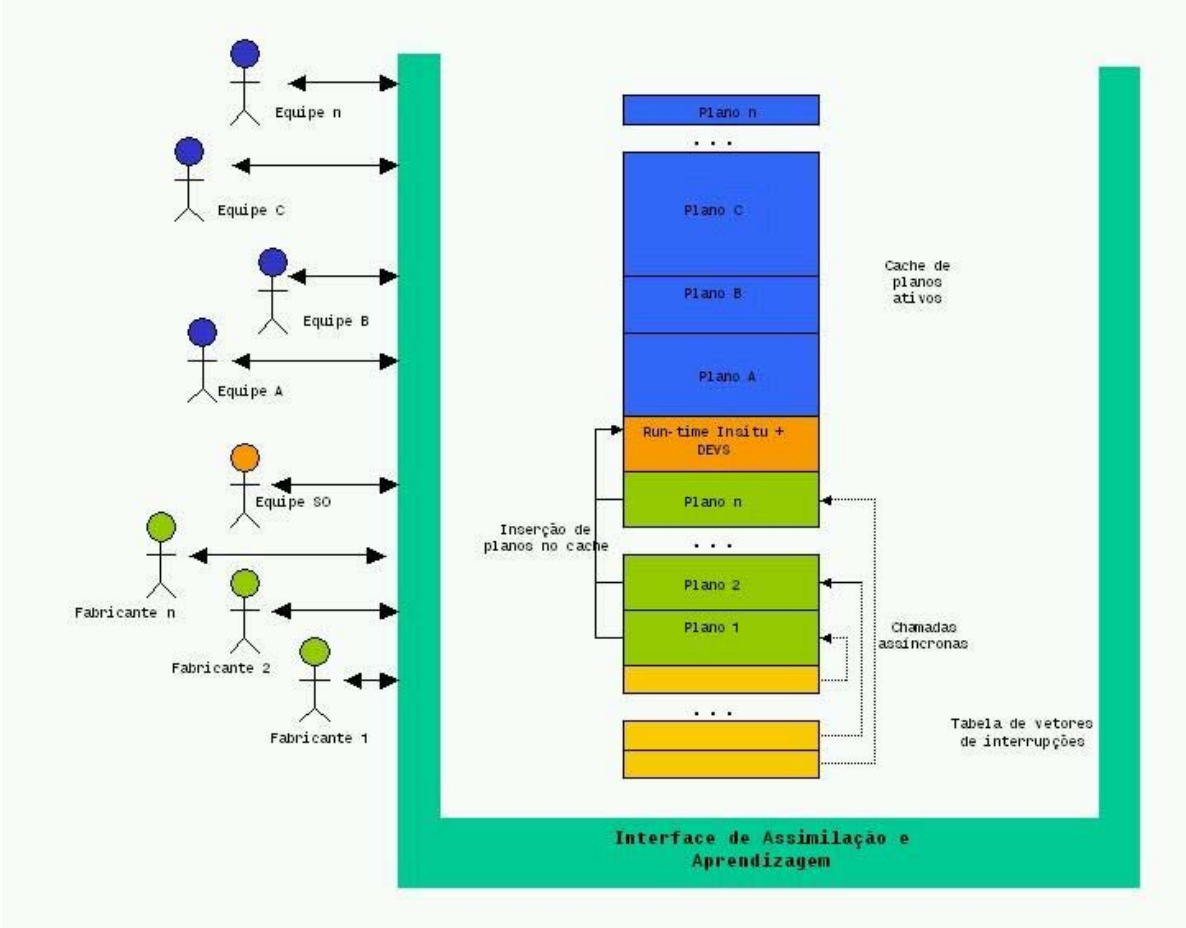


Figura 73 – Interface de assimilação e aprendizagem

[93] Contrapondo-se ao modelo utilizado no sistema Unix, onde "tudo são arquivos".

7.4.4.2 Meta modelo de representação de documentos

Um aspecto que demanda cada vez mais atenção das equipes de pesquisa refere-se ao volume de dados cada vez maior que é armazenado nas máquinas dos usuários e tornado disponível através da Internet. Neste sentido, a seção 4.3.2(pág.124) apresentou algumas considerações sobre a questão de visualização de grandes volumes de informações armazenadas.

Este é um aspecto normalmente ignorado nos projetos de sistemas operacionais, haja vista que é aparentemente inconcebível admitir um sistema operacional que não suporte a abstração de arquivo.

No paradigma atual, um arquivo é composto basicamente por um fluxo (*stream*) de bits, os quais são tornados persistentes através de técnicas de armazenamento em meio ótico ou magnético. A partir deste conceito, impõe-se a necessidade de algum método que permita o acesso a este fluxo de bits. Daí surge o conceito de sistema de arquivos.

Contudo, arquivo não é conhecimento. Basta considerar que se fez necessário o desenvolvimento de técnicas de gerenciamento de bases de dados e mineração de dados para viabilizar a "descoberta" de relacionamentos entre dados armazenados e a conseqüente transformação destes relacionamentos em informação útil, a qual também é armazenada na forma de arquivo.

Neste sentido, o modelo proposto contempla uma forma de assimilação do conteúdo dos tradicionais arquivos nos moldes da interface de aquisição descrita anteriormente. A Figura 74 caracteriza a situação proposta.

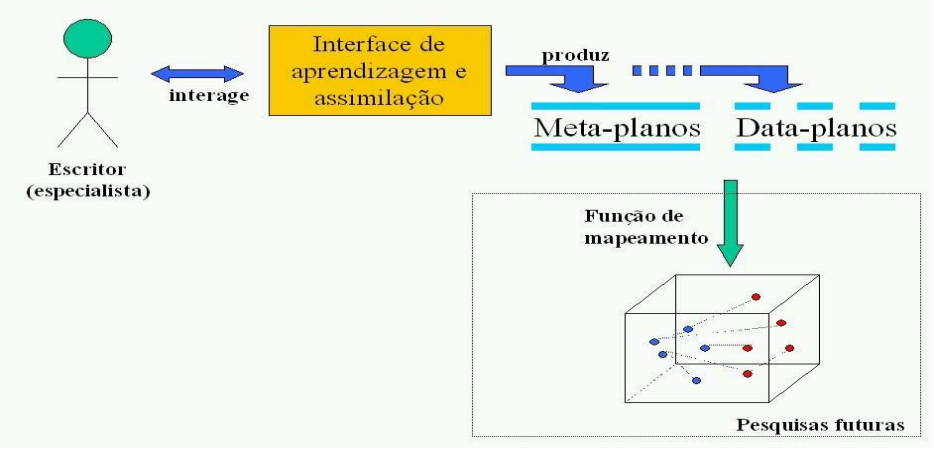


Figura 74 – Interação do especialista escritor

O processo de aquisição dos meta-planos conceituais pela interface de aprendizagem e assimilação, também é representado na forma de árvores de decisões. O conceito foi validado através de um protótipo em software desenvolvido pelo acadêmico Gilson Klotz[94]. O apêndice 10.7(pág.371) apresenta um extrato do modelo, cuja descrição completa pode ser obtido em Klotz (2002).

O protótipo descrito em Klotz(2002) utiliza a interface de aprendizagem para guiar o escritor no processo de desenvolvimento de uma redação. No final do processo de interação, um meta-plano descreve o "projeto da redação" que funciona como elemento de ligação entre os dados e o significado.

Deve-se destacar que, no atual estágio de especificação do modelo proposto, os dados são "armazenados" como ***steppables*** do tipo ***data***, o que foi denominado de data-planos. Isto remete, em algum grau, ao conceito de fluxo de bits discutido anteriormente e, portanto, novamente ao conceito de arquivo.

Contudo, o modelo arquitetônico completo prevê o mapeamento entre os meta-planos e os data-planos para representações equivalentes no espaço tri-dimensional. Até o presente momento, embora os resultados preliminares sejam promissores, eles ainda não são conclusivos, permanecendo, portanto, a indicação de uma direção de pesquisas para elucidar esta questão.

7.4.4.3 Integração entre modelo de aplicação e modelo de tempo

A forma como as aplicações são executadas em sistemas operacionais tradicionais foi caracterizada no capítulo 2 através da Figura 13(pág.57). Neste modelo, chamadas de procedimentos podem ser representadas conceitualmente da seguinte forma:

Call memória[endereço],

ou seja, independentemente do fato de o desenvolvedor ter utilizado técnicas avançadas de programação (OO, *Aspect*) ou ter simplesmente codificado uma seqüência de instruções sem conhecimento de aspectos relativos à qualidade e/ou organização do código, os compiladores, em última instância, traduzem a especificação lógica para um padrão de bits que é submetido ao sistema operacional para que ele escalone o tempo de

[94] Trabalho de conclusão de curso em Bacharel em Ciências da Computação realizado na FURB – Universidade Regional de Blumenau, sob orientação do autor do presente trabalho.

processador a fim de que este padrão de bits seja executado pelo processador (Conforme referido anteriormente, as unidades de tempo de processador são freqüentemente denominadas de *quantum* - base do conceito de multiprogramação). Nenhuma outra informação adicional é fornecida ao sistema operacional, como finalidade, tempo de execução, etc.

O modelo de aplicação do modelo proposto funciona de uma forma diferente. Neste caso, uma aplicação não tem como ser construída senão considerando o estado instantâneo do mundo. Esta afirmação pode ser representada conceitualmente da seguinte forma:

Call memória[endereço[Estado do Mundo,Prazo]]

O parâmetro **Estado do Mundo** é obtido através do mecanismo de redução de dimensionalidade apresentado anteriormente. Já o parâmetro **Prazo** é obtido da seguinte forma:

- Cada plano de execução possui associado uma estrutura do tipo sensor lógico, a qual serve como base de referência de tempo relativo de execução daquele plano;
- Cada *steppable* possui em sua estrutura um marcador que armazena somente o deslocamento dentro da estrutura de sensor lógico do tempo correspondente ao instante de sua execução. Por exemplo, se um steppable foi selecionado para execução 3 unidades de tempo após o disparo do plano, este marcador armazena o valor 3;
- Toda a vez que um *steppable* for escalonado para execução, ele toma o número do instante relativo de execução a partir da sua base de referência de tempo e o compara com o valor armazenado como tempo da última execução. Três situações podem ser detectadas nesta operação: (i) o valor armazenado no *steppable* é menor que o valor coletado da base de referência, o que implica que aquele *steppable* está atrasado em relação ao tempo da última execução; (ii) o valor armazenado no *steppable* é maior que o valor coletado da base de referência, o que implica que aquele steppable está adiantado em relação ao tempo da última execução; e (iii) o valor armazenado no *steppable* é igual ao valor coletado da base de referência, o que implica que aquele *steppable* está sendo executado no mesmo tempo da execução anterior – portanto está no prazo.

É justamente o parâmetro prazo que caracteriza o aspecto de validação da fase de testes caracterizado na Figura 63(pág.261), isto é, se uma aplicação não foi testada, os valores de Prazo não são inicializados. Portanto, ao ser selecionado para execução, e constatando-se que um prazo está indefinido, verifica-se imediatamente que aquele plano não foi testado. Por outro lado, para que todos os campos prazo tenham sido preenchidos, implica uma de duas situações:

- O plano foi devidamente testado e todos os possíveis caminhos foram verificados, o que é desejável, mas dificilmente constatado na realidade; ou
- o plano foi forjado, ou seja, os campos prazo foram, de alguma forma, inicializados com valores que não correspondem à realidade. A conseqüência disto é que provavelmente os tempos reais serão muito diferentes dos armazenados, o que fará com que o plano seja executado sob condições muito adversas em relação às ideais, podendo não atingir a meta planejada em função dos caminhos de exceção que venha a percorrer.

Ao substituir a forma tradicional de "chamada de procedimentos", introduz-se o conceito de adaptação já nas primeiras fases do processo de desenvolvimento de software. Isto implica que, ao desenvolver uma aplicação, além dos requisitos que tradicionalmente são considerados na área de Engenharia de Software, o aspecto de adequação ao ambiente passa a constituir-se como um requisito explícito.

Uma forma de considerar o conceito de tempo implícito no contexto de um plano é apresentada na Figura 75. Contudo, outras formas de referência ao tempo também devem ser contempladas de forma a permitir que tanto as aplicações quanto a máquina de inferência tenham um referencial de tempo para executarem suas ações.

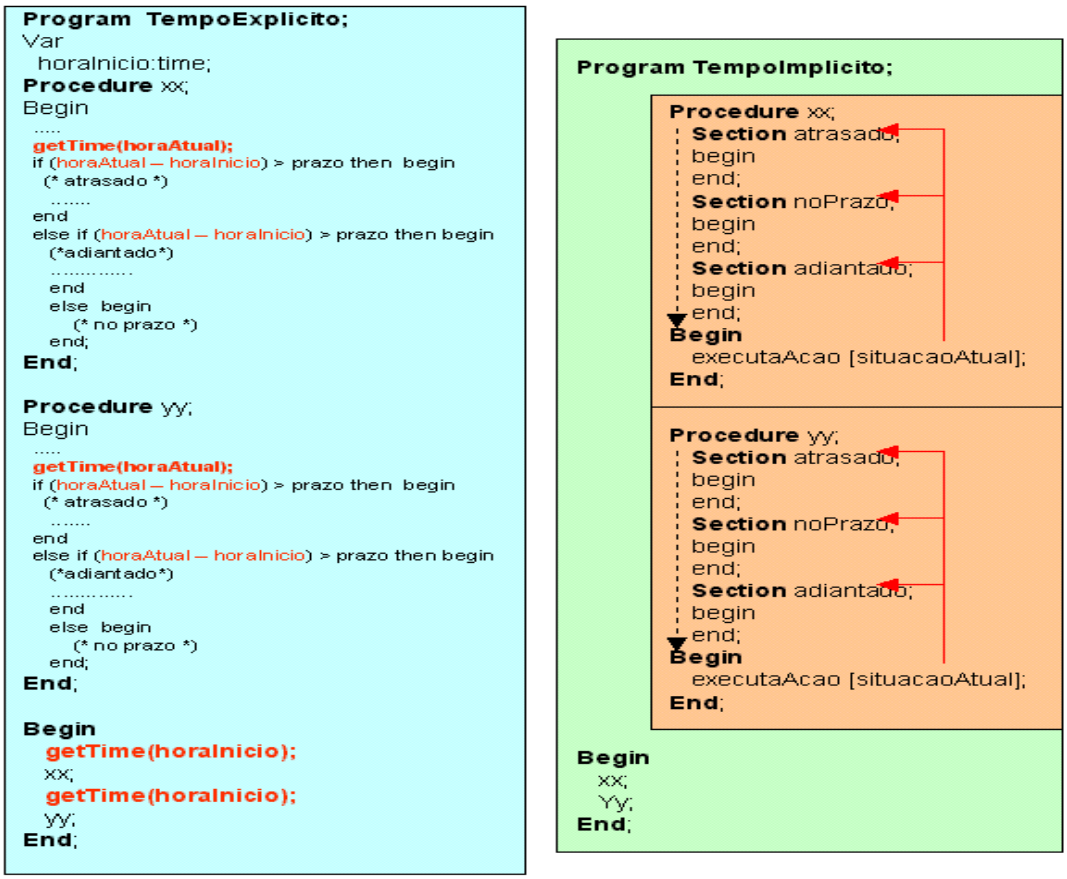


Figura 75 – Tratamento de tempo explicito x tempo implícito

Deve-se observar que, ao tratar implicitamente o tempo como um referencial para nortear a execução dos algoritmos, contribui-se para que uma das facetas do problema do frame[95] seja minimizada, já que a granularidade de sensibilidade dos programas, com relação ao tempo decorrido, passa a ser muito pequena, permitindo-lhes adaptarem-se dinamicamente durante sua execução. Contudo, esta solução implica também em que, a partir de agora, as aplicações tenham que considerar explicitamente as variações ambientais relativamente à passagem do tempo.

A obrigatoriedade de adequação ao ambiente é relativa. Uma aplicação pode escolher não se adequar às flutuações ambientais e declarar que não sabe o que fazer em determinada situação. Para tanto, deve informar ao sistema a sua intenção. De posse dessa informação, o sistema tem como utilizar políticas diferentes para, por exemplo, escalonar esta aplicação numa situação extrema. Poderiam ser consideradas as seguintes estratégias:

- A aplicação não é escalonada ou tem sua prioridade diminuída até que a flutuação termine;
- O usuário pode determinar que a aplicação deve continuar executando, em detrimento da flutuação na disponibilidade de recursos. Neste caso o sistema pode atuar em outras aplicações tendo em vista enfrentar a situação de forma mais adequada.

A declaração de que a aplicação não sabe como proceder em determinada situação permite ao sistema operacional tomar medidas preventivas, além de poder identificar explicitamente, sem ter que recorrer a *logs* de execução, em que ponto a aplicação falhou e qual era a situação do sistema naquele momento. De posse destas informações, o desenvolvedor tem melhores possibilidades de adequar a lógica da aplicação, o que contribui sobremaneira para a correção das soluções algorítmicas e a robustez das aplicações.

Portanto, a questão da percepção imediata destas situações faz com que as aplicações não tenham mais necessidade de ficar perguntando ao sistema operacional, sistematicamente, a hora absoluta e realizando operações aritméticas para detectar atrasos ou adiantamentos. Desta forma, o modelo de tempo permite que a aplicação situe-se em alguma das seguintes situações em tempo de execução:

---

[95] Refere-se à questão levantada por McCarthy e Hayes (1969) relativamente à representação de fatos que mudam enquanto algum tipo de inferência está sendo executada. Segundo Rich e Knight (1991,pág.151), "em alguns domínios, a única parte difícil é representar todos os fatos enquanto em outros, determinar que fatos mudam não é uma tarefa trivial".

- Atraso em relação a execuções anteriores;
- Adiantamento em relação a execuções anteriores;
- No prazo em relação a execuções anteriores.

7.4.4.4 Redução de dimensionalidade

Pode ocorrer, contudo, que alguma aplicação não indique explicitamente o que fazer em determinada situação. Neste caso, o sistema deve ser informado através de uma cláusula "**NÃO_SEI**" (Figura 76) nas seções onde não há código ou reação prevista antecipadamente. O sistema, ao deparar-se com esta cláusula, pode tomar uma das seguintes posições:

- Cancelar a execução daquela aplicação, indicando em que ponto do código e em que situação de mundo ocorreu o problema;
- Suspender a aplicação até que o mundo retorne a uma situação para a qual há tratamento previsto.

Assim, a aplicação torna-se responsável pela adaptação, conforme sugerido nas seções 2.4.5[96] (Pág.69) a 2.4.8[97] (Pág.74), sem necessidade de que o desenvolvedor tenha que conhecer detalhes do núcleo do sistema, como é o caso das propostas micronúcleo, exonúcleo e nanonúcleo discutidas anteriormente.

Esta estratégia, associada à utilização de *extendible hashing*, permite que subconjuntos de situações possam ser *clusterizadas*, fazendo com que seja reduzido significativamente o tamanho da tabela de possibilidades de mundo a serem efetivamente terão implementadas.

A detecção de uma situação desconhecida é efetivada imediatamente pela função de *hash*, que não retorna valor válido. Isto permite ao sistema chavear-se para um modo de operação denominado de modo de aprendizado. Neste, podem ser disparados algoritmos para analisar o mundo de uma forma menos reativa e mais reflexiva, certamente com o custo associado de consumo de recursos computacionais. Uma vez estabelecido o plano para atender esta situação anormal, é ele inserido na base de planos do sistema. Este, então, chaveia-se novamente para o modo reativo, liberando capacidade computacional para as necessidades do usuário.

---

[96] Estruturas Configuráveis ou Adaptáveis.
[97] Estruturas Baseadas em Nanonúcleos.

Supondo-se uma situação em particular onde o sistema não sabe o que fazer, é possível realizar uma fotografia instantânea do mundo, identificando qual a disponibilidade dos recursos no instante do erro, que planos estavam sendo executados e em que ponto deles ocorreu o problema.

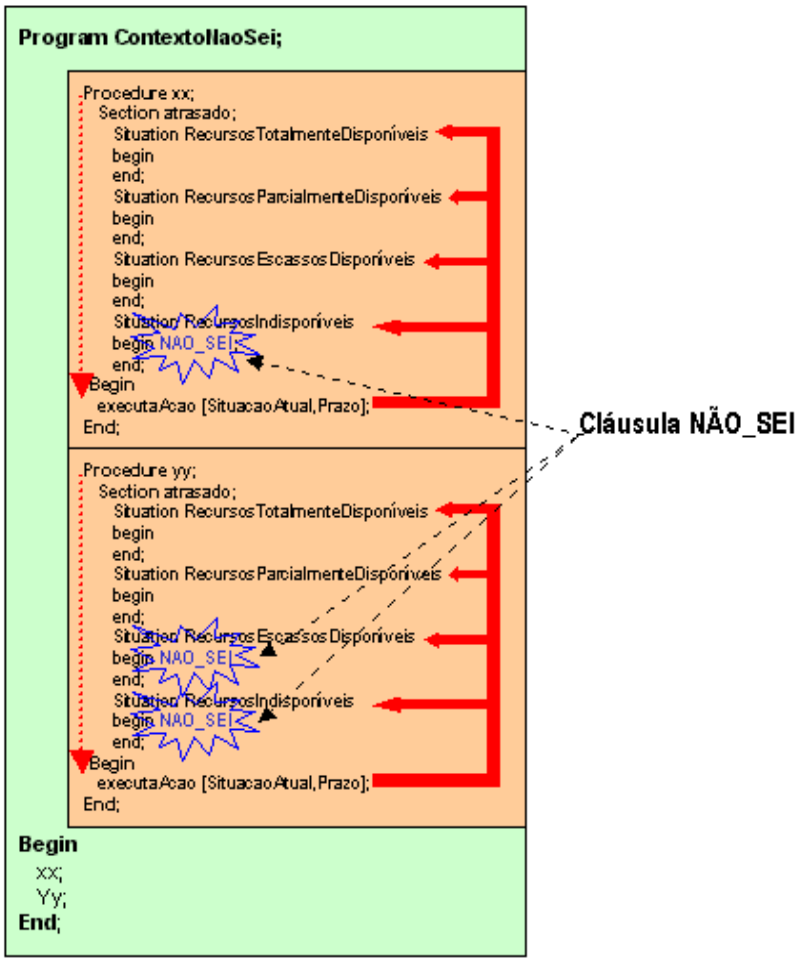


Figura 76 – Uso da cláusula NAO_SEI

Outro aspecto do uso da estrutura de *extendible hashing* é que a profundidade de clusterização pode ser controlada diretamente, por clusters, ao invés da estrutura toda. Assim, à medida que o sistema vai aprendendo a lidar com situações novas, a profundidade dos blocos vai aumentando. Da mesma forma, a capacidade de generalização do sistema vai sendo aprimorada, enquanto a profundidade de clusterização vai sendo diminuída dinamicamente. Tudo isto faz com que procedimentos padrão sejam facilmente desenvolvidos para situações semelhantes.

7.5 Considerações finais

Um aspecto de fundamental importância neste **MM** proposto é que, ao contrário dos modelos descritos no capítulo 2(pág.55), representados na Figura 77[a], onde ficou clara a tendência de permitir às aplicações estender a funcionalidade do sistema a partir de plug-ins instaláveis pelo usuário, no **MM** as aplicações afetam o mundo e sentem, instantaneamente, o efeito de outras entidades que nele atuam (Figura 77b). Ou seja, ao invés de deixar as aplicações penduradas ao redor do núcleo, o modelo proposto as embute dentro do **MM**. Assim não se faz necessário um mecanismo de mensagens na forma tradicional de sinalização via troca de mensagens (mensagens send/receive).

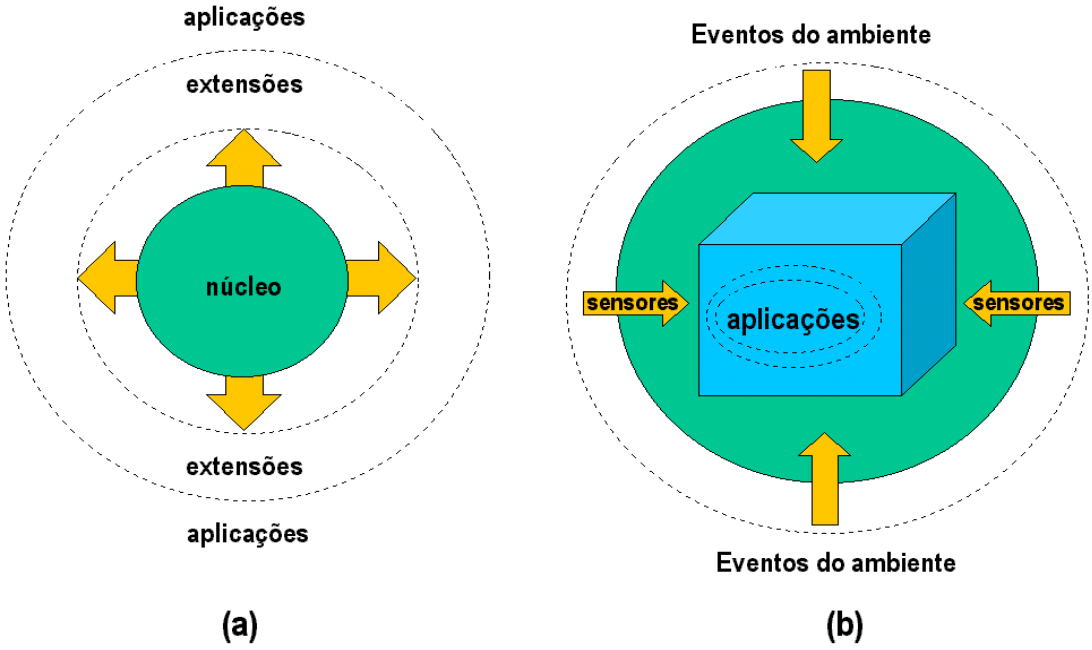


Figura 77 – Comparativo entre modelos exógenos e o modelo proposto (endógeno)

A diferença entre o proposto e os modelos reflexivos é que nestes a aplicação não fica sabendo que houve um *trap* (reificação) para o meta-nível, enquanto no modelo proposto há uma constatação imediata e explícita deste fato, já que o código a ser executado é disparado em função dos parâmetros: estado do mundo/prazo.

Sob a ótica do desenvolvimento de aplicações, o modelo proposto conduz a uma evolução na forma como ocorre o processo de desenvolvimento de soluções em software. Isto porque passa-se de um modelo de ferramentas case gráficas, onde o projetista

manipula ícones, para um modelo de ferramenta case interativa, onde o conhecimento fornecido é validado e assimilado pelo sistema.

O que ocorre no modelo atual pode ser caracterizado através da seguinte analogia: ao não ensinarmos o sistema operacional a resolver os problemas, jamais poderemos dispor de um verdadeiro assistente pessoal no rigor da expressão. Isto porque os programas de aplicação continuam sendo construídos segundo a concepção de quem os desenvolveu (programou). A questão da aprendizagem e adaptação, quando eventualmente existe, é construída sob a ótica de quem (o SO) jamais terá condição de prever os efeitos de todas as possíveis implicações que as generalização e abstrações disponibilizadas no programa terão no contexto do usuário. Além disso, os sistemas de aprendizagem caracterizam-se também como uma casca (*Shell*) que, para o sistema operacional, continua sendo como qualquer outro programa em execução. Assim sendo, o que cada programa individualmente "aprendeu" fica restrito a ele e este conhecimento não é compartilhado pelas demais aplicações e pelo sistema operacional.

Para que seja possível atender aos requisitos identificados, uma nova geração de sistemas operacionais terá que verificar e gerenciar todos os aspectos relativos às atividades que estão ocorrendo na máquina em determinado momento. Assim sendo, algumas abstrações, tal como são conhecidas hoje, deverão ser absorvidas, dando origem a uma nova concepção de aplicação e forma de interação. Por exemplo, o conceito de arquivo deve desaparecer dando origem ao conceito de documento. Não havendo arquivos, o usuário não tem mais necessidade de aprender conceitos de pastas, *drives*, diretórios, permissões, operações de cópia, troca de nomes, etc. Isto porque, como um bom administrador de recursos, o SO sabe onde e de que forma armazenar os antigos "arquivos". Além de saber onde os documentos foram carregados, o sistema conhece seu conteúdo. Assim sendo, o usuário pode interagir com o sistema da mesma forma que um gerente solicita à sua secretária: "dona fulana, por favor, traga-me a carta que enviei para o Sr.Z ontem"; e não mais "Iniciar-Documentos-Carta SrZ.txt" como na metáfora de hoje. Também as funções de consulta às informações deixam de ser orientadas à expressão de busca na linguagem de computação (ex: dir *.txt), e passam a ser orientadas ao conteúdo – EX. procure as cartas enviadas para Sr.Z.

Observe-se que retirando-se a visibilidade do sistema de arquivos, remove-se uma enorme parcela de problemas relacionados ao seu mau-uso, intencional ou não. Usando-

se a premissa popular "o que não é visto, não é lembrado", atinge-se um nível de confiabilidade maior. Se um usuário mal intencionado não sabe o nome, nem a localização dos arquivos no disco, não pode dispor dos dados quaisquer que sejam seus objetivos.

Outra abstração importante que deve desaparecer nesta nova geração é o conceito de programa como é conhecido hoje – um código binário, uma configuração de bits compatível com a arquitetura do processador alvo, que recebe uma fatia de tempo para ser executada e, neste momento, possui o controle completo do mesmo, respeitadas as restrições tradicionais de visibilidade de memória, direitos de acesso, etc.

Se o sistema possui identidade, é ele quem decide quanto de seus recursos emprestar à coletividade (SMA), dependendo do que ele está encarregado de fazer no momento. Máquinas de casa podem ter comportamentos diferentes das máquinas utilizadas em ambientes comerciais e o conhecimento adquirido nestes ambientes será diferente.

O conjunto de requisitos funcionais estabelece uma nova perspectiva de discussão, uma vez que considera um computador tradicional como sendo um robô autônomo e não um simulador de um robô (ou de inteligência humana). Esta afirmação, a princípio, pode ser considerada incomum, dadas as restrições impostas pela constatação da forma física de um computador tradicional. Contudo, cabem as seguintes considerações a este respeito:

- O fato de os computadores não possuírem acessórios que lhes permita mobilidade autônoma (rodas, pernas, ou qualquer outro mecanismo utilizado em robôs autônomos) no espaço físico que os rodeia, não impede que eles percorram grandes distâncias lógicas, ultrapassando qualquer robô físico tradicional em tempo e espaço. Isto é possível a partir do advento da Internet, que possibilita acesso a informações em locais remotos praticamente deforma instantânea;
- As pessoas são representadas em outras máquinas como personagens que exibem alguma forma de expressão de personalidade (como em *Multiple User Domain* - MUD[98]), mas interagem com a própria máquina através de comandos arcaicos;
- O advento da tecnologia de interação por voz começa a impor a necessidade de uma forma de comunicação mais humana entre homens e computadores;
- Pessoas geralmente referem-se aos computadores (e aos problemas associados) como se eles mesmos incorporassem alguma espécie de entidade;

[98] Programas onde o usuário pode entrar utilizando uma representação gráfica denominada avatar. Neste ambiente, os avatares podem comunicarem-se entre si e explorar o ambiente entre outras possibilidades. Para maiores informações consultar: www.mudconnect.com.

- A necessidade de que o usuário passe a valer-se do computador como uma ferramenta útil deve superar a realidade de alguém que, para realizar suas tarefas, necessita ter conhecimento, por exemplo, sobre a localização de objetos dentro do sistema ou tenha que tomar decisões técnicas pelo sistema, geralmente sem qualquer relação com o trabalho realizado.

Finalmente, é possível afirmar que o modelo de mundo proposto é uma estrutura hiperdimensional que permite às aplicações perceber instantaneamente quaisquer alterações ambientais. Da mesma forma, cada uma das aplicações pode "sentir" o efeito de outras que estão executando simultaneamente. Isto porque qualquer flutuação na disponibilidade de algum recurso, imediatamente altera o estado dos sensores da dimensão lógica. Como toda a aplicação precisa estabelecer o que fazer em cada situação, a reatividade é inerente ao modelo.

Além disso, a capacidade de aprendizagem também é inerente ao modelo, conforme foi demonstrado na construção dos protótipos descritos anteriormente:

O protótipo descrito em **Gubler** (2002) mapeou o conhecimento de como adquirir conhecimento sobre construção de algoritmos para uma estrutura que pode ser facilmente convertida para steppables.

O protótipo descrito em **Klotz** (2002) mapeou o conhecimento de como adquirir conhecimento sobre a construção de textos para uma estrutura que também pode ser convertida para steppables

O protótipo descrito em **Heinzen** (2002) demonstrou a viabilidade da utilização de técnicas de RBC para adequar a qualidade da aprendizagem a partir de uma base de casos que vai sendo expandida à medida das interações da entidade inteligente com o seu ambiente.

8 CONCLUSÕES

Conforme apresentado no capítulo 1, a presente tese consistiu, essencialmente, na identificação e sistematização da base conceitual de sustentação de um novo paradigma de concepção de sistema operacional. Os capítulos precedentes resumiram a reflexão realizada e apresentaram as interpretações produzidas.

Portanto, esta tese limitou-se a realizar uma análise conceitual das noções fundamentais que sustentam a concepção de um sistema operacional baseado em conhecimento para, com isso, estabelecer os princípios de um trabalho nesta área. Compreende um primeiro estabelecimento de requisitos funcionais e não funcionais de um projeto de software. Não pretendeu, em momento algum, desenvolver um sistema operacional completo. Contudo, em função do tamanho e da complexidade do projeto, a técnica de prototipação da Engenharia de Software foi utilizada objetivando validar alguns aspectos que dão sustentação ao modelo proposto.

Tanto no exame de qualificação de tese (Mattos,2001) como nos capítulos precedentes, foi caracterizado o problema com o paradigma atual de concepção de sistemas operacionais, através do confrontamento entre seus requisitos e os requisitos das principais demandas da sociedade atual em termos de comércio eletrônico, computação ubíqua e teletrabalho. Conforme exposto nos últimos dois capítulos e, mais especificamente, na discussão do capítulo 7, pode-se afirmar que os fundamentos de um novo modelo de sistema operacional baseado na abstração de conhecimento e comportamento inteligente foi apresentada e detalhada conforme estabelecia o objetivo geral da pesquisa.

Considerando ainda que:

- O capítulo 2(Pág.55) apresentou uma revisão sobre o estado da arte em construção de sistemas operacionais, de forma a permitir uma posterior análise sobre a forma como os sistemas são concebidos e comparar os requisitos dos projetos de sistemas operacionais com os requisitos que as novas demandas estão impondo aos projetos;
- O capitulo 3(Pág.94) apresentou uma revisão sobre as tendências das pesquisas em computação ubíqua, comércio eletrônico e teletrabalho e os requisitos que as aplicações deverão apresentar para operarem neste contexto;
- O capitulo 4(Pág.111) analisou algumas das abordagens de tecnologia de informação que estão sendo consideradas atualmente tendo em vista instrumentalizar o sistema operacional de

forma a atender algumas demandas que o atual modelo ainda não conseguiu resolver. Foram apresentadas algumas pesquisas que procuram instrumentalizar o sistema operacional através do emprego de técnicas da área de Inteligência Artificial. Além disso, foram apresentadas outras tecnologias que estão sendo empregadas com o mesmo propósito. Também foram considerados alguns conceitos fundamentais da área de Robótica. O capítulo concluiu com uma análise que demonstra claramente a dissociação entre os objetivos dos projetos de sistemas operacionais, as demandas identificadas no capítulo 3 e a orientação dos projetos de Robótica;

- O capítulo 5(Pág.142) detalhou a caracterização do problema de pesquisa através do estabelecimento de seus conceitos básicos, seu domínio e seu campo de aplicação, gerando uma declaração formal sobre o mesmo. Esta definição do problema iniciou com algumas considerações sobre o estado da arte em projetos de sistemas operacionais, tendo em vista caracterizar que o modelo legado possui deficiências conceituais, ou seja, enfatizar a dissociação entre as novas demandas projetadas e a forma de concepção dos projetos baseados no atual modelo. Algumas destas falhas e as tarefas que elas impedem de realizar foram discutidas sob os aspectos de confiabilidade, segurança e complexidade de utilização. O capítulo encerrou analisando as projeções e tendências que ressaltam a necessidade de uma nova concepção de projeto de sistema operacional;
- O capítulo 6(Pág.182) apresentou os fundamentos de um novo modelo de sistema operacional baseado na abstração de conhecimento e comportamento inteligente como uma possível solução para os problemas identificados nos capítulos anteriores, utilizando-se de técnicas também apresentadas em capítulos anteriores,

é possível afirmar que os objetivos específicos também foram alcançados.

O modelo proposto apresentou os fundamentos conceituais que permitem a utilização de conceitos de inteligência artificial na construção do sistema e das aplicações que deverão ser executadas pelo mesmo.

O aspecto mais importante a ser considerado nos projetos de sistemas operacionais, visando torná-los mais inteligentes, eficientes e amigáveis, é como torná-los mais sensíveis ao contexto em que estão operando. Como visto até agora, os sistemas operacionais e toda a gama de software desenvolvida por conseqüência, estão baseados no paradigma de Entrada-Processamento-Saída, que é muito restritiva. A tecnologia que tornará os equipamentos do futuro mais inteligentes deverá ser capaz de operar sobre dados obtidos por eles mesmos. Eles terão que "sentir" o ambiente, decidir que aspectos da situação são realmente importantes e inferir a intenção do usuário a partir de ações concretas. As ações do sistema serão dependentes do tempo, do local, ou do histórico das interações com o usuário – ou seja, deverão ser mais dependentes de contexto.

Este aspecto foi abordado através da especificação de um modelo autocontido, onde qualquer evento é imediatamente sentido pelo mundo como um todo. Situações não previstas também são tratadas de forma trivial, tanto através da condição Não-Sei, como

através da constatação de que não há plano escalonado para a execução em determinada situação (seção 7.4.3$_{pág.259}$).

O trabalho abordou o conceito de conhecimento sob a perspectiva antropológica, isto é, como uma ferramenta produzida pelo homem e históricamente realizada. Isto exclui a perspectiva epistemológica (que busca identificar as condições de possibilidade metafísica do conhecimento) e viabiliza o enfoque naturalista proposto em Costa (1993).

Não foi objetivo desse estudo aprofundar a discussão que existe em torno do conceito de inteligência. Mas é preciso destacar que um sistema que tenha as características apresentadas no modelo proposto torna-se, com o passar do tempo, um competente assistente pessoal. Afinal, além de gerenciar os recursos de hardware, ele também pode auxiliar o usuário nas suas atividades como:

- Identificação das informações pertinentes ao contexto do trabalho em execução;
- Gerência de versões dos trabalhos do usuário;
- Construção de associações sobre as operações e contatos do usuário.

Se for feita uma análise entre as projeções futuristas e a situação atual, não é difícil verificar a necessidade de profundas mudanças nos conceitos atuais sobre os quais está baseada a indústria de software. Em função dos argumentos apresentados, consideramos que o projeto apresenta os requisitos mínimos necessários para atingir aos objetivos desta tese de doutoramento, quais sejam:

- Originalidade
  - Concepção de um novo framework de sistema operacional com sólida base científica, bem como em uma revisão bibliográfica significativa.
- Não trivialidade
  - O modelo é complexo e possui uma série de implicações que vão desde a alteração na forma de concepção de programas até à sua forma de execução;
- Relevância
  - Na medida em que o modelo apresenta um indicativo para a solução de problemas que o paradigma atual ainda não resolveu e está longe de resolver completamente;
  - É evolucionário na medida em que se propõe a utilizar tecnologias disponíveis e sedimentadas em outras áreas;
  - É revolucionário na medida em que representa uma nova abordagem de concepção de sistemas computacionais.
- É viável
  - Os protótipos construídos indicam a viabilidade da proposta.

8.1 Recomendações finais

Esta tese não elabora uma teoria da inteligência de máquina, mas antecipa uma variedade de trabalhos experimentais que o desenvolvimento desta estrutura pode comportar. Pensamos, portanto, que todo o trabalho propriamente sistemático, relacionado à construção do modelo proposto, está por fazer.

Uma questão que merece aprofundamento de pesquisa refere-se ao modelo de representação do conhecimento, o qual, atualmente, utiliza somente a estrutura de *steppables*. Pesquisas efetuadas durante a realização deste trabalho indicaram como um caminho promissor a representação espacial do conhecimento adquirido.

Como explicitado ao longo do texto, a necessidade do desenvolvimento de uma nova geração de sistemas operacionais é um fato incontestável. Esta nova geração certamente dependerá de recursos que deverão ser providos pelo sistema operacional, de tal forma que o controle e o conhecimento sobre o que as aplicações estão realizando seja unificado. Assim, vários aspectos relacionados no capítulo 5 podem ser enfocados de forma mais natural, o que deve contribuir para o surgimento de aplicações mais eficientes e orientadas ao auxílio na solução dos problemas do usuário, ou seja, orientadas a tarefas e não a aplicações, como ocorre hoje.

Além disso, identificou-se uma tendência cada vez maior no sentido de incorporar-se conceitos da área de inteligência artificial, de um modo geral, nas várias etapas que compõe a arquitetura de um sistema operacional, de forma a torná-lo mais amigável e fazer com que ele efetivamente assuma um papel de administrador de recursos no sentido mais amplo da palavra, facilitando não só a utilização dos equipamentos através de interfaces mais amigáveis, mas também auxiliando na solução de problemas, o que hoje constitui-se num grande desafio.

A partir das considerações apresentadas anteriormente, pode-se antecipar que:

- Um sistema operacional construído com essas características deve produzir profundas alterações na forma como a computação é tratada hoje. Vale destacar a questão da não visibilidade do sistema de arquivos. Isso acaba com a necessidade de esquemas de proteção atualmente utilizados, já que somente o sistema operacional conhece o lugar onde as informações se encontram e em que formatos estão armazenadas. Naturalmente, há uma contra-partida. As aplicações têm que antecipar suas intenções ao sistema operacional para que este disponibilize acesso ao conteúdo armazenado na forma de visões, semelhantemente ao conceito de visões utilizado em sistemas de gerenciamento de bancos de dados.

- As interfaces de consulta deixam de ser orientadas por palavras-chave ou pelo formato de arquivo e passam a ser orientadas a partir do conteúdo. Assim, usuários leigos podem utilizar o sistema sem necessidade de treinamento. Deixam de existir os aspectos de baixo nível - como extensão de arquivo, diretório, formatação de disquete, reorganização de sistema de arquivos e outros conceitos que hoje fazem parte da dinâmica de qualquer usuário leigo. A conseqüência direta disso é uma maior abrangência e produtividade.
- Outro aspecto importante a ser destacado está relacionado à questão das aplicações. A geração de código executável, na forma como ocorre hoje, é eliminada. A área de engenharia de software também passa a trabalhar num nível de abstração mais alto, obtendo ganhos em produtividade e qualidade. O resultado é que as ferramentas *CASE* (*Computer-Aided Software Engineering*) deixam de ser poderosas ferramentas de desenho e passam a ser poderosas ferramentas de gerenciamento e validação de conhecimento altamente especializado. Com isso, o chamado conhecimento corporativo, adquirido ao longo do tempo através do desenvolvimento de aplicações, vai sendo potencializado. O enfoque passa a ser a otimização de especificações e não a otimização de bits. Fuchs (1992) apresenta uma proposta neste sentido.
- Segundo Mendes e Aguiar (1989, pág.173), o uso de um modelo formal de execução é importante porque provê a base matemática que falta à Engenharia de Software, base esta presente no campo das Engenharias mais antigas e de cuja falta a Engenharia de Software muito se ressente. Neste sentido, o uso de métodos formais permitirá a detecção de erros nas fases iniciais da especificação, com isto diminuindo o custo do software a ser desenvolvido. Portanto, o modelo proposto abre espaço para um caminho de pesquisas no sentido do desenvolvimento e aprimoramento de métodos formais e também para a construção de ferramentas automatizadas que possibilitem o seu uso mais eficaz;

Finalmente, um sistema operacional que apresente as características relacionadas acima, passa a apresentar, como um todo, um comportamento inteligente.

9 BIBLIOGRAFIA

ABOWD, G. D. **Software Engineering Issues for Ubiquitous Computing**. In Proceedings of ICSE'99, ACM Press, pp. 75-84.A2,1999.

AHL,A. et al. **AIX 5L and Windows 2000: Side by Side**. IBM Redbooks. Third Edition. Disponível em: <http://ibm.com/redbooks>, June 2001.

AGRAWAL,R et al. **A Study of Student Privacy Issues at Stanford University**. Communications of the ACM. Vol.45,No.3,pp 23-25, March 2002.

ALFIERI,R.**An Efficient Kernel-Based Implementation of Kernel Threads**. Proc. Of the 1994 Summer Usenix Technical Conference, Boston, MA. June 1994**.**

ALPERN,B.,et al. **The Jalapeño Virtual Machine**. IBM Systems Journal, Vol.39,No 1, February 2000.

AHMED,O. **The Application-specific Operating System**. Computer Design, pp78, Oct 1998.

AMARAAL,J.P., CASTRO, J.F.B. **Assist/DS: Um Assistente Inteligente para o Desenvolvimento de Software**. P.419-432 SBIA93 – X Brazilian Symposium on Artificial Inteligence, 1993.

ALAG,S., PATKI,A.B. **Fuzzy Logic Integrated Circuits**. Joint Conference on Information Sciences-JCIS'95. Wrights Ville Beach, North Carolina, USA, September 1995.

ANCONA,M.,et al. **Channel Reification: a Reflective Approach to Fault-Tolerant Software Development**. In OOPSLA'95 (poster section), page 137, Austin, Texas, USA, on 15th-19th October, ACM, 1995. Disponível em <ftp://ftp.disi.unige.it/ftp/person/CazzolaW>.

ANDERSON,T., et al. **Scheduler Activations: Effective Kernel Support for the User-Level Management of Parallelism**. Proc. Of the 13 ACM Symposium on Operating Systems Principles. Monterey CA, pp.95-109,October 1991.

ANDERSON, T.,et al. **Serverless Network File Systems**. ACM Transactions on Computer Systems, 14(1), Feb. 1996.

ANTIFAKOS,S.,SCHIELE,B. **Beyond Position Awareness**. Personal and Ubiquitous Computing. Springer Verlag London. 6:313-317. 2002.

ARBANOWSKI,S.,STEGLICH,S. **Profiling Contextual Information**. 2nd international workshop on Ubiquitous computing and communications. PACT 2001 International Conference. Barcelona. Sep. 2001.

ARKIN,R. **Behavior-Based Robotics**. Second printing. The MIT Press. 1999.

ASTALOS, J. e HLUCHY,L. **CIS – A Monitoring System for PC Clusters**.Proceedings of EuroPVM/MPI2000, LNCS 1908, J. Dongarra et al (eds.), Springer-Verlag Berlin Heidelberg, 2000, pp.225-232.

ATKINSON, M.P. et al. An Orthogonally Persistent Java,ACM Sigmod Record, 25(4), 1996.

AUN,M.P. **A construção de políticas nacional e supranacional de informação: desafio para os Estados nacionais e blocos regionais**. Revista Ciência da Informação – Políticas e Gestão da Informação. V.28, n.2, p.115-122, maio/ago. 1999.

BACK, G.,HSIEH,W.,LEPREAU,J. **Processes in KaffeOS: Isolation, Resource Management, and Sharing in Java**. Proceedings of OSDI, San Diego, CA, 2000.

BALLESTEROS,F.J. **Off -- Un Nuevo Enfoque en la Construcción de Sistemas Operativos Distribuidos.** Tesis Doctoral. Universidad Politecnica de Madrid. Diciembre 1997.

BALLESTEROS,F.J. HESS,C.K.,KON,F. and Campbell,R.H. **The Design and Implementation of the Off++ and vOff++ μkernels**. Report UIUCDCS-R-99-2086,UILU-ENG-99-1707.University of Illinois at Urbana-Champaign. March 1999.

BALLESTEROS,F.J., et al. **Using Interpreted Composite Calls to Improve Operating System Services**. Software Practice & Experience. John Wiley&Sons. 1999.

BANINO,J.S. **Distributed Couple Actors: A CHORUS proposal for Reliability**. IEEE 3rd International

Conference on Distributed Computing Systems, 1985.

BARRETO,A.A. **A oferta e a demanda de informação: condições técnicas, econômicas e políticas**. Revista Ciência da Informação – Políticas e Gestão da Informação. V.28, n.2, p.168-173, maio/ago. 1999.

BARROS,F. **Towards Standard Interfaces in Dynamic Structure Discrete Event Models**. Proceedings of 2002 AI, Simulation and Planning in High Autonomy systems, AIS'2002. Lisbon, Portugal, April 2002.

BENSOUSSAN,A.,CLIGEN,C.T., DALEY,R.C. **The Multics Virtual Memory: Concepts and Design**. CACM, 15(5),pp.308-318,1972.

BARAK,A. , LA'ADAN,O. **The Mosix Multicomputer Operating System for High Performance Cluster Computing**. Journal of Future Generation Computer Systems, Vol.13,No.4-5, pp.361-372, March 1998.

BEDERSON,B. **Fisheye Menus**. Proceedings of ACM Conference on User Interface Software and Technology (UIST 2000),pp. 217-226. ACM Press. 2000.

BEIGL,M.,ZIMMER,T.,DECKER,C. **A Location Model for Communicating and Processing of Context. Personal and Ubiquitous Computing**. Springer-Verlag London. 6:341-357. 2002.

. **Why develop for the BeOS?** Disponível em: http://www.be.com/developers/developer_program/whybe.html,1998.

,B.N. et al. **SPIN - An Extensible Micro-Kernel for Application-Specific Operating System Services**. Technical Report 94-03-03, Univ. of Washington, 1994.

BERTOLINO,P. Florianópolis, agosto/2003. Entrevista concedida a Mauro Marcelo Mattos.

BEUCHE,D. **An Approach for Managing Highly Configurable Operating Systems**. In J.Bosch and Mitchell, editors, Object Oriented Technology. ECOOP'97 Workshop Reader, volume 1357 of LNCS,pp531-536. Springer-Verlag Berlin/Heidenlberg,1998.

BEUCHE,D., et al. **The PURE Family of Object-Oriented Operating Systems for Deeply Embedded Systems**. Proceedings of 2nd IEEE International Symposium on Object-Oriented Real-Time Distributed Computing. Saint-Malo, France. 1999.

BITTENCOURT G. **Representação de Conhecimento: da Metafísica Aos Programas**. Cursos JAI'98 ed.Belo Horizonte : UFMG, 1998, p.355.

BLACK,A. et al. **Mach and Matchmaker – Kernel and Language Support for Object-Oriented Distributed Systems**. Proceedings of OOPSLA'86,v.21,1986.

BOCHS. **The Cross Plataform IA-32 Emulator**. Disponível em <http://bochs.sourceforge.net>. Último acesso em 12/07/2003.

BOLOSKY, W.J., et al. **Operating System Directions for the Next Millennium**. Microsoft Research. Disponível em <http:// research.microsoft.com/research/os/Millennium/mgoals.html >,2001.

BOMBERGER,A.C.,et al. **The KeyKOS Nanokernel Architecture**. Proceedings of the USENIX Workshop on Micro-Kernels and Other Kernel Architectures, 1992.

BONA,C. **Avaliação de Processos de Software: Um Estudo de Caso em XP e Iconix**. Dissertação de Mestrado. EPS-UFSC, 2002.

BOOKER,P.S.K, et al. **Software Agents and Their Use in Mobile Computing**. Declarative Systems and Software Engineering Group – Technical Report DSSE TR-99-5. University of Southampton.United Kingdom. February 1999.

BORENSTEIN, J.,EVERETT, H.R.,FENG, L.**Where am I? Sensors and Methods for mobile robot prositioning**. University of Michigan.1996. Disponível em: <ftp://ftp.eecs.umich.edu/people/johannb/pos96rep.pd>

BRACHMAN,R.,LEVESQUE,H. **Knowledge Representation and Reasoning**. Chapter 1. Draft (não publicado) - 2000. Disponível em <http://www.cs.washington.edu/education/courses/574/01wi/readings/brachman-levesque-kr.pdf>.

BRACHMAN,R.J. **Systems That Know What They're Doing**. IEEE Intelligent Systems. November/December 2002. pp.67-71.

BRIDGES,S.M. , VAUGHN,R.B. **Intrusion Detection Via Fuzzy Data Mining**. In. Proc. Of the 12th

Annual Canadian Information Technology Security Symposium. pp 111-121. Otawa, June 2000.

BRINCH HANSEN,P. **The Nucleus of a Multiprogramming System**. CACM, 13(4),pp.238-241, 1970.

BRINCH-HANSEN,P. **The programming language Concurrent Pascal**. IEEE Trans. On Software Engineering.1,2,pp.199-207, June 1975.

BROOKS,R. **Humanoid Robots**. Communications of the ACM. March 2002/Vol.45.No.3.

BRUNO,J.,et al. **The Eclipse Operating System: Providing Quality of Service Via Reservation Domains**. Proceedings Of Usenix 98. 1998.

BURK,R.,HORVATH,D.B. **UNIX Unleashed, System Administrator's Edition**. Sams Publishing, 1997.

CAMPBELL,R.H.,JOHNSON,G.M.,RUSSO,V.F. **CHOICES: Class Hierarchical Open Interface for Custom Embedded Systems**. Operating Systems Review, 21(3),pp.9-17,1987.

CAMPBELL,R.H., HINE,J.H., RUSSO,V.F. **Choices for mission critical computing. Studies in Computer and Communication Systems**. chapter 2 pages 11-20. IOS Press, Amsterdam. Netherlands, 1992.

CARDOSO JUNIOR,W.F. **A Inteligência Competitiva Aplicada nas Organizações do Conhecimento como Modelo de Inteligência Empresarial Estratégica para Implementação e Gestão de Novos Negócios**. Tese de Doutorado (Engenharia de Produção). UFSC- Universidade Federal de Santa Catarina. 2003.

CAZZOLA,W. **Evaluation of Object-Oriented Reflective Models**. 1998.

CHAN,W. **Using CoolBase to Build Ubiquitous Computing Applications**. Mobile Systems and Services Laboratory Hewlett-Packard Company.Palo Alto, CA 94304.2001

CHANDAK,M. **Implementation of Sesame in Java**. Master's thesis, Department of Computer Science, University of Illinois at Urbana-Champaign, 1999.

CHEN,L.T. **AgentOS: The Agent-based Distributed Operating System for Mobile Networks**. ACM Crossroads. Disponível em <http://acm.org/crossroads/xrd5-2/agentos.htlm>,2000.

CHERITON,D.R. et al. **The V-Kernel: A software Base for Distributed Systems**. IEEE Software, v.1, n.2, p 19-43, 1984.

CHERITON,D.R.,DUDA,K.J. **A Caching model of operating system kernel functionality**. In Proceedings of the First Symposium on Operating System Design and Implementation, Monterey, California, pp.179-194,1994.

CHIAVENATO, I. **Introdução À Teoria Geral da Administração. 4.ed**. São Paulo: McGraw-Hill, 1993.

CHIKAYAMA,T. **Overview of the Parallel Inference Machine Operating system (PIMOS)**. Proc. Of the Intl. Conference of Fifth Generation Computer Systems. Pp. 230-235. 1988.

CHOU,A.,YANG,J.,CHELF,B.,HALLEM,S.,ENGLER,D. **An Empirical Study of Operating Systems Errors**. Proceedings 18th Symposium on Operating Systems Principles. October 2001.

COADY,Y.,KICZALES,G.,FEELEY,M. **Exploring an Aspect-Oriented Approach to Operating System Code**. Department of Computer Science University of British Columbia. September,2000.

COCKCROFT,A.**New release of the SE Performance Toolkit**. Disponível em: <http://www.sun.com/960301/columns/adrian/ column7.html>, March 1995.

COLUSA. **Omniware Technical Overview**. Colusa Software White Paper. Disponível em: <http://www.cs.umbc.edu/ agents/papers/omniware.ps>, 1995.

COMER,D. **Operating System Design: The Xinu Approach**. Prentice Hall,1984.477p.

CONNECTIX. **The Technology of Virtual Machines**. White Paper.
Disponível em: < http://www.connectix.com>. 2001.

CORBATÓ,F.J.,MERWIN-DAGGETT,M.,DALEY,R.C. An experimental time-sharing system. In Proceedings of the AFIPS Spring Joint Computer Conference, pp 335-344, May 1962.

CORBATÓ,F.J.,VYSSOTSKY,V.A. **Introduction and Overview of the Multics System**. Proceedings of AFIPS FJCC,pp.619-628, 1965.

COSTA,A.C.R. Inteligência de Máquina: Esboço de uma Abordagem Construtivista. Tese de Doutorado. PPGCC – UFRGS, Universidade Federal do Rio Grande do Sul, 1993.

COSTA,A.,et al. **Hardware Solutions for Fuzzy Control**. Proceedings Of the IEEE,v.83,pp.422-434, March 1995.

COWAN,C. **Related Work in Adaptive Operating Systems**. Disponível em: <http://cse.ogi.edu/DISC/projects/synthetix>, 1997.

CREASY,R.J. **The Origin of the VM/370 Time-sharing Systems**. IBM Journal of Research and Development, V.25,n.5,p483-490,1981.

D'INVERNO,M,P. **Agents, Agency and Autonomy: A Formal Computational Model**. Doctoral Thesis. Department of Computer Science. University College London,1998.

DABEK,F., et al. **Wide-area Cooperative Storage With CFS**. Proceedings of UbiComp'01, Atlanta, USA, Lecture Notes in Computer Science, Springer, Heidelberg, 2001.

DALAL, Y.K. et al. **Pilot: An Operating System for a Personal Computer**, Communication of the ACM, v.23, n.2, 1980.

DALEY,R.C.,DENNIS,J.B. **Virtual memory, processes, and sharing in MULTICS**. Communications of the ACM, 11(5):306-312, May 1968.

DASGUPTA,P., et al. **The Design and Implementation of the Clouds Distributed Operating System**. Computing Systems,3(1),pp.11-46, 1990.

DAVIDSSON,P. **On the Concept of Concept in the Context of Autonomous Agents**. Second World Conference on the Fundamentals of Artificial Intelligence, pp 85-96, 1995.

DE SOUZA,C.S. **The Semiotic Engineering of User Interface Languages**. International Journal of Man-Machine Studies, v.39,p.753-773. 1993.

DEARLE,A., et al. **Grasshopper: an Ortogonally Persistent Operating System**. Computer Systems, Summer, pp.289-312, 1994.

DEARLE, A.,HULSE D. **Operating systems support for persistent systems: past, present and future**. Software Practice and Experience, 30(4), 295-324, 2000.

DERTOUZOS, M. L. **The future of computing**. Scientific American, July 1999.

DIENG,R. et al . **Method and Tools for Corporate Knowledge Management**. Proceedings of KAW'98 - Eleventh Workshop on Knowledge Acquisition, Modeling and Management, Voyager Inn, Banff, Alberta, Canada. April, 1998

DEITEL,H.M. **Operating Systems**. Addison-Wesley, 2nd edition, 1990.

DENIS,J.B.,VAN HORN,E.C. **Programming Semantics for Multiprogrammed Computations**. MCACM, Vol. 9, No3, pp.143-155, March 1966.

DIXON,N.,et al.**The Treatment of Persistent Objects in Arjuna**. The Computer Journal, vol.32, no.4, 1989.

DTOS. **DTOS Home Page**. Disponível em: <http://www.sctc.com/randt/HTML/summary.html>. Acesso em 8/11/2001.

DUMON,P. **OS Kernels: A Little Overview and Comparisson**. Disponível em: <http://unios.dhs.org/osKernels.html>, 1997.

ECOOP. **2nd ECOOP Workshop on Object-Orientation and Operating Systems**. Lisbon, June 1999. Disponível em <http://www.tu-chemintz.de/informatik/osg/ecoopooosws/ecoop-ooosws99>.

EICHFELD,H.,et al. **A General Purpose Fuzzy Inference Processor**. IEEE Micro,v.15,n.3,pp.12-17, June 1995.

EINSTEIN ,A. **A Teoria da Relatividade Especial e Geral**. Tradução Carlos Almeida Pereira. Contraponto Editora, 1ª Edição. 2001, 132p.

ELHAJJI,M. **Novas estratégias organizacionais no cenário global**. Revista Ciência da Informação – Políticas e Gestão da Informação. V.28, n.2, p.111-114, maio/ago. 1999.

ELLIOTT,J. **Ubiquitous Computing**. Disponível em: <http://www.cc.gatech.edu/classes/cs6751_97_fall/projects/follow_me/exam/jason.html>. Último acesso em março/2003.

ENDO,Y.,et al. VINO: The 1994 Fall Hrvest. Center for Research in Computing Technology. TR-34-94.Harvard University. Cambridge, Massachusetts. 1994.

ENGLEMORE,R. MORGAN,T. (eds.). **Blackboard Systems**, Addison-Wesley, Reading, MA. 1988.

ENGLER,D.R.,KAASHOEK,M.F. e O'TOOLE JR,J. **The Operating system kernel as a secure programmable machine**. Proceedings of the Sixth SIGOPS European Workshop: Matching Operating Systems to Application Needs. 1995.

ENGLER,D.R., KAASHOEK, M.F. E**xokernel: An Operating System Architecture for Application-Level Resource Management.** Proceedings of the Fifteenth ACM Symposium on Operating System Principles, Copper Mountain, Colorado, 1995.

ERIKSSON,H.E., PENKER,M. **UML Tooklit**. John Wiley & Sons. 1998.

ETZIONI,A.,et al. **The Softbot Approach to OS Interfaces**. IEEE Software, 12(4):42-51, 1995.

EURESCON. **Jini & Friends @ Work: Towards Secured Service Access**. Operator Specific Security Requirements for Ubiquitous Computing. June 2001.

FABRE,J, et al. Implementing fault tolerant applications using reflective object-oriented programming. IEEE Proceedings of FTCS-25 "Silver Jubilee", Pasadena, CA USA, June 1995.

FANO,R.M.,CORBATÓ,F.J. **Tempo-Partilhado em Computadores**. Scientific American 1966. Computadores e Computação:Textos do Scientific American. Editora Perspectiva. 1977, 332p.

FERREIRA,M.P. **Desenvolvimento de Software Alinhado aos Objetivos Estratégicos do Negócio: Proposta de Uma Metodologia**. Dissertação de Mestrado. UFSC, 2002.

FIUCZYNSKI, M.E., MARTIN, R.P., BERSHAD B.N.,CULLER, D.E. **SPINE: An Operating System for Intelligent Network Adapters**. University of Washington Technical Report UW-CSE-98-08-01.1998.

FLEISH, B.D. **Operating Systems: A perspective on future trends**. ACM SIGOPS Operating Systems Review. Vol 17, no.2, pp 14-17, April 1983.FOLLIOT,B. **The Virtual Virtual Machine Project**. Invited Talk at SBAC'2000 - Simpósio Brasileiro de Arquitetura de Computadores e Processamento de Alto Desempenho, 2000.

FORD,W. **Computer Communication Security: Principles,Standard Protocols and Techniques**. Prentice Hall, Englewood Cliffs,NJ 1994.494p.

FORRESTER,J.E. **An Empirical Study of the Robustness of Windows NT Applications Using Random Testing**. 4th USENIX Windows Systems Symposium. Seattle, August 2000.

FOSTER,I.,KESSELMAN,C. **Globus: A metacomputing Infrastructure Toolkit**. Ian Foster, Carl Kesselman. The International Journal of Supercomputer Applications and High Performance Computing, 1996.

FRANKLIN,S., KELEMEN,A.,MCCAULEY,L. **IDA: A Cognitive Agent Architecture**. In IEEE Conference on Systems, Man and Cybernetics. : IEEE Press. 2646-2651. 1998.

FREITAS,C.M.D.S. **Computação Gráfica**. Anais VI Escola de Informática da SBC Regional Sul. Maio 1998. pp.1-14.

FRIEDRICH,L.F. **COMPOSES: Uma Abordagem para Composição de Sistemas Operacionais**. Revista RITA – Universidade Federal do Rio Grande do Sul,2002.

FUCHS N. **Specifications are (preferably) executable**. Software Engineering Journal,pages 323-334, September 1992.

GAUSE,D.C.,WEINBERG,G.M. **Explorando Requerimentos de Sistemas**, Makron Books, 1991. 368p

GENERA. **Genera Concepts: The Best Software Environment Available**. Disponível em: <http://kogs-www.informatik.uni-hamburg.de/~MOELLER/symbolics-info/GENERA/genera.html>1990.

GHEZZI,C.,VIGNA,G. **Mobile code paradigms and technologies: a case study**. In K.Rothermel and Popescu-Zeletin,R., editors, Proceedings of the 1st International Workshop on Mobile Agents (MA'97),

LNCS 129, Springer, April 1997.

GIARRATANO,J.C.,RILEY,G. **Expert Systems; principles and programming**. PWS Publishing Co, boston, 1994.

GOLM,M. **Design and Implementation of a Meta Architecture for Java**. Diplomarbeit im Fach Informatik, Friedrich-Alexander-Universität Erlangern-Nürenberg, Jan 1997.

GOLM,M, KLEINÖDER,J. **MetaJava – A Platform for Adaptable Operating-System Mechanisms**. ECOOP 97 – Workshop on Object-Orientation and Operating Systems. Finland. June 1997.

GOLM,M, KLEINÖDER,J. **Ubiquitous Computing and the Need for a New Operating System Architecture**. Proceedings of UbiComp'01, Atlanta, USA, Lecture Notes in Computer Science, Springer, Heidelberg, 2001.

GOLM,M. WAWERSICH,C. BAUMANN,J., FELSER,M. KLEINÖDER ,J. **Understanding the Performance of the Java Operating System JX using Visualization Techniques**. Workshop on Software Visualization, OOPSLA 2001, Tampa, FL, October 15, 2001

GOWING,B.,CAHILL,V. **Making Meta-Object Protocols Practical for Operating Systems**. proceedings of 4th International Workshop Object Oriented in Operating Systems,pp.52-55,April 1995.

GRIMM,R., et al. **System-Level Programming Abstractions for Ubiquitous Computing**. University of Washington. 2001.

GUBLER,A. **Protótipo de um sistema especialista para auxiliar no ensino de algoritmos**. Trabalho de Conclusão de Curso. FURB, 2002.

GUTIÉRREZ,D.A.,et al. **An Object-Oriented Abstract Machine as the Substrate for an Object-oriented Operating System**. ECOOP'97 – Workshop on Object-Orientation and Operating Systems. Jyväskylä, Finland. June 1997.

GÜRER,D.W.,et al.**An Artificial Intelligence Approach to Network Fault Management**. SRI International, 2001.

HAN,J,KANEBER,M. **Data Mining: Concepts and Techniques**. Morgan Kaufmann,2000.

HARDY,N. **The KeyKOS Architecture**. Operating Systems Review, September 1985.

HARNAD, S. **Problems,Problems: The Frame Problem as a Symptom of the Symbol Grounding Problem**. Psycoloquy 4(34). 1993.

HARTMAN,J.H. et al. **Scout: A communication-oriented Operating System**. TR 94-20, University of Arizona, Tucson, AZ, June 1994.

HAZZAH,K. **Writing Windows VxDs and Device Drivers**. R& D Books, Lawrence, 1997.

HEINZEN,L. **Módulo de RBC em uma Ferramenta de Apoio ao Ensino de Lógica de Programação.** Trabalho de Conclusão de Curso. FURB, 2002.

HEBBARD,B., et al. **A penetration Analysis of the Michigan Terminal System**. Operating Systems Review, Vol.14,No.1, pp.7-20, June 1980.

HERNÁNDEZ,L.,VIVANCOS,E.BOTTI,V**. Intelligent Scheduling of Production Systems in a Real-Time Architecture**. IBERAMIA'98, 1998, p429-438.

HERRMANN, F. et al. **Chorus distributed operating system**. Computing Systems,v.1(4), p.305-367, 1988.

HOFSTADTER,D.R. **Godel,Escher,Bach: An Eternal Golden Braid**. Penguin Books,1980.

HOHMUTH,M. **The Fiasco Kernel: Requirements Definition**. Technical Report, TU Dresden. September 1998.

HOUAISS. **Dicionário Eletrônico Houaiss da Língua Portuguesa**. 2002.

HOPPEN,N.,LAPOINTE,L,MOREAU,E. **Um Guia para a Avaliação de Artigos de Pesquisa em Sistemas de Informação**. READ-Revista Eletrônica de Administração, Ed.3,no.2. UFRGS, Nov 1996.

HULSE, D., DEARLE, A. **Trends in Operating System Design: Towards a Customizable Persistent Micro-Kernel**. Report Pastel RT1R4. University of Stirling 1998.

HIROTA,K.,OZAWA,K. **The Concept of Fuzzy Flip Flop**. IEEE Trans. On Systems, Man, and Cybernetics, v.19,n.5,pp.980-997. Sept/Oct 1989.

HORVITZ,E. et al. **The Lumière Project: Bayesian User Modeling for Inferring the Goals and Needs of Software Users**. Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence,. Morgan Kaufman Publishers, pp Madison, WI, July 1998

HÜBNER,J.F. **Um modelo de reorganização de sistemas multiagentes**. Tese de Doutorado, Escola Politécnica da USP, São Paulo, 2003.

HUGH,J. **User Interface Design – How Can It Be Improved?** Disponível em: < http://www-dce.doc.ic.ac.uk/~nd/surprise97/journal/vol2/hafj/hafj.html>,1997.

HUNT,G.,BRUBACHER,D. **Detours: Binary Interception of Win32 Functions**. Proceedings of the 3rd USENIX Windows NT Symposium, Seattle, WA, July 1999.

HYDARI,M.Z. **Design of the 2k Naming Service**. Tese de Doutorado, University of Illinois at Urbana-Champaing, 1999.

IEEE Std 1003.1, 1996 Edition – Description. Disponível em: <http://standards.ieee.org/READING/IEEE/std_public/description/POSIX/9945-1-1996_desc.html>, 1996.

DORWARD,S. et al. **Inferno**. Proceedings of the 42nd IEEE Computer Society International Conference. San Jose, CA, Feb 1997.

ISDALE,J. **IV'98 Trip Report**. IEEE Information Visualization 98 Symposium and Visualization Toolkit Tutorial. Trip Report. Research Triangle Park, October 1998. Disponível em: <http://www.isdale.com/jerry/InfoVis/noframes/IEEE_InfoVis98-Info.html>,1998.

IYER,S.,DRUSCHEL,P. **Anticipatory scheduling: A disk scheduling Framework to Overcome Deceptive Idleness in Synchronous I/O**. Proceedings of the 18th ACM Symposium on Operating System Principles, October 21-24, 2001. Operating System Review 35(5), ACM, pp.117-130, 2001.

JONES,M.B., REGEHR,J.,SAROIU,S. **Two case studies in predictable application scheduling using rialto/NT.** RTAS 2001.

KAASHOEK,F. **The Aegis System**. Proceedings of the 1995 Conference on Hot Operating Systems, Orcas Island, WA, May 1995.

KAELBLING,L.P. **An Architecture for Intelligent Reactive Systems**. In Allen,J,Hendler,J,Tate,A. (eds.) Readings in Planning. Morgan Kaufmann Publishers,1990,754p

KALAKOTA,R.,WHINSTON,A.B. **Frontiers of Electronic Commerce**. Addison-Wesley Publishing Company. 1996.

KAMIBAYASHI,N.,OGAWANA,H.,NAGAYAMA,K.,AISO,H. **HEART: An Operating System Nucleus Machine Implemented by Firmware**. Proceedings of Symposium on Architectural Support for Programming languages and Operating Systems. Palo Alto,Califórnia, March 1982. Pp.195-204

KANDEL,A.,ZHANG,YQ, HENNE,M. **On use of fuzzy logic technology in operating systems**. Fuzzy Sets and Systems 99, Elsevier Science, pp 241-251, 1998.

KEEDY,J.L. **Support for Objects in the MONADS Architecture**. Proceedings of the Third International Workshop on Persistent Object Systems, Newcastle, Australia, pp.392-406, 1989.

KICZALES,J.,et al. **Aspect-Oriented Programming**. M.Aksit and S.Matsuoka, editors, Proceedings of the 11th European Conference on Object-Oriented Programming (ECOOP'97), volume 1241 of Lecture Notes in Computer Science, pp 220-242. Springer-Verlag, June 1997.

KISTLER,J.J.,SATYANARAYANAN, M. **Disconnected Operation in the Coda File System**. ACM Transactions on Computer Systems, 10(1):3-25, February 1992.

KLINGER,A.,ALEXANDRIDIS,N. **Picture decomposition,Tree Data Structures, and Identifying Directional Symmetries as Node Combinations**. Computer Graphics and Image Processing, 8, p.43-77,1978.

KLOTZ,G. **Protótipo de um sistema de apoio à escrita de Redações**. Trabalho de Conclusão de Curso. FURB, 2002.

KON,F. **Automatic Configuration of Component-Based Distributed Systems**. PhD thesis, Dept. of Comp. Science, Univ. of Illinois, Urbana-Champaign, may 2000.

KORNER,K. Intelligent Caching for Remote File Service. Proc. Of the 10th Intl. Conference on Distributed Computing Systems. Pp220-226. 1990.

KRUMM,J. et al. **Easyliving: Technologies for intelligent environment**. Microsoft Research. 1999.

KUMAR,S.,MANDELBAUM,Y.,YU,X..,LI,K. **ESP: A Language for Programmable Devices**. Programming Languages Design and Implementation. ACM, 2001.

LAMPING,J., et al. **An Architecture for an Open Compiler**. In Akinori Yonezawa and Brian C. Smith, editors, Proceedings of the Int'l Workshop on Reflection and Meta-Level Architecture, pages 95–106, 1992.

LANGE,D.B.,OSHIMA ,M. **Programming Mobile Agents in Java - with the java aglet api (the aglet cookbook)**.Disponível em: <http://www.trl.ibm.co.jp /aglets/ aglet-book/index.html>, 1997.

LARNER,D.L**. A Distributed, Operating System Based, Blackboard Architecture for Real-Time Control**. CACM. 1990.

LARMAN,C.**Utilizando UML e Padrões: Uma Introdução à Análise e ao Projeto Orientados a Objetos**.Tradução: Salgado,L.A.M.,Bookman,Porto Alegre, 2000.

LECKIE,C. **Experience and Trends in AI for Network Monitoring and Diagnosis**. Proceedings of the IJCAI95 – Workshop on AI in Distributed Information Networks. 1995.

LEROUX,P.N. **Real Time or Real Linux? A realistic Alternative**. White Paper. Disponível em: <http://www.qnx.com/ literature/whitepapers/reallinux.html>, 2001.

LESLIE,I et al. **The Design and Implementation of an Operating System to Support Distributed Multimedia Applications**. Disponível em:
<http://www.cl.cam.ac.uk/Research/SRG/netos/nemesis/documentation.html>, 1997.

LI,X.,XING,D.,Jun,C.,YUHUA,Z.,ZHONGXIU,S. **An Introduction to Intelligent Operating System KZ2**. ACM Operating Systems Review, Vol.29 no.1,1995.

LI,X. **Li Home Page**. Disponível em <http:\\dawww.nju.edu.cn\expert\xieli\thesis.htm>, 2003.

LIEDTKE,J. **On u-Kernel Construction**. Proceedings of the 15th ACM Symposium in Operating Systems Principles, pp.237-250. Copper Mountain, CO, 1995.

LINDE,R. **Operating System Penetration**. AFIPS Conference Proceedings. Vol. 44,1975.

LOPES,C.V.,KICZALES,G. **D:A Language Framework for Distributed Programming**. Xerox Palo Alto Research Center. Disponível em: <http://www.parc.xerox.com/sp1/projects/aop>,1997.

LOWEKAMP,B. et al. **A Resource Query Interface for Network-Aware Applications**. Cluster Computing. Baltzer Science Publishers, (to appear in 1999).
Disponível em <http://www.cs.cmu.edu/~cmc/remulac>.

MACHADO,C.S. **Gerenciamento da Segurança da Informação em Sistemas de Teletrabalho**. Dissertação de Mestrado. PPGEP-UFSC. 2002.

MACHADO,F.B.,MAIA,L.P. **Arquitetura de Sistemas Operacionais**. LTC – Livros Técnicos e Científicos Editora S.A. Rio de Janeiro, 2002.

MADHYASTHA,T.M., REED,D.A. **Intelligent, Adaptive File System Policy Selection**. Proceedings of the Sixth Symposium on the Frontiers of Massively Parallel Computation, October 1996.

MANZOUL,M.A. **Faults in Fuzzy Logic Systolic Arrays**. International Journal on Cybernetics and Systems, v.21,pp.513-524, 1990.

MARINOS,P.N. **Fuzzy Logic and Its Application to Switching Systems**. IEEE Transactions On Computers,vC18,No.4,pp.343-348, April 1969.

MATTOS,M.M.FERNANDES,A. LÓPEZ,O.C. **Sistema especialista para apoio ao aprendizado de lógica de programação**. VII Congreso Iberoamericano de Educación Superior en Computación.-CIESC99, Asunción, Paraguay. Set. 1999.

MATTOS,M.M.,TAVARES,A.C. **VXt: Descrição da Implementação de um Simulador de Hardware**.

Anais VI Seminário de computação. FURB, Blumenau. Novembro 1999.

MATTOS,M.M., MARTINS, A., PACHECO, R. **O Emprego de Técnicas de IA no suporte a Administração de Sistemas Operacionais**. Anais, II Simpósio de Informática do Planalto Médio - SIPM'2000, Universidade.de de Passo Fundo , Passo Fundo, maio de 2000.

MATTOS,M.M., PACHECO, R. **Computação reconfígurável em sistemas operacionais baseados em contexto**. Anais Seminário de Computação Reconfígurável - SCR'2001, Instituto de Informática, PUC de Minas Gerais, Belo Horizonte, MG agosto de 2001.

MATTOS,M.M. **Sistemas Operacionais Baseados em Conhecimento (Knowledge-based operating systems)**. Exame de qualificação de tese. PPGEP – Programa de Pós-Graduação em Engenharia de Produção, UFSC. Florianópolis, Dezembro 2001.

MATTOS,M.M.,PACHECO,R. **Dynamic models as Knowledge in Operating Systems Kernels**. Proceedings of 2002 AI, Simulation and Planning in High Autonomy systems, AIS'2002. Lisbon, Portugal, April 2002.

MATTOS,M.M. **Learning How to Build Abstractions in Programming Logics Classes**. IE-2002 - VI Congreso Iberoamericano de Informática.IV Simpósio Internacional de Informática Educativa. VII Taller Internacional de Software Educativo. Vigo, Espanha, novembro de 2002.

MATURANA, H.R.. **The Organization of the Living: a Theory of the Living Organization**. International Journal of Man-Machine Studies, 7:313-332,1975

MENDES,S.,AGUIAR,T.C. **Métodos para Especificação de Sistemas**. Ed. Edgard Blücher, 1989, 183p.

MCCAULEY,E.J. **KSOS: The Design of a Secure Operating System**. AFIPS Conference Proceedings. Vol.48.,pp.345-353, 1979.

MCCARTHY,J. **Informação**. Scientific American 1966. Computadores e Computação: Textos do Scientific American. Editora Perspectiva. 1977, 332p.

MCCARTY,J.,HAYES,P.J. **Some Philosophical Problems from the Standpoint of Artificial Intelligence**. In D.Michie e B.Meltzer, editors, Machine Intelligence 4, pp.463-502. Edimburgh University Press, Edimbourgh,GB,1969.

MEDEIROS,A.A.D. **A Survey of Control Architectures for Autonomous Mobile Robots**. Journal of the Brazilian Computer Society. N.3,V.4.Rio de Janeiro. 1998.

MERY,D. **Why a different operating system needed?** Disponível em: <http://www.symbian.devnet.com/techlib/ techcomms/techpapers>, 2001.

MILLER,B.P.,FREDRIKSEN,L.,SO,B. **An Empirical Study of the Reliability of UNIX Utilities**. CACM 33,12, pp.32-44, December 1990.

MILLER,B.P., et al. **Fuzz Revisited: A Re-Examination of the Reliability of UNIX Utilities and Services**. University of Winsconsin-Madison. Appears in Empirische Studie zur Zuverlasskeit von UNIX-Utilities: Nichts dazu Gerlernt.iX, September 1995.

MINSKY, M.L**. Inteligência Artificial**. Scientific American 1976.Computadores e Computação:Textos do Scientific American. Editora Perspectiva. 1977, 332p.

MITCHELL,S.E.,et al. **Adaptive Scheduling using Reflection**. ECOOP'97 Workshop on Reflective Real-time Object-Oriented Programming and Systems, Jyvaskyla, Finland, June 1997.

MOELLER,B. **Constructing Programs from Specifications**. Ed North Holland, 1991.

MOHR,S.T. **Software Design for a Fuzzy Cognitive Map Modeling Tool**. Dissertação de Mestrado. Rensselaer Polytechnic Institute, 1997.

MONTEZ,C.B. **Um Sistema Operacional Com Micronúcleo Distribuído e um Simulador Multiprogramado de Multicomputador**. Dissertação de Mestrado. UFSC. 1995

MONTZ, A. B, MOSBERGER, D., O'MALLEY, S. W. , PETERSON, L. L. , PROEBSTING, T. A.. **Scout: A Communications-Oriented Operating System**. Hot OS, May 1995.

MOSBERGER,D. **Scout: a Path-based Operating System**. Doctoral Thesis. University of Arizona, 1997.

MOORE,R.B. **An Extensible Architecture for Distributed Object System Interoperability.** Masters

Thesis, Department of Computer Science, University of Illinois at Urbana-Champaign, Aug 1998.

MORRISON,R. et al. **The Napier88 Reference Manual.** Technical Report PPRR-77-89 URL:, Universities of Glasgow and St Andrews,1989.
Disponível em: < http://www-fide.dcs.stand.ac.uk/Publications/1989.html#napier.reference.manual>.

MULLENDER,S.J. et al. **Amoeba, A Distributed operating system for the 1990's**. IEEE Computers. V.23,p44-53,1990.

MUSSER,G. **Um Buraco no Coração da Física**. Scientific American Brasil. Out. 2002 pp60-61.

MUTHITACHAROEN,A.,CHEN,B.,MAZIÈRES,D. **A low-bandwidth network file system**. Proceedings of UbiComp'01, Atlanta, USA, Lecture Notes in Computer Science, Springer, Heidelberg, 2001.

MYERS, B. A., HUDSON S. E.,PAUSCH R. **Past, Present, and Future of User Interface Software Tools**. ACM Transactions on Computer-Human Interaction, Vol. 7, No. 1, pp. 3-28, March 2000.

NAGLE,D., UHLIG,R., MUDGE,T., SECHREST,S. **Optimal Allocation of On-chip Memory for Multiple-API Operating Systems**. In Proceedings of the 21st Annual International Symposium on Computer Architecture, pages 358--369, April 1994.

NARAYANAN,D., FLINN,J., SATYANARAYANAN,M**. Using History to Improve Mobile Application Adaptation**. Proceedings of the 3rd IEEE Workshop on Mobile Computing Systems and Applications, Monterey, CA, December,2000.

NEEFE, J. et al. **Improving the Performance of Log Structured File Systems**. Disponível em: <http://http.cs.berkeley.edu/~neefe/papers/osdi_submit.ps>, 1996.

NELSON,T.H. **Whats on my mind**. Invited talk at First Weareable Computer Conference, Fairfax VA. Disponível em: <http://www.sfc.keio.ac.jp/~ted/zigzag/xybrap.html>, May 1998.

NEWMAN,J.,INGRAM,D.,HOPPER,A. **Augmented Reality in a Wide Area Sentient Environment**. In Proceedings of the 2nd IEEE and ACM International Symposium on Augmented Reality (ISAR 2001), New York ,October 2001.

NICOL, J. **Operating System Design: Towards a Holistic Approach**. Operating System Review. v.21, n.1, 1987.

NICOL, J., et al. **Cosmos: An Architecture For a Distributed Programming Environment**. Computer Communications, v.12, n.3, p.147-157, 1989.

NIELSEN,J. **Enhancing the Explanatory Power of Usability Heuristics**. Proceedings of ACM CHI'94, Conference. Boston, MA, April 1994.

NOBLE,B. **System Support for Mobile, Adaptive Applications**. IEEE Personal Communications. February 2000.

NOVOSELSKY**,A.,Karun,K. XSLTVM – An XSLT Virtual Machine.** Proceedings of XML Europe 2000. Paris, France, June 2000.

NSA. **National Security Agency Shares Security Enhancements to Linux**. NSA Press Releases. Disponível em: <http://www.nsa.gov/releases/selinux_01022001.html>, January 2001.

OETTINGER,A.G. **O Uso do Computador na Ciência**. Scientific American 1966. Computadores e Computação:Textos do Scientific American. Editora Perspectiva. 1977, 332p.

OLIVEIRA,R.S.,CARISSIMI,A.S.,TOSCANI,S.S. **Sistemas Operacionais**. Série Livros Didáticos. Instituto de Informática UFRGS. N.11. Ed.Sagra Luzzatto,2000.

OLIVEIRA,S.L. **Tratado de Metodologia Científica: Projetos de Pesquisas, TGI, TCC, Monografias, Dissertações e Teses**. Ed.Pioneira, São Paulo, 2002.

ONEY,W. **Programming the Microsoft Windows Driver Model**. Microsoft Press, Redmond,1999.

ORWANT,J. **For Want of a Bit The User Was Lost: Cheap User Modeling**. IBM Systems Journal, Vol 35, Nos 3&4, 1996.

PABLO. **Intelligent Information Spaces: A test bed to Explore and Evaluate Intelligent Devices and Augmented Realities**. Disponível em:
<http://www-pablo.cs.uiuc.edu/Project/SmartSpaces/SmartSpaceOverview2.htm>, 2001.

PASQUALE,J. Using Expert Systems to Manage Distributed Computer Systems. IEEE Network. Set.1988.

PATKI,A.B. **Fuzzy Systems: Technology Mission Approach**. Technical Report – DE/NMC/93/5, May 1993.

PATKI,A.B. **Fuzzy Logic Based Hardware: Some Experiences**. Proceedings of First International Discourse on Fuzzy Logic and the Management of Complexity. FLAMOC'96, January, 15-18, Sydney, 1996.

PATKI,A.B.,RAGHUNATHAN,G.V. **On Datatypes for Object Oriented Methodology for Fuzzy Software Development**. WSC1, Proceedings Of 1st On-line Workshop on Soft-Computing, Nagoya, Japan, pp163-167,1996.

PATKI,A.B. **Exploration of Developmental Trends in Java Technology.** Electronics Information & Planning,Vol.25,No.3,pp.125-133,December 1997.

PATKI A.B., RAGHUNATHAN G.V., KHURSHID A. **FUZOS—Fuzzy Operating System support for Information Technology**. Proceedings of Second On-line World Conference On Soft Computing In Engineering, Design And Manufacturing. Cranfield University, UK, June 1997.

PATKI A.B., RAGHUNATHAN G.V**. Trends in Fuzzy Logic Hardware**. Proceedings of the First On-line Workshop On Soft Computing, WSC1. Pp. 180-185. Nagoya, Japan, August 19-30, 1996.

PLAICE,J.,Paquet,J. Introduction to intensional programming. Intensional Programming I, World Scientific, Singapore, 1996.

POPEK,J., KLINE,C. **Verifiable Secure Operating System Software**. AFIPS Conference Proceedings,Vol. 43, 1974.

RASHID,R. et al. **Mach: A System Software Kernel**. Proceedings of the 34th computer Society International Conference COMPCON 89, 1989.

REED,D.,DONNELLY,A.,FAIRBAIRNS,R. **Nemesis the Kernel: a Brief Overview**. Disponível em: <http://www.cl.cam.ac.uk/Research/SRG/netos/pegasus/publications/overview>, 1997.

REED,D.A. **PPFS II - The Next Generation Intelligent, Adaptive Parallel File System**. Disponível em: <http://www-pablo.cs.uiuc.edu/Projects/PPFS/PPFSII/PPFSIIOverview.htm>, 2001

REDELL,D.D. et al. **Pilot: An Operating System For a Personal Computer**. Proceedings of the 7th ACM Symposium on Operating Systems Principles, Pacific Grove, California, 1979.

RICH,E.,KNIGHT,K. Inteligência Artificial. Segunda Edição. MacGraw-Hill. 1991. 722p.

ROBERTSON,G. et al. **Data Mountain: Using Spatial Memory for Document Management**. Proceedings of the 11th annual ACM Symposium on User Interface Software and Technology. San Francisco, Califórnia, 1998.

ROBERTSON,J,ROBERTSON,S. **Mastering the Requirements Process**. ACM Press.Addison-Wesley, 1999.

ROBILLARD,M.P., MURPHY,G.C. **Regaining Control of Exception Handling**. Technical Report TR-99-4. Depto. of Computer Science, University of British Columbia. December 1999.

RODEHEFFER,T.L.,SCHROEDER, M.D. **Automatic Reconfiguration in Autonet**. Proceedings of the Thirteenth ACM Symposium on Operating Systems Principles, Pacific Grove, California, pp.183-197, October 1991.

ROMÁN,M., CAMPBELL ,R.H. **Gaia: An Operating System to Enable Active Spaces**. Submitted to the 9th ACM SIGOPS European Workshop, 2000.

ROSENBERG**,**J. **The MONADS Architecture: A Layered View.** Proceedings of the Fourth International Workshop on Persistent Objects Systems. Martha's Vineyard, Massachusetts, pp.215-225, 1990.

ROSENBERG,J., et al. **Operating System Support for Persistent and Recoverable Computations**. CACM, 39(9),pp.62-69, 1996.

ROSENBLUM,M. et al. **The Impact of Architectural Trends on Operating System Performance.** The 15th ACM Symposium on Operating Systems Principles, December 1995.

ROSENBLUM,M., et al. **Using the SimOS Machine Simulator to Study Complex Computer Systems.**

ACM Transactions on Modeling and Computer Simulation. Vol.7,No.1, pp.78-103, January 1997.

RUIZ,J.A. **Metodologia Científica:Guia Para Eficiência nos Estudos**. Ed.Atlas, São Paulo, 2002.

SAMET,H. The **Quadtree and related hierarchical data structures**.Computing Surveys, 16(2),Jun.,1984.

SANSONNET,J.P.,CASTAN,M.,PERCEBOIS,C.,BOTELLA,D.,PEREZ,J. **Direct Execution of LISP on a List Directed Architecture**. Proceedings of the Symposium on Architectural Support for Programming Languages and Operating Systems. Palo Alto, California, March 1982, pp. 132-139.

SATYANARAYANAN,M.FLINN,J. WALKER,K.R. **Visual Proxy: Exploiting OS Customizations Without Application Source Cod**e. Operating Systems Review. 33(3): 14-18. 1999.

SAULPAUGH.T.,MIRHO,C. **Inside the JavaOS Operating System.** Addison Wesley Longman. 1999.

SCHAAD,R. **Representation and Execution of Situated Action Sequences**. Tese de Doutorado. Universitat Zürich, August 1998.

SCOTT,J.,HOFFMAN,F.,ADDLESEE,M.,MAPP,G.,and HOPPER,A. **Networked Surfaces: a New Concept in Mobile Networking**. Technical Report 2000.9.
Disponível em: <ftp://ftp.uk.research.att.com/pub/docs/att/tr.2000.9.ps.gz>, 2000.

SEGAL,R. **St.Bernard: The File Retrieving Softbot**. Dissertação de Mestrado. Depto. Computer Science and Engineering, University of Washington, Seattle, 1992.

SEIGNEUR,JM., FARRELL,S., JENSEN,C.D. **Secure Ubiquitous Computing Based on Entity Recognition.** Ubicomp2002 Security Workshop, Göteborg, October 2002..

SELTZER,M.,SMALL,C.,SMITH,K. **The Case for Extensible Operating Systems**. Harward Computer Center for Research in computing Technology -Technical Report TR-16-95 Depto. Of Computer Science, Harward University. 1995.

SELTZER,M. et al. **An Introduction to the VINO Architecture**. VINO: The 1994 Fall Harvest. Harvard University Technical Report TR-34-94, 1994.

SELTZER,M. et al. **Dealing With Disaster: Surviving Misbehaved Kernel Extensions**. Proceedings of the 1996 Symposium on Operating System Design and Implementation - OSDI, 1996.

SELTZER,M.,SMALL,C. **Self-monitoring and Self-adapting Operating Systems**. Proceedings of the Sixth Workshop on Hot Topics in Operating Systems,pp.124-129, May 1997.

SENRA,N.C. **Informação estatística: política, regulação , coordenação**. Revista Ciência da Informação – Políticas e Gestão da Informação. V.28, n.2, p.124-125, maio/ago. 1999.

SCHUBERT,F. **A Reflective Architecture for an Adaptable Object-Oriented Operating System Based on C++**. Chemnitz University of Technology. Operating System Group. Germany, May 1997.

SIEWIOREK,D.P., BELL,C.G., NEWELL,A. **Computer Structures: Principles and Examples**. International Student Edition. McGraw-Hill, Japan 1983.

SILVA,E.L., MENEZES,E.M. **Metodologia da Pesquisa e Elaboração de Dissertação**. Florianópolis: Laboratório de Ensino a Distância da UFSC, 2000 118p.

SILVA NETO, A.B. **Competitividade e desempenho competitivo no nível da firma: análise comparativa de conceitos e de indicadores**. Dissertação (Mestrado em Economia) – Programa de Pós-GRADUAÇÃO EM ECONOMIA, UNIVERSIDADE FEDERAL DE SANTA CATARINA, FLORIANÓPOLIS.2000.

SIMITCI,H. **Adaptive Disk Striping for Parallel Input/Output**. Tese de Doutorado. University of Illinois, Urbana Champaign, 2000.

SIRAJ,A.,BRIDGES,S.M. VAUGHN,R.B. **Fuzzy Cognitive Maps for Decision Support in Intrusion Detection Systems**.2001. Disponível em:
<http://www.sc.msstate.edu/~security/iids/publications/nafips_ifsa_2001.htm>.

SIY,P.,CHEN,C.S. **Minimization of Fuzzy Functions**. IEEE Trans. On Computers,v.21,pp.100-102,January 1992.

SKJELLUM,A.,et al. **Systems Administration**. Cluster Computing White Paper Version 2.0. Cap 6. Mark

Baker – Editor. University of Portsmouth, UK. December 2000.

SOVEREIGN,J. **The Power of Posix Thinking. How to simplify porting to the portable system interface**. UnixWorld, July 1992,p93-103.

SOWA,J.F. **Knowledge Representation: Logical, Philosophical, and Computational Foundations**. Brooks/Cole. 2000. 594p.

STANKOVIC,J. **Admission control, Reservation, and Reflection in Operating Systems**. IEEE Bulletin of the Technical Committee on Operating Systems and Application Environments (TCOS), Vol. 10, No. 2,Summer 1998.

STEENSGAARD,B.,JUL,E. **Object and Native Code Thread Mobility Among Heterogeneous Computers**. 15th ACM Symposium on Operating System Principles (SOSP), pages 68-78, Copper Mountain Resort, CO, December 1995.

STEERE,D.C.,et al. **A Feedback-driven Proportion Allocator for Real-Rate Scheduling**. Proc. 3th Symposium on Operating Systems Design and Implementation, New Orleans, Louisiana, February 1999.

STETS,R,J.,HUNT,G.C., SCOTT,M.L. **Component-Based APIs: A Versioning and Distributed Resource Solution**. *IEEE Computer*, Vol. 32, No. 7, pp. 54-61. July 1999. IEEE.

STEVEN,H. **Self-Paging in the Nemesis Operating System**. Proceedings of Usenix Third Symposium on Operating Systems Design and Implementation, February 1999.

STIX,G. **Tempo Real**. Scientific American Brasil. Pp 50-53, Outubro 2002.

SUGERMAN,J.,VENKITACHALAM,G.,Lim, B.**Virtualizing I/O Devices on VMware Workstation's Hosted Virtual Machine Monitor**. Proceedings of the 2001 USENIX Annual Technical Conference Boston, Massachusetts, USA, June 2001.

SUTHERLAND,I.E. **Dispositivos de Entradas e Saídas de Computador**. Scientific American 1966. Computadores e Computação: Textos do Scientific American. Editora Perspectiva. 1977, 332p.

SUTHERLAND,I.E. **Dispositivos de Exibição de Computador**. Scientific American 1970. Computadores e Computação: Textos do Scientific American. Editora Perspectiva. 1977, 332p.

SVAHNBERG,M. **Background Analysis and Design of ABOS, An Agent-Based Operating System**. Dissertação de Mestrado. University of Karlskrona. Ronneby, 1998.

SZTAJNBERG,A. **Flexibilidade e Separação de Interesses para Concepção e Evolução de Sistemas Distribuídos: A Abordagem R-RIO**. Exame de qualificação. COPPE/UFRJ, Julho 1999.

SZYPERSKE, C., GOUGH, J. **The Role of Programming Languages in The Life-Cycle of Safe Systems**. Proceedings of the International Conference on Safety Through Quality, Kennedy Space Center, FL, USA, October 1995.

TAMCHES, A., MILLER,B.P. **Fine-Grained Dynamic Instrumentation of Commodity Operating System Kerne**ls. Proceedings of the 3th Symposium on Operating Systems Design and Implementation, New Orleans, Louisiana, February 1999.

TANENBAUM,A.S. **Structured Computer Organization**. Prentice-Hall, 1984. 465p.

TANENBAUM,A. **Operating Systems:Design and Implementation**.Prentice-Hall,1987.

TANDLER, P. **Software Infrastructure for a Ubiquitous-Computing Environment Supporting Collaboration with Multiple Single- and Multi-User Devices**. Proceedings of UbiComp'01, Atlanta, USA, Lecture Notes in Computer Science, Springer, Heidelberg, 2001.

TIRELO,F.,MAIA,M.A.,DI IORIO,V.O.,BIGONHA, R.S. **Máquinas de Estado Abstratas**, Technical Report 06/99), 32 pages, III Simpósio Brasileiro de Linguagens de Programação, Porto Alegre, RS, Brasil, Maio 1999.

TKACH,D.,PUTTICK,R. **Object technology in application development**. IBM International Technical Support Organization, The Benjamin/Cummings Publishing Company,Inc. 1994. 212p.

WADGE,W.W. **Intensional logic in context**. Intensional Programming II, World Scientific, Singapore, 2000.

VAHDAT, A. et al. **WebOS: Software Support for Scalable Web Services**. Submitted to the Sixth

Workshop on Hot Topics in Operating Systems, Chatham, Massachussetts, May 1997.

VMWARE. **VMWare User's Manual**. Disponível em <http://www.vmware.com>, 2001.

VENNERS, B. **Inside the Java 2 Virtual Machine**. McGraw-Hill, 1999.

VILENSKY,R., ARENS,Y., CHIN, D. **Talking to Unix in English: An Overview of UC**. CACM 17,6, pp.574-593, Junho 1984.

WAHBE, R.,et al. **Software Fault Isolation**. Proceedings of the 14th ACM Symposium on Operating System Principles, Vol.27,pp 203-216,Dec 1993.

WALKER, B.,KEMMERER,R., POPEK,G. **Specification and Verification of the UCLA UNIX Security Kernel**. Proceedings of the 7th ACM Symposium on Operating Systems Principles. December 1979.

WANADOO Home Page. **Study: Epistemological Foundations of the Human Reason**. Disponível em <http://perso.wanadoo.fr/zgmet/indstd.htm>. Acesso em 29/03/2001.

WANG, Z, GARLAN, D. **Task-Driven Computing**. Technical Report, CMU-CS-00-154, School of Computer Science, Carnegie Mellon University, May 2000.

WANT,R. **New Horizons for Mobile Computing**. IEEE Pervasive Computing & Communication Conference. Keynote Address 1. Dallas, Fort Worth,Texas.March 2003.

WATANABE**,** H., DETTLOFF, W., YOUNT, E. A. **VLSI Fuzzy Logic Inference Engine for Real-time Process Control.** IEEE Journal of Solid State Circuits, v.25,n.2,pp.376-382, April 1990.

WATER,J.K. **Microsoft Unveils Windows Server 2003**. Application Development Trends, June 2003, p.15.

WEISER, M., BROWN, J. S. **The coming age of calm technology**. In Denning P. J.,. Metcalfe, R. M (Eds.). Beyond calculation: The next fifty years of computing. New York, NY: Copernicus. 1998.

WEISER,M. **The computer for the twenty-first century**. Scientific American, 265(3):94-104, Sept. 1991.

WEXELBLAT,A.,MAES,P. **Footprints: History-Rich Web Browsing**. RIAO'97 - Third International Conference on Computer-Assisted Information Retrieval.Quebec, Canadá, June 1997

WILKE,W. BERGMANN,R. **Adaptation with the INRECA System**. ECAI'96 Workshop: Adaptation in CBR. 1996.

WILKINS,D.E. **Domain-independent Planning: Representation and Plan generation**. Elsevier Science Publishers. Artificial Intelligence (22) pp269-301, 1984.

WOOLDRIDGE,JENNINGS. **Intelligent Agents: Theory and Practice**.(eds.). New York: Springer, 1995, 407p.

WRIGHT,K. **Os tempos de nossa vida**. Scientific American Brasil. Out. 2002 pp70-77.

WULF,W.A. et al. **HYDRA: The Kernel of a Multiprocessor Operating System**. Communication of the ACM, V.17, p337-345, 1974.

YOKOTE,Y. **The Apertos Reflective Operating System: The Concept and Its Implementation**. Proceedings of OOPSLA'92, ACM Sigplan Notices, v. 27, pages 414–434, 1992.

YU,H.,VAHDAT,A. **The Costs and Limits of Availability For Replicated Services**. Proceedings of UbiComp'01, Atlanta, USA, Lecture Notes in Computer Science, Springer, Heidelberg, 2001.

ZAIANE,O.R.,JIAWEI,R. **Resource and Knowledge Discovery in Global Information Systems: A Preliminary Design and Experiment**. Proceedings of the KDD-95, pages 331-336,1995.

ZADEH, L.A.,YAGER,R.R. **Fuzzy Sets, Neural Networks, and Soft Computing**. VNR, New York, 1994.

ZANCANELLA, L.C.,NAVAUX,P.O.A. **AURORA: Um Sistema Operacional Orientado a Objetos Para Arquiteturas Multiprocessadoras**. Anais XX SBAC-PAD. XIII Congresso da SBC, Florianópolis, 1993.

ZDANCEWIC, S.ZHENG,L.,NYSTROM,N. MYERS,A. **Untrusted Hosts and Confidentiality: Secure**

Program Partitioning. Proceedings of UbiComp'01, Atlanta, USA, Lecture Notes in Computer Science, Springer, Heidelberg, 2001.

ZHAO, B.Y., KUBIATOWICZ, J.D., JOSEPH. A. D. **TAPESTRY: An Infrastructure For Fault-Resilient Wide-Area Location and Routing**. Technical Report UCB//CSD-01-1141, U. C. Berkeley, April 2001.

ZIEGLER,B.P.,SARJOUGHIAN,H.S. **DEVS Component-Based M&S Framework: An Introduction**. Proceedings of 2002 AI, Simulation and Planning in High Autonomy systems, AIS'2002. Lisbon, Portugal, April 2002.

ZOMAYA,A.Y.,CLEMENTS,M.,OLARIU,S. **A Framework for Reinforcement-Based Scheduling in Parallel Processor Systems.** IEEE Transactions on Parallel and Distributed Systems. V9, N3,p249-260,Mar 1998.

10 APÊNDICES

Esta seção do trabalho contém os seguintes apêndices:

- Apêndice 1 – Call for papers ACM SIGOPS 2003.
- Apêndice 2 – Descrição detalhada das arquiteturas de núcleos de sistemas operacionais.
- Apêndice 3 – Exemplo de log de erro no sistema (DrWatson –Windows).
- Apêndice 4 – Exemplo de log obtido através de device driver (Windows).
- Apêndice 5 – Exemplo de log obtido através de Sar (Unix).
- Apêndice 6 – Modelo de árvore de decisões para algoritmos.
- Apêndice 7 – Modelo de árvore de decisões para redações.
- Apêndice 8 – Planilha .

10.1 Apêndice 1 – Call for papers ACM SIGOPS 2003

SIGOPS

19th ACM Symposium on Operating Systems Principles

October 19-22, 2003
The Sagamore,
Bolton Landing (Lake George), New York

General chair Michael L. Scott, University of Rochester
scott@cs.rochester.edu (585)-275-7745
Program chair Larry Peterson, Princeton University
llp@cs.princeton.edu, (609)-258-6077
Sponsored by ACM SIGOPS

Authors are invited to submit papers to the 19th Symposium on Operating System Principles (SOSP) reporting on original research related to the design, implementation, analysis, evaluation, and deployment of operating systems. We seek submissions of high quality that significantly further the knowledge and understanding of the systems community. In keeping with SOSP tradition, we will favor work that explores new territory, continues a significant research dialog, or reflects on experience with practical applications of the community's knowledge. Papers of particular merit will be forwarded to ACM Transactions on Computer Systems for possible publication in a special issue.

The symposium attracts attendees with diverse backgrounds. We solicit papers in the traditional core of the OS field, as well as in the interfaces to areas such as computer architecture, networking, programming languages, and databases. Topics of interest include, but are not restricted to:

High availability	Mobile computing
Scalability and performance	Ubiquitous computing
Security	Power management
File systems	Sensor Networks
I/O architectures	Web support
Overlay networks	Multimedia systems
Transactional support	Empirical studies

Figura 78 – Chamada de trabalhos para o congresso na área de Sistemas Operacionais sob a chancela da ACM em 2003.

10.2 Apêndice 2 – Descrição detalhada das arquiteturas de núcleos de sistemas operacionais

O presente apêndice apresenta uma revisão das diversas abordagens de construção de núcleos de sistemas operacionais identificando-se os principais modelos de arquitetura, suas características, vantagens e desvantagens com as respectivas justificativas citadas na literatura. Conforme referido no capítulo2, os primeiros sistemas operacionais começaram a surgir a partir da segunda geração de computadores, em meados da década de 50 e início da década de 60, mais como um produto de fatoração de rotinas de *run-time*[99] comuns do que efetivamente resultado de intenção em desenvolvê-los.

A seguir relacionam-se as principais abordagens estruturais dos sistemas operacionais descritas na literatura:

- Modelos monolíticos;
- Modelos em camadas (ou anéis)
- Modelos hierárquicos;
- Modelos baseados em micronúcleos (*microkernel*);
- Modelos cliente-servidor;
- Núcleos coletivos:
    - *Library operating systems;*
    - *Modelos exonúcleo (exokernel);*
    - *Modelos cache kernel;*
    - *Modelos nanonúcleo (nanokernel);*
- Modelos de máquinas virtuais;
- Modelo baseado em objetos;
- Modelos baseados em reflexão computacional;
- Modelos baseados em conhecimento;
- Modelos híbridos.

As próximas seções apresentam as principais características destas abordagens estruturais.

[99] Procedimentos necessários durante a execução de um programa, tais como: bibliotecas de manipulação de arquivos, de tratamento de números com ponto flutuante, etc.

10.2.1 Estruturas Monolíticas

Os antigos projetos de núcleos monolíticos foram escritos como uma agregação de toda a funcionalidade necessária para permitir a utilização do hardware, como processos, gerenciamento de memória, multiprogramação, comunicação de processos, acesso a dispositivos, sistema de arquivos e protocolos de rede. Projetos mais recentes (principalmente **Unix**) possuem uma arquitetura modular, o que oferece a possibilidade de adição/remoção de serviços em tempo de execução (run-time). Sistemas desta geração geralmente eram concebidos para atuarem como servidores em situações de execução contínua e não apresentavam muita flexibilidade para inserção/remoção de periféricos. Por outro lado, esta geração de sistemas apresentava um alto grau de estabilidade (DUMON, 1997).

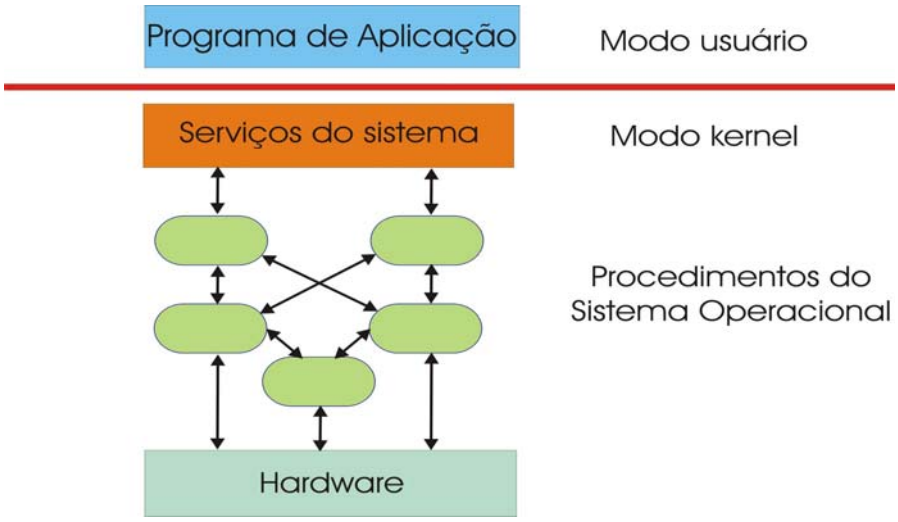


Figura 79 –Modelo de núcleo monolítico.

FONTE: Adaptado de Ahl (2001).

De acordo com Dearle e Hulse (2000), muitos consideram os projetos de núcleos monolíticos (Figura 79) como os precursores das modernas arquiteturas de sistemas operacionais. A premissa básica nesta abordagem é abstrair detalhes de hardware e prover abstrações de mais alto nível. Neste enfoque, o sistema consiste de um conjunto de procedimentos que podem chamar qualquer outro procedimento. O sistema não é suficientemente modular para permitir a atualização seletiva de componentes sem necessidade de atualização de outros componentes. Outro aspecto importante refere-se ao fato de que, como uma porção significativa do código do sistema operacional executa no mesmo espaço de memória, é possível que um componente do sistema venha a corromper

dados usados por outros componentes. Como conseqüência disto, o aspecto de manutenção deste tipo de abordagem geralmente é problemático.

Este enfoque foi usado nos sistemas **MS-DOS** e em algumas versões de **Unix**[100] (AHL,2001), **OS/360**, **VMS** e **VME** (DEARLE e HULSE,2000).

Algumas versões de **Unix** adotam esta abordagem, implementando todos os serviços no núcleo. Isto possibilita um alto grau de estabilidade e rapidez, mas é tecnicamente desatualizado. Alguns melhoramentos foram incorporados nesta abordagem para torná-la mais moderna. Por exemplo, o conceito de módulo permite que partes de código possam ser conectadas no núcleo, possibilitando estender a funcionalidade do núcleo da mesma forma que os micronúcleos o fazem. No entanto, estes módulos também executam em modo supervisor, o que implica na necessidade de um alto grau de estabilidade.

O sistema operacional **Linux** possui um projeto de núcleo monolítico. O modelo de núcleo adotado é modular, o que permite a carga de módulos em tempo de execução. No entanto, todos os módulos executam em modo supervisor (mesmo aqueles que não necessitam deste privilégio) e não são protegidos dos demais (nem o próprio núcleo), o que caracteriza um problema de segurança (MACHADO e MAIA,2002).

O aspecto de maior impacto no projeto de núcleos monolíticos está relacionado à rigidez dos projetos, o que compromete operações de expansão (para fazer frente a novas demandas), correção e testes do sistema.

10.2.2 Estruturas Baseadas em Camadas

Em função destes aspectos relacionados à estrutura monolítica e também à medida que a complexidade do ambiente aumentou, tornou-se clara a necessidade do desenvolvimento de novas propostas.

Segundo Montez (1995), o primeiro sistema em camadas foi construído por Dijkstra e seus estudantes. A estrutura do sistema foi publicada no primeiro Simpósio em Princípios de Sistemas Operacionais da ACM, em 1967. O sistema **THE** era organizado em sete níveis. Cada nível consistia de uma coleção de objetos abstratos e um conjunto de regras

[100] Apesar do **UNIX** ter sido originalmente concebido sobre a estrutura monolítica (o Linux ainda a utiliza), atualmente há implementações comerciais que utilizam outros tipos de estrutura, como por exemplo o sistema **AIX SL** (AHL,2001).

controlando a interação entre eles. As regras eram codificadas em procedimentos do sistema operacional denominados “operações primitivas” (COMER,1984).

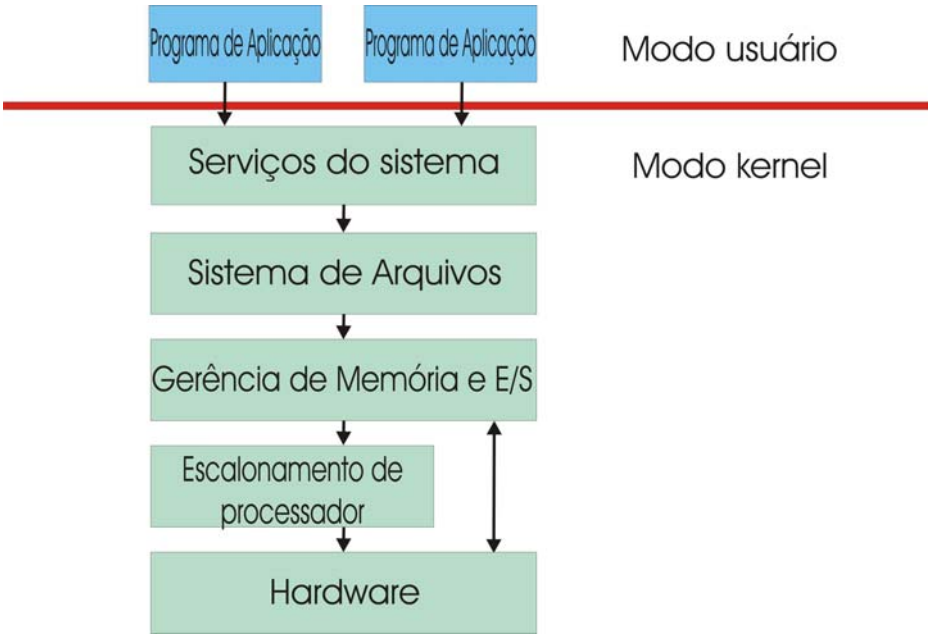


Figura 80 – Modelo em camadas.

FONTE: Adaptado de Ahl (2001).

O modelo em camadas (Figura 80) introduz o conceito de programação modular e pode ser considerado como uma generalização do modelo anterior. É projetado como um conjunto de camadas sobrepostas onde cada uma provê um conjunto de funções para a camada superior e é implementada em termos das funções providas pela camada inferior (ZANCANELLA e NAVAUX,1993). Não há como um nível inferior enviar comandos para um nível superior.

Segundo Machado e Maia (2002), a maioria dos sistemas comerciais utiliza o modelo de 2 camadas onde existem os modos de acesso usuário e kernel (ex. **Unix** e **Windows 2000**).

Este enfoque apresenta como característica a facilidade de manutenção, uma vez que todo um nível pode ser substituído sem afetar o comportamento dos níveis inferiores e superiores. Segundo Nicol (1987), os projetos baseados nesta estrutura apresentavam deficiências principalmente devido ao fato de que as camadas inferiores precisavam ser genéricas (com natural perda de performance).

Além disso, segundo Zancanella e Navaux (1993), ainda que atraentes, projetos baseados em camadas complicam mais a construção de sistemas complexos, tendo em vista que nem sempre é possível garantir a existência de um processo restrito a uma camada.

Comer (1984) indica que a estrutura em camadas foi aplicada no sistema operacional **VENUS**, em 1972, e no projeto **PSOS** (*Provably Secure Operating System*), em 1975.

Apesar de apresentar uma série de facilidades do ponto de vista de construção, estas características permanecem isoladas dentro do contexto do software do sistema, não podendo ser utilizadas pelos programadores de aplicação.

10.2.3 Estruturas Hierárquicas

A estrutura hierárquica, embora semelhante à estrutura anterior, apresentava o diferencial de permitir a transposição de níveis da hierarquia funcional. Outro aspecto importante é que a estrutura hierárquica é baseada em funções e não em processos. Segundo Zancanella e Navaux (1993), "cada nível é caracterizado por um conjunto de funções estaticamente identificadas e conhecidas. Além disso, o conceito de módulo é usado para expandir vários níveis por meio de um único módulo. Na estrutura em camadas, o conceito de módulo é utilizado para a divisão de um nível em vários módulos distintos" .

Um exemplo de utilização desta estrutura é o sistema Pilot (DALAL et al.,1980). O sistema e todas as aplicações são escritos em Mesa. Isto garante a proteção do sistema sem a necessidade de uma MMU. Contudo, os mecanismos de proteção eram orientados para um ambiente monousuário onde erros de programação são mais graves que ataques externos (BACK,2001). O relacionamento entre os maiores componentes é ilustrado conforme apresentado por Redell et al. (1979). Segundo Zancanella e Navaux (1993), uma característica da estrutura hierárquica e que pode ser vista na Figura 81, é o entrelaçamento de módulos, como ocorre com os módulos do sistema de arquivos e memória virtual.

A contribuição do sistema Pilot foi a integração, em um sistema relativamente compacto, de uma série de idéias que anteriormente haviam sido propostas individualmente. A combinação de streams, comunicação via pacotes, memória virtual mapeada em arquivos, suporte à programação concorrente e uma linguagem modular de alto nível, caracterizava o ambiente com relativamente poucas limitações no tamanho e complexidade de programas que poderiam ser suportados (REDELL et al.,1979).

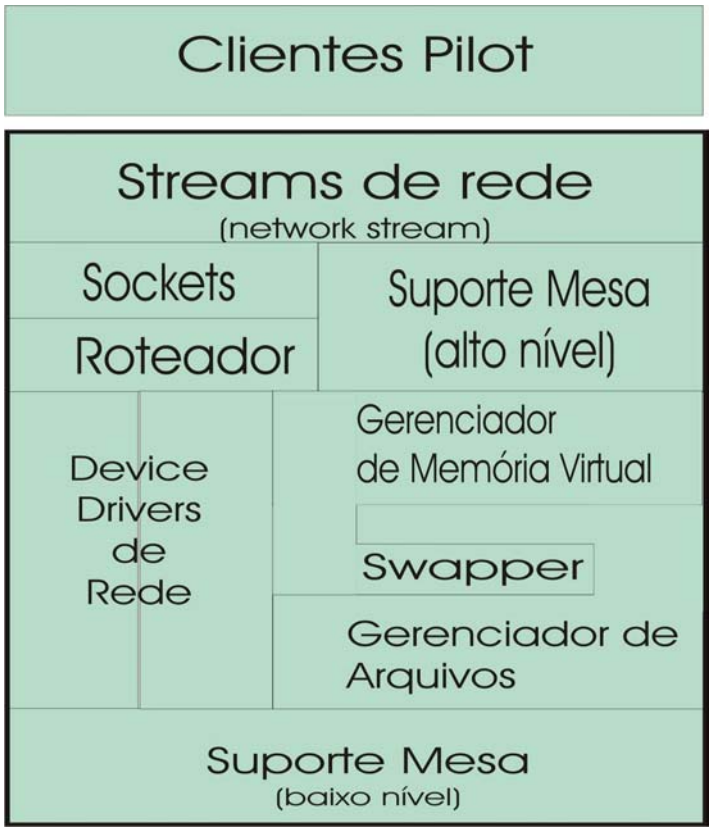


Figura 81 -Organização do sistema Pilot.

FONTE: Adaptado de Zancanella e Navaux (1993,pág 24).

Como benefícios deste esquema pode-se ressaltar (Zancanella e Navaux,1993):

- Modularização.
- Reusabilidade.
- Facilidade para manutenção.

Ainda segundo Zancanella e Navaux (1993), tais benefícios são exclusivamente internos ao sistema operacional, o que se torna uma limitação, visto que seria desejável uma visão uniforme dos objetos a serem aplicados tanto fora como internamente ao sistema".

10.2.4 Estruturas Baseadas em Micronúcleos

Segundo Liedtke (1995), sistemas baseados em micro-núcleos já eram construídos mesmo antes do próprio termo ter sido introduzido por Brinch-Hansen (1970) e Wulf et al. (1974). A proposta de micronúcleo está baseada na idéia de implementar fora do núcleo todos os serviços possíveis. Isto reduz o tamanho do código do núcleo e aumenta a flexibilidade do sistema como um todo. Serviços de mais alto nível, tais como: sistemas de

arquivos, device drivers, gerência de processos e mesmo parte das funções de gerência de memória, podem ser construídos sob a forma de processos que executam sobre o micro-núcleo (MULLENDER et al., 1990).

Esta abordagem conduz a uma organização dos processos na forma de clientes e servidores. O núcleo, executando em modo supervisor, encarrega-se de entregar as mensagens para o servidor responsável pelo seu atendimento. Após a execução do serviço, o núcleo encarrega-se de retornar os resultados para o cliente através de outra mensagem. Neste modelo, os componentes do sistema operacional podem ser pequenos e como cada processo servidor executa como diferentes processos em modo-usuário, um determinado servidor pode falhar sem comprometer o restante do sistema operacional.

As vantagens deste enfoque são as seguintes:

- Uma interface de micronúcleo conduz a uma estruturação mais modular do sistema;
- Os servidores podem utilizar os mecanismos implementados no núcleo, da mesma forma que qualquer outro programa de aplicação;
- Um mau-funcionamento de um servidor pode ser isolado, não comprometendo os demais serviços do sistema;
- O sistema torna-se mais flexível e configurável. É possível implementar diferentes estratégias e
- em diferentes servidores e tolerar sua coexistência simultânea;
- Adaptabilidade ao uso em sistemas distribuídos, visto que, se um cliente comunica-se com um servidor pelo envio de uma mensagem, ele necessariamente não necessita conhecer onde a mensagem é manipulada, se na própria máquina ou em uma máquina remota. (ZANCANELLA e NAVAUX,1993).

Contudo, a questão chave em projetos de micronúcleo é a eficiência da implementação de primitivas de comunicação entre processos (IPC). Apesar desta característica ser o ponto forte da abordagem, representa também o ponto de gargalo do sistema como um todo.

Segundo Dumon (1997), deve-se fazer uma distinção entre duas gerações de micronúcleos. A primeira, representada pelo núcleo **Mach** (RASHID et al.,1989), caracterizava-se por ser pesada e apresentar uma interface (API) ambígua. Em particular, o Mach apresenta como característica o fato de que todas as abstrações do núcleo são representadas como objetos. Assim, o **Mach** é totalmente orientado a objetos, apesar de ter sido implementado completamente em linguagem **C**. Segundo Tanenbaum (1992,pág.639)

apud Montez (1995), o micronúcleo, seus sub-sistemas e os processos de usuários compõe um sistema de três camadas conforme apresentado na Figura 82.

A segunda geração segue um caráter mais purista e caracteriza-se por apresentar um tamanho bem reduzido, fornecendo somente abstrações estritamente necessárias e com uma interface (API) simples e não ambígua. Exemplos desta segunda geração são: **L4** e **QNX** (DUMON,1997). **Fiasco** (HOHMUTH,1998) é um novo núcleo compatível com **L4**. O núcleo do sistema **Fiasco** pode sofrer preempção[101] praticamente em qualquer ponto de sua execução. Esta característica possibilita tempos de resposta muito pequenos para *threads*[102] de alta prioridade, o que o torna adequado ao suporte a aplicações de tempo-real.

Em **BeOS** (1998) também é descrita uma abordagem de micronúcleo semelhante àquela proposta no núcleo Mach. No entanto, o modelo é baseado em um grande número de threads, o que o torna adequado para suporte a aplicações multimídia. O sistema apresenta uma API simples, possui um sistema de arquivos de 64 bits e permite acesso direto ao sistema gráfico. Além disso, apresenta uma interface POSIX[103] (SOVEREIGN,1992; IEEE,1996), contemplando todos os utilitários tradicionais do sistema **Unix**.

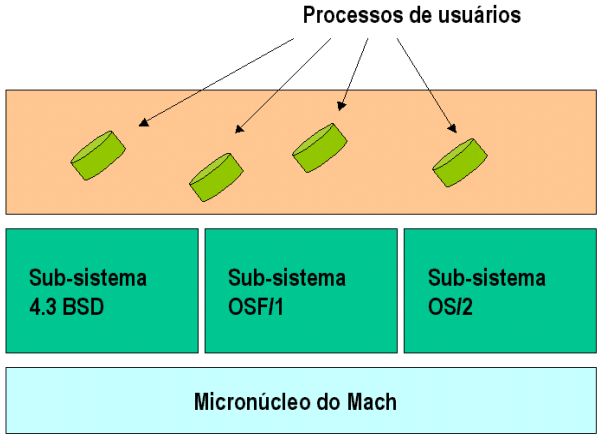


Figura 82 - O micronúcleo do Mach e seus sub-sistemas.

FONTE: Adaptado de Montez (1995,pág.17).

[101] Segundo Oliveira,Carissimi e Toscani (2000), um algoritmo é dito ser preemptivo se o processo em execução puder perder o processador para outro processo, por algum motivo que não seja o término do seu ciclo de processador. Se o processo só libera o processador por vontade própria (chamada de sistema), então este algoritmo é dito não-preemptivo.

[102] Simplificadamente, pode-se considerar uma thread como sendo um procedimento (ou conjunto de procedimentos) de um programa que pode ser disparado como unidades de escalonamento individuais.

[103] POSIX (*Portable Operating System Interface*) é um conjunto de padrões internacionais (ISSO/IEC 9945-1:1996(E), ANSI/IEEE Std 1003.1 1996 Edition) que define uma interface com sistemas abertos. Eles definem a interface das aplicações com serviços básicos do sistema, tais como E/S, acesso a arquivos e gerência de processos. Além disso, estabelece um formato padrão para troca de dados.

Um último exemplo de uma variação do conceito de micronúcleo é apresentado pelo sistema **QNX** (LEROUX,2001). Este sistema caracteriza-se por apresentar um núcleo muito pequeno (32Kb), o que permite sua utilização inclusive em sistemas embarcados (*embedded*). O núcleo implementa somente serviços de multitarefa, tratamento de interrupções, comunicação entre processos (IPC) e gerenciamento de memória. Todos os demais serviços do sistema, bem como as aplicações, são disparadas como processos com seu próprio espaço de endereçamento.

Assim, uma falha em uma aplicação ou módulo não afeta o restante do sistema. Esta estratégia permite inclusive a adição de um novo módulo contendo atualizações enquanto o módulo antigo ainda está em execução. Neste caso, as novas requisições passam a ser atendidas pelo novo módulo e, quando o modulo antigo terminar seus atendimentos, o sistema automaticamente encerra sua execução (DUMON,1997).

10.2.5 Estruturas Configuráveis ou Adaptáveis

A abordagem de micronúcleo introduz o conceito de núcleos adaptáveis (*customizable kernels*), os quais podem ser dinamicamente especializados para atender a requisitos de performance e funcionalidade das aplicações. A motivação que conduziu a esta abordagem foi o fato de que as abordagens anteriores não permitiam a construção de sistemas que atendessem a todas as demandas de serviços e performance requisitados pelas aplicações.

Por serem estáticas, as abordagens anteriores dificultavam a adaptação e/ou substituição de módulos do núcleo tendo em vista atender a estes requisitos (SELTZER,SMALL e SMITH,1995). Uma ressalva deve ser feita relativamente aos projetos monolíticos. Neste caso, a aplicação pode desviar chamadas de sistema, instalando filtros nas mesmas, e re-direcionando o fluxo de execução apropriadamente quando necessário. Esta estratégia era amplamente adotada no sistema operacional **MS-DOS** (ONEY,1999) através da substituição do endereço dos procedimentos de atendimento de interrupções, alterando-se a tabela de vetores de interrupção do sistema.

As características principais deste tipo de abordagem são (SELTZER,SMALL e SMITH,1995):

- Eficiência;

- Correção e;
- Flexibilidade.

O aspecto de eficiência relaciona-se ao fato de a estrutura permitir as extensões sem causar overhead além daquele provocado pelo código da própria extensão. O aspecto de correção está relacionado ao fato de que uma extensão, em tese, não deve comprometer a integridade do sistema como um todo. O grau de granularidade dos filtros (*hooks*[104]) facilita a instalação de código do usuário em substituição a parte de código do núcleo, elevando o grau de flexibilidade do sistema. Um alto grau de granularidade caracteriza um sistema bastante flexível, mas também introduz uma grande possibilidade de falhas (SELTZER,SMALL e SMITH,1995).

Em sistemas tradicionais, este problema é minimizado, tendo em vista que a integridade é garantida pela coesão dos módulos. Código não confiável pode ser isolado através da combinação de mecanismos de proteção de memória e verificação de argumentos em tempo de execução. Extensões ao sistema, tais como device drivers ou sistemas de arquivos dinamicamente carregados, são geralmente confiáveis em sua totalidade. Sistemas extensíveis, ao contrário, apresentam um grande número de interfaces não confiáveis. Estas extensões não confiáveis devem ser isoladas, de modo a não comprometer o sistema (SELTZER, SMALL e SMITH,1995).

Contudo, não é possível implementar este mecanismo somente através de proteção de memória. Isto porque um código instalado no núcleo pode, por exemplo, entrar em um loop infinito, ou implementar um algoritmo falho de comunicação entre processos. Esta característica conduz à necessidade de verificação antecipada e em tempo de execução, tendo em vista garantir a integridade do sistema (SELTZER,1996).

A questão chave nesta abordagem, portanto, está em determinar em que parte do código do sistema são tomadas decisões sobre políticas, de modo a expor esta porção de código de tal forma que a funcionalidade do sistema possa ser estendida. Segundo Engler e Kaashoek (1995), a comunidade de sistemas operacionais não possui uma definição rigorosa para política ou mecanismo. No sentido mais geral, política refere-se às metas de um sistema computacional (ex: quais recursos proteger). No contexto de sistemas operacionais

[104] *Hooks* são pontos dentro de um procedimento (ou programa) em que é permitido ao programador indicar o endereço de um procedimento particular a ser chamado quando determinada situação ocorrer. O mecanismo de substituição/instalação de tratadores de interrupção do sistema MS-DOS é um exemplo deste tipo de mecanismo.

extensíveis, política refere-se ao algoritmo usado para tomar decisões de segurança ou gerenciamento de recursos, enquanto mecanismo refere-se à infra-estrutura (machinery) usada para implementar uma política em particular. Por exemplo, uma política de paginação de memória virtual pode ser a de evitar enviar páginas *least recently used pages* (LRU) para o disco. As estruturas de dados utilizadas para implementar esta solução (tais como tabela de páginas e lista de páginas ordenada), seriam classificados como mecanismo.

Segundo Zancanella e Navaux (1993), a separação entre políticas e mecanismos:

> *"é na verdade uma variante da estrutura hierárquica. Políticas são definidas em níveis externos ao núcleo, enquanto mecanismos internos ao núcleo provêem um conjunto de primitivas que visa permitir a definição e implementação destas políticas, significativamente diferentes, ao nível mais externo (nível de usuário). Esta separação acrescenta ao sistema a flexibilidade de permitir a troca da política sem a necessidade de modificação dos mecanismos básicos".*

Segundo Cowan (1997), há três critérios que podem ser utilizados para classificar os sistemas adaptáveis, quais sejam:

- Quem implementa a adaptação: o sistema ou a aplicação. Uma possibilidade é a aplicação explicitamente solicitar um procedimento de adaptação. A outra é o sistema automaticamente adaptar-se em função das circunstâncias do momento. Alem disso, deve-se considerar se o procedimento de adaptação é transparente para os programadores de aplicações.
- Quando adaptar: em tempo de compilação e/ou ligação, de carga da aplicação ou em tempo de execução. Quanto mais tarde, mais informações estão disponíveis sobre as quais as decisões podem ser baseadas. Contudo, adaptações tardias apresentam um custo maior, tendo em vista que este custo não pode ser amortizado através de múltiplas chamadas do serviço ao sistema que está sendo especializado;
- O que é adaptado: este critério está associado a como a funcionalidade é agregada. Uma nova funcionalidade pode ser caracterizada por uma nova implementação de uma interface existente; ou um nível de software criando uma nova interface sobre uma já existente. Ou, ainda, uma alteração completa na funcionalidade de uma interface.

Um exemplo deste enfoque é adotado no sistema **SPIN** (BERSHAD et al.,1994). O sistema é projetado para permitir o download de código executando em modo usuário para o núcleo, de modo a aumentar a performance de uma aplicação. Este sistema consiste de um conjunto básico de serviços tais, como gerência de memória e escalonamento, e um conjunto de extensões que permitem a configuração do comportamento do sistema. Esta adaptação é obtida através da carga das extensões no núcleo, onde as mesmas são integradas à infra-estrutura existente. Contudo o sistema **SPIN** não controla o heap ou outros recursos usados por suas extensões e nem sempre consegue desconectar ou desativar uma extensão (BACK, 2001).

Um projeto derivado do **SPIN** é apresentado por Fiuczynski et al. (1998) no sentido de separar partes do código da aplicação e, ao invés de instalar extensões no núcleo do sistema, assentá-las em dispositivos inteligentes com processador e memória dedicados. O projeto denomina-se SPINE e os autores apresentam as seguintes considerações com relação aos aspectos de extensibilidade:

> *"...em um sistema de multiprocessamento assimétrico, como em SPINE, o desenvolvimento de aplicações remete a algumas questões sérias como: (i) quando faz sentido dedicar o esforço de um programador para dividir uma aplicação em módulo principal e extensões ao núcleo (ou aos dispositivos inteligentes); (ii) como pode um pequeno conjunto de módulos residindo em dispositivos de E/S ser utilizado pelas demais aplicações?; (iii) Quais funções podem ser descarregadas no processador de E/S do dispositivo e quais devem permanecer residentes no computador hospedeiro?".*

Segundo Back (2001),

> *"Como as extensões são escritas em Modula-3 (linguagem type-safe), o acesso das extensões ao núcleo podem ser controladas. No entanto, é impossível controlar os recursos usados por determinada extensão. Além disso, para evitar ataques de DoS, certos procedimentos de uma extensão podem ser declarados como efêmeros (ephemeral), o que implica que eles não podem chamar procedimentos que também não tenham sido declarados desta forma".*

O sistema **VINO** (SELTZER,1994), é um outro exemplo de núcleo estruturado de forma a permitir que a funcionalidade do sistema operacional possa vir a ser modificada ou estendida em passos pequenos e incrementais. Segundo os autores, se a interface que implementa as políticas do sistema possui uma granularidade suficientemente fina, então aquele modelo também adequadamente suporta extensões ao sistema operacional. Segundo Back (2001), as extensões (*grafts*) funcionam de forma similar ao conceito de transações. No caso de uma extensão abortar, as alterações podem ser desfeitas. Embora o conceito seja eficaz, sua implementação é muito complexa e apresenta comprometimento de performance.

Segundo Seltzer (1996) o enfoque de núcleo configurável apresenta alguns problemas, quais sejam:

- Garantir a segurança do núcleo contra algum código que possa provocar falhas é uma atividade complexa, uma vez que o núcleo deve evitar que uma aplicação monopolize a utilização de determinados recursos comprometendo o sistema como um todo.
- A garantia contra o mau uso de memória através da:
  - Leitura de dados inapropriados como registradores de dispositivos, dados de outros usuários;
  - Escrita em áreas inapropriadas; e
  - Execução de instruções erradas (privilegiadas).

Todas as técnicas impõem alguma complexidade ao núcleo, uma vez que um passo de verificação deve ser executado. Em algum momento esta característica vai apresentar conflito de requisitos com a idéia de substituição das abstrações por blocos de construção independentes de política. Além disso, o programador tem que se preocupar com a identificação de que módulos do núcleo comprometem a performance de sua aplicação e como substituí-los ou alterá-los de forma a se adequarem às necessidades da aplicação que está sendo desenvolvida.

10.2.6 Estruturas Baseadas em Núcleos Coletivos

Segundo Montz (1995), a estrutura de núcleos coletivos é uma variação do modelo de micronúcleo. Neste modelo, o núcleo é construído como uma coleção de processos independentes. A proposta deste tipo de estrutura é eliminar a noção de que um sistema operacional deve:

- Prover abstrações sobre as quais as aplicações são construídas;
- Suportar a multiplexação segura do hardware; e
- Suportar as interações entre processos que implementam os serviços do sistema operacional.

Todas as aplicações primitivas básicas e serviços podem implementar as tradicionais abstrações de sistemas operacionais, especializadas para cada necessidade. Dessa forma, é possível fazer com que todas as decisões sobre política sejam tomadas em modo usuário. Assim, o núcleo deve conter somente os mecanismos que devem ser implementados com segurança. A maior parte da funcionalidade do sistema é implementada ao nível do usuário e pode ser ajustada independentemente para cada aplicação. Aplicações que não apresentam requisitos particulares podem ser ligadas às bibliotecas padrão do sistema. Aquelas que apresentam requisitos diferentes podem implementar políticas específicas para se adequarem a estes requisitos (MONTZ,1995).

Um outro aspecto importante é destacado em Nagle (1994), relativamente à questão do custo de um projeto de sistema operacional versus custo do porte do código para outra plataforma de hardware.

Segundo Nagle (1994),

> *"Muitos projetos recentes tais como Windows NT, Mach 3.0, Chorus, System V e Sprite são projetados para minimizar estes custos suportando múltiplas APIs. Contudo, as facilidades adicionais destes projetos possuem um custo associado: são mais lentos que os sistemas tradicionais".*

Em função disto, a abordagem de núcleos coletivos pode contribuir para minimizar estes aspectos. Segundo Zancanella e Navaux(1993), este modelo de estrutura "é o atual estado da arte para o projeto de sistemas operacionais, explorando de alguma forma as vantagens da técnica estruturada."

Além disso, o modelo apresenta o benefício da separação do núcleo em duas partes: mecanismos que manipulam recursos do sistema (tais como memória física, espaços de endereçamento, etc) e o modelo de programação que auxilia os programadores a usar efetivamente o sistema (ZANCANELLA,1993).

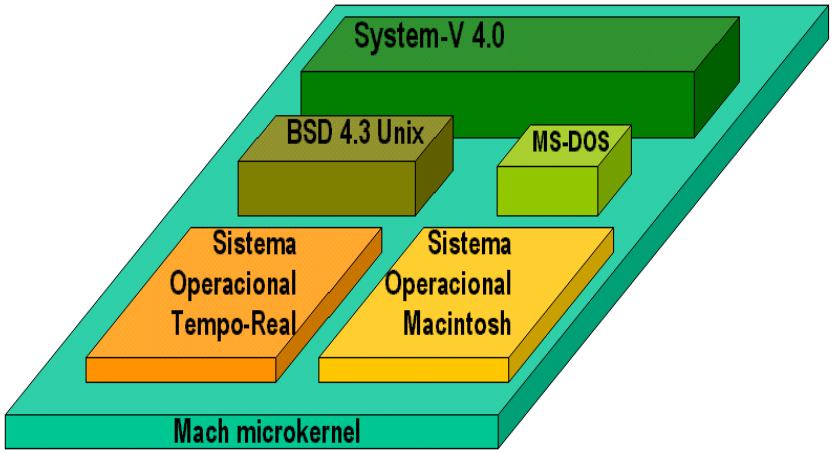


Figura 83 - Estrutura de núcleo coletivo.
FONTE: Adaptado de Zancanella (1993,pág. 27).

Pode-se citar como exemplo os sistemas **Chorus** (HERRMANN,1988), **V-Kernel** (CHERITON,1984), **Mach** (BLACK,1986) e **Amoeba** (MULLENDER,1990).

10.2.7 Estruturas Baseadas em *Library Operating Systems*

A linha dos *library operating systems* (DEARLE,2000) caracteriza-se como uma variação do modelo de micronúcleo onde grande parte da funcionalidade do sistema operacional é assentada em bibliotecas (LibOS) implementadas em modo usuário e específicas para cada aplicação. Há duas correntes nesta linha, quais sejam:

- Exonúcleos (*exokernels*) e
- *Cache kernels.*

Segundo Engler e Kaashoek (1995), a diferença entre *Cache Kernel* e *Exokernel* está centrada em aspectos filosóficos. O *cache kernel* concentra-se primariamente em confiabilidade (*reliability*), ao invés de preocupar-se em exportar com segurança os

recursos de hardware para as aplicações. Em função disto, esta é uma proposta limitada a estruturas baseadas em servidores. A seguir são detalhadas as duas abordagens.

10.2.7.1 Estrutura em Exonúcleo

Os projetos de exonúcleo partem do princípio de que a criação de abstrações sobre o hardware é um enfoque errôneo para a construção de sistemas operacionais. Isto porque, independentemente do nível de abstração fornecido por um sistema operacional, sempre haverá alguma classe de aplicação para a qual as abstrações serão inadequadas. Em função desta reflexão, ao invés de prover todo um conjunto de abstrações, um exonúcleo exporta diretamente os recursos de hardware. A única responsabilidade do núcleo é prover acesso multiplexado e seguro aos recursos de hardware (KAASHOEK,1995).

Segundo Engler e Kaashoek (1995), projetos de sistemas operacionais baseados em exonúcleo exportam uma interface com o hardware de muito baixo nível, permitindo a cada aplicação construir suas próprias abstrações de sistema operacional. Se, por um lado, esta facilidade remove as implicações de tratar com as abstrações inadequadas (*sub-optimal*) que os sistemas tradicionais impõem às aplicações, por outro há o potencial consumo de memória e disco, uma vez que as aplicações, geralmente, são ligadas de forma estática ao código dos LibOS.

Segundo Kaashoek (1995), o sistema **Aegis** é construído sobre uma estrutura que permite extensibilidade com uma granularidade bastante fina. Um núcleo de baixo nível (exonúcleo) provê somente abstrações do hardware. O restante da funcionalidade do núcleo é implementada em bibliotecas ao nível de usuário. Como a maioria das decisões são implementadas nesta camada, as aplicações podem mais adequadamente selecionar aquelas políticas que melhor se ajustam às necessidades específicas de cada uma. Neste sistema os recursos exportados incluem: páginas de memória física, fatias de tempo do processador, blocos de disco, entradas na *Translation Look Aside Buffers* (TLB) e eventos de interrupção. Uma conseqüência direta disto é que uma interface de exonúcleo não é portável.

Além disso, um exonúcleo permite que programas do usuário sobrescrevam o código exportado pelo sistema e pelo próprio núcleo. Isto possibilita ganhos de performance porque o programador pode controlar a implementação de algoritmos específicos de forma mais eficiente. Assim sendo, os projetos baseados em exonúcleo permitem aos usuários

obter controle da funcionalidade do núcleo. A conseqüência imediata é a falta de segurança: como restringir o que os programas do usuário podem fazer. Em função disto, o projeto que pretende ser pequeno e simples, começa a tornar-se mais complexo com a adição de algoritmos de proteção no núcleo.

A principal diferença entre este enfoque e aquele proposto pelos núcleos monolíticos, é que o limite da portabilidade não mais coincide com a interface entre as aplicações e o núcleo. A vantagem disto é que aplicações especializadas, tais como gerenciadores de banco de dados, módulos de *run-time*[105] de linguagens e *garbage collectors*[106], podem substituir a interface padrão e trabalhar diretamente com o hardware para obter maiores níveis de performance (KAASHOEK,1995).

A questão chave, no entanto, é como expor os recursos de hardware de modo seguro e eficiente visando atingir alto grau de flexibilidade. Em Aegis, a segurança é controlada utilizando-se um conceito semelhante ao de *capabilities*, o qual determina os direitos de seu proprietário. Cada vez que um recurso é usado, o sistema verifica os direitos de acesso codificados com a *capability* associada. Recursos alocados anteriormente, são recuperados através de um protocolo de revogação em que as aplicações são notificadas de que ocorreu uma falha no recurso, para que tomem providências no sentido de liberá-lo. Para evitar que uma aplicação não libere os recursos, deliberadamente ou por erro, o sistema emprega também um protocolo de cancelamento (*abort*) em que os recursos são recuperados à força (MULLENDER,1990)

Um dos problemas de disponibilizar-se uma interface de tão baixo nível é que a freqüência das chamadas de sistema pode aumentar demasiadamente. Isto porque a implementação de níveis mais altos de abstrações pode requerer uma série de chamadas de baixo nível à interface com o núcleo, com conseqüente comprometimento de performance. Este comprometimento não é tão visível em implementações de núcleos monolíticos ou micro kernels (DEARLE et al. 1994).

Bershad (1994) discute como este problema foi solucionado no sistema **SPIN**, permitindo que o código de operações de alto nível seja incluído no núcleo. Dessa forma, operações críticas, tais como escalonamento de threads ou tratamento de interrupções, podem ser executadas em modo supervisor, evitando o overhead de chamadas de sistema.

[105] Módulos (programas, bibliotecas) complementares necessários durante a execução de um programa.
[106] São módulos responsáveis pela desfragmentação e reorganização do espaço da memória dinâmica.

Outra forma é expor as estruturas de dados do núcleo e, assim, diminuir a necessidade de algumas chamadas. No entanto, esta solução aumenta o tamanho do núcleo.

Segundo Engler (1995), máquinas virtuais são uma estrutura de sistema operacional em que um módulo privilegiado (*Virtual Machine Monitor- VMM*) isola software menos privilegiado em cópias emuladas do hardware. Máquinas virtuais (seção 2.4.9) e exonúcleo diferem sob os seguintes aspectos:

- Máquinas virtuais emulam o hardware, enquanto os exonúcleos exportam funções primitivas de baixo nível;
- O processo de emulação esconde informação, o que pode contribuir para o uso ineficiente dos recursos de hardware;
- Um exonúcleo tenta expor toda a informação sobre o sistema e, explicitamente, comunica-se com um *library operating system*, ao invés de apresentar uma visão virtual;
- Máquinas virtuais confinam sistemas operacionais especializados e processos associados em máquinas virtuais isoladas.

O núcleo **Nemesis** (LESLIE et al.,1997; REED, DONNELLY e FAIRBARINS,1997; e STEVEN,1999) possui várias similaridades com a proposta exonúcleo, embora sua abrangência seja bem mais ampla. **Nemesis** foi concebido com o objetivo de melhorar a qualidade de serviço de aplicações multimídia.

Os sistemas tradicionais tornam a contabilização de recursos uma tarefa complexa, uma vez que o código, executando em favor de uma aplicação, ou seja, o desmembramento de uma chamada de biblioteca ou uma chamada de sistema, pode estar espalhado por vários servidores além do próprio núcleo. E esse é o problema que o **Nemesis** pretende resolver. Assim como a proposta exonúcleo, a solução em Nemesis é mover a funcionalidade do sistema operacional para as aplicações. Desta forma, fica mais fácil implementar um controle mais rigoroso dos recursos consumidos pelas aplicações individualmente.

O principal paralelo entre **Nemesis** e um sistema exonúcleo são as interfaces de baixo nível exigidas pelas aplicações multimídia, tais como eventos, rede, disco e processador, e a confiança em library operating systems. Enquanto Nemesis é projetado para simplificar o processo de contabilização, um exonúcleo pretende melhorar a inovação. Em **Nemesis** isto é conseguido liberando-se, para as aplicações, todo o controle que não é necessário para a função de proteção das mesmas. Dessa forma, uma interface exonúcleo garante mais poder para as aplicações e também promove um melhor compartilhamento de recursos (ENGLER,1995).

10.2.7.2 Estruturas em *cache kernel*

Segundo Cheriton et al. (1994) os projetos de *cache kernel* adotam uma filosofia diferente. Ao invés de exportar uma interface com o hardware, como fazem os projetos de exonúcleo, o enfoque *cache kernel* suporta um pequeno conjunto de abstrações primitivas para as quais todas as decisões sobre política são implementadas por bibliotecas em nível de usuário. Um *cache kernel* somente armazena threads, espaços de memória, comunicação entre processos e núcleos. O *cache kernel* executa em modo supervisor. Para cada objeto armazenado, é associada uma aplicação do núcleo. Esta aplicação efetivamente executa em modo usuário e implementa funções de gerência de espaço de endereçamento e escalonamento de suas próprias threads.

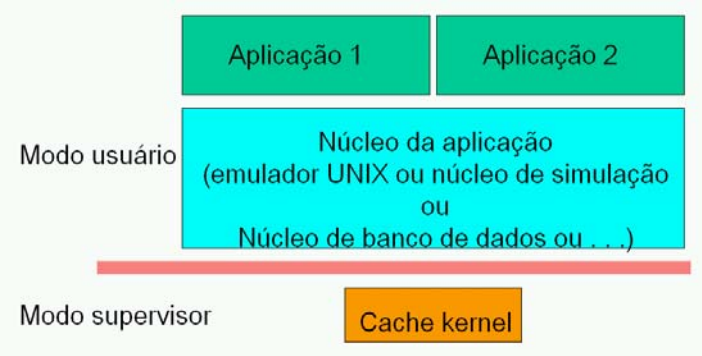


Figura 84 – Modelo de cache kernel.
FONTE: Adaptado de Cheriton e Duda (1994).

Por exemplo, uma aplicação, para controlar a política de escalonamento de suas threads, carrega os respectivos descritores nas estruturas de dados do núcleo. Quando as respectivas políticas de escalonamento determinarem que as threads estão elegíveis para executar, o cache kernel vai escaloná-las e disparar um conjunto de objetos *threads* de modo *round-robin*, efetivamente compartilhando os recursos entre elas. Quando uma *cache* é bloqueada, seu descritor é descarregado do núcleo e retornado para a aplicação que o detém (CHERITON et al. ,1994).

De acordo com Cheriton et al. (1994), este enfoque apresenta três benefícios:

- A interface de código de baixo nível permite às aplicações controlar o gerenciamento de recursos de acordo com qualquer política desejada;
- O enfoque de armazenamento temporário (*caching*) previne as aplicações de exaurir a fonte de descritores de objetos, como pode ocorrer em sistemas convencionais;
- O modelo de armazenamento (*cache*) reduz a complexidade do núcleo, se comparado com os modelos de micronúcleo.

10.2.8 Estruturas Baseadas em Nanonúcleos

Segundo Dearle (2000), as abstrações fornecidas pelos sistemas operacionais convencionais suportam processos que acessam dados que se encontram em memória real ou virtual. Dados armazenados não podem ser acessados da mesma forma. Isto leva à necessidade de uma interface (API) diferente para cada classe de dados. Por exemplo: o **Unix** possui chamadas de sistema para alocação de memória virtual e outras para acesso a arquivos.

Sistemas que suportam persistência ortogonal[107] tratam todos os dados de forma idêntica (como objetos persistentes) e usam uma interface (API) única para manipular tais objetos. Isto implica que os objetos devem ser endereçados de modo uniforme e o passo de transferência do disco para a memória (e vice-versa) deve ocorrer de forma transparente à aplicação (DEARLE e HULSE, 2000).

Quando a persistência é incluída como abstração básica de um sistema operacional, muitos dos problemas de inadequação às necessidades dos programas são eliminados e a tarefa de desenvolver aplicações é simplificada. Sistemas desta categoria devem possuir os seguintes requisitos básicos:

- Suporte a objetos persistentes como abstração básica;
- Os objetos persistentes devem ser tanto estáveis como elásticos[108] (*resilient*);
- O estado dos processos deve estar contido em objetos persistentes;
- Deve haver algum grau de proteção para restringir o acesso aos objetos persistentes.

A necessidade de uma nova abordagem para aplicações persistentes decorre do fato de que aplicações com este perfil, quando desenvolvidas sobre sistemas operacionais tradicionais, como **Unix**, **Mach** ou **Windows NT**, não apresentam os requisitos de performance necessários. Isto decorre do fato de que há uma lacuna (*gap*) semântica entre os requisitos das aplicações e os serviços providos pelos sistemas.

Estas características, quando implementadas sob núcleos monolíticos em particular, apresentam pelo menos mais três diferentes classes de problemas, quais sejam:

- As abstrações de alto nível são inadequadas para a construção de sistemas persistentes;

---

[107] Segundo Dixon et al. (1989), persistência ortogonal implica que a propriedade de persistência é independente tanto da classe de um objeto como da maneira como ele á usado.

[108] *Resilience*: elasticidade, capacidade de recuperação rápida.

- A evolução de núcleos monolíticos é extremamente difícil, devido à fraca estruturação interior e à natureza monolítica; e
- Como todos os módulos do núcleo residem no mesmo espaço de endereçamento, um erro em um módulo pode comprometer o sistema operacional.

Segundo Dearle (2000), o primeiro aspecto é evidenciado pelo grande tamanho dos sistemas persistentes que foram implementados usando como base sistemas monolíticos, como, por exemplo, **Napier88** (MORRISON,1989) e **Pjama** (ATKINSON,1996). Isto porque muito da complexidade destes sistemas deve-se, também, à forma como a máquina virtual que os suporta foi concebida e à plataforma inapropriada. Dearle e Hulse (2000), identificaram, entre outros, os seguintes sistemas que suportam alguma forma de persistência: **Multics** (BENSOUSSAN,CLIGEN e DALEY,1972,CORBATÓ E VYSSOTSKY,1965), **Monads** (KEEDY,1989, ROSENBERG,1990), **Clouds** (DASGUPTA,1990), **Choices** (CAMPBELL JOHNSON e RUSSO,1987 e, HARDY,1985) e **Grasshopper** (DEARLE et al. 1994; ROSENBERG et al.,1996).

O segundo aspecto tem como decorrência o fato de tornar o processo de desenvolvimento e manutenção mais difíceis, devido à existência de complexas interações entre módulos. Uma conseqüência direta é que o processo de evolução dos núcleos monolíticos também é complexo. Um enfoque para a solução do problema é o desenvolvimento de máquinas virtuais, as quais servem como uma ponte entre as aplicações e os sistemas operacionais. A necessidade deste nível extra é que compromete a performance das aplicações (DEARLE ,2000).

Segundo Hulse e Dearle (1998), em função das questões identificadas, relativas à construção de aplicações persistentes sobre os núcleos de sistemas operacionais existentes, tornou-se necessário o desenvolvimento de um novo enfoque, o qual foi denominado nanonúcleo[109]. Esta camada implementa toda a porção de código dependente de hardware. Isto permite a construção de um sistema operacional independente de hardware. Este nível de código contém primitivas para alocar memória física, mapeamento de tradução de endereços, mascarar interrupções, refletir exceções e interrupções e troca de contexto.

Este modelo tem como conseqüência uma dificuldade adicional na determinação do limite entre modo supervisor e modo usuário. Segundo Hulse e Dearle (1998), uma proposta é a de construir um nível de proteção sobre o nanonúcleo e limitar o modo supervisor ao

[109] Também referida na literatura como *Hardware Abstraction Level -HAL.*

próprio nanonúcleo. Esta implementação resultaria em uma arquitetura exonúcleo onde o micronúcleo se tornaria uma espécie de *library operating system*. Outro enfoque poderia ser o de considerar tanto o código do micronúcleo quanto o código do nanonúcleo, como sendo protegidos pelo modo supervisor. Este enfoque produziria uma estrutura de micronúcleo, o que poderia não ser tão flexível quanto a alternativa anterior.

Bomberger et al.(1992) descrevem um projeto de nanonúcleo denominado **KeyKOS**. O sistema está baseado em três conceitos arquitetônicos que não são comuns na comunidade **Unix**, quais sejam:

- Núcleo *stateless*.
- Armazenamentopersistente.
- *Capabilities*.

Uma das primeiras decisões de projeto foi eliminar do núcleo a necessidade de manutenção de estruturas de dados que representam os estados críticos do sistema, como, por exemplo, filas de mensagens. Assim, todo o estado do nanonúcleo é derivado de informações que persistem através de reinicializações (*restarts*) do sistema e falhas de alimentação de energia (*power failures*). Por questões de eficiência, o nanonúcleo reformata informações de estado em uma área privativa. Todo o armazenamento privativo é meramente um cache do estado persistente, o qual pode ser descartado a qualquer momento. Quando a informação descartada for necessária novamente, ela é reconstruída a partir de informações em nodos e páginas. Assim, o nanonúcleo não executa alocação dinâmica da área de armazenamento de dados do núcleo. Isto têm como conseqüências:

- O núcleo é rápido, visto que não há código complexo de alocação de memória;
- O núcleo nunca consome nem endereça área não alocada;
- Não há armazenamento no nanonúcleo, como filas de mensagens, que deva fazer parte de um *checkpoint*, e;
- A ausência de alocação dinâmica significa que não há interação entre estratégias de alocação dinâmica. Esta é uma fonte predominante de problemas de consistência e deadlock em muitos sistemas operacionais;
- O sistema fora do nanonúcleo é completamente descrito pelo conteúdo de nodos e páginas que são persistentes. Este estado inclui arquivos, programas, variáveis dos programas, contadores de instruções, status de operações de entrada/saída e qualquer outra informação necessária para reinicializar o sistema. Além disso, a possibilidade de recuperar todo o ambiente de *run-time* do núcleo a partir de um estado de *checkpoint* significa que uma interrupção de energia não corrompe o estado dos programas em execução.

O sistema **KeyKOS** possui um modelo de armazenamento persistente. Dessa forma, do ponto de vista das aplicações, todos os dados estão em memória virtual persistente. Somente o nanonúcleo faz distinção entre memória real e páginas do disco. O sistema de paginação é associado ao mecanismo de checkpoint. Os checkpoints periódicos de todo o sistema garantem a persistência de todos os dados do sistema. Em **KeyKOS**, uma aplicação também é persistente, ou seja, ela continua sua execução até que seja explicitamente parada. O período de desligamento/re-ligamento (*shutdown*) é visível somente como um inesperado salto (*jump*) para o tratador do relógio de tempo real (no máximo). Como conseqüência, as questões relativas ao seqüenciamento dos programas durante a inicialização não se aplicam neste contexto (BOMBERGER et al.,1992)

A última característica é que **KeyKOS** é um sistema baseado em capabilities, as quais são implementadas como se fossem chaves (*keys*). Cada objeto no sistema é exclusivamente referenciado por uma ou mais chaves associadas. As entidades em execução ativam serviços de outras entidades enviando mensagens através de uma chave. A mensagem inclui uma chave de retorno construída pelo núcleo, a qual pode ser utilizada pela entidade recipiente para enviar uma resposta. Este mecanismo é utilizado para implementar uma forma de RPC no sistema.

Segundo Bomberger et al.(1992), as maiores vantagens de um projeto de nanonúcleo são:

- O núcleo inclui todo o código do sistema em modo supervisor[110];
- O núcleo em execução ocupa 100Kb de memória principal;
- O núcleo é a única parte do sistema que interpreta as chaves. Nenhum outro programa tem acesso direto aos bits contidos nas chaves, o que evita a questão de *key forgery*;
- O núcleo inclui código que define os objetos primitivos do sistema. Estes objetos são suficientes para construir as abstrações de alto nível suportadas pelos sistemas operacionais convencionais. Estas abstrações são: suporte a multiprogramação, escalonador primitivo e filtros (*hooks*) para escalonadores mais sofisticados que executam como aplicações;
- Suporte à persistência de um nível (*single-level store*);
- Cada processo possui seu próprio espaço de endereços virtuais;
- Suporte a um mecanismo global de *checkpoint* (*system-wide*);
- Suporte ao sistema de chaves a partir das quais mensagens são enviadas de uma aplicação para outra de forma segura;

[110] O núcleo do sistema é implementado em aproximadamente 20.000 linhas de código fonte em C e 2000 de código assembly, sendo 1000 linhas usadas para implementar a operação de troca de contexto.

- Controle de acesso primitivo e limitado a dispositivos de entrada/saída;
- A interpretação das chaves pelo núcleo esconde a localização do objeto (se está no disco ou em memória principal).

Para destacar as vantagens da utilização do conceito de nanonúcleo, Bomberger et al.(1992) descrevem uma implementação de um sistema operacional **Unix** utilizando as primitivas abstratas exportadas pelo nanonúcleo **KeyKOS** em uma plataforma IBM System/370. Destacam ainda Bomberger et al.(1992) que o nanonúcleo **KeyKOS** foi utilizado por pelo menos 9 anos[111] de forma satisfatória.

10.2.9 Estruturas Baseadas em Máquinas Virtuais

Uma máquina virtual é uma ilusão de uma máquina real. Ela é criada por um sistema operacional de máquina virtual, a qual faz com que uma máquina real pareça ser várias máquinas reais. Do ponto de vista do usuário, as máquinas virtuais podem parecer semelhantes ao hardware real, ou serem muito diferentes dele. Este conceito foi utilizado na série de sistemas **VM/370**, da IBM (DEITEL,1990).

Segundo Folliot (2000), as arquiteturas e sistemas são complexas, respondem precariamente a várias necessidades emergentes e não são facilmente especializáveis para suportar uma determinada aplicação ou arquitetura. Este fato leva à proliferação de soluções ad-hoc e impõe requisitos restritivos aos desenvolvedores. A estrutura de máquinas virtuais surgiu como uma nova forma de abstração sobre os recursos de hardware. Até então, as abordagens procuravam maximizar ou minimizar a visão que os programadores de aplicações tinham do hardware. A abordagem de máquinas virtuais, ao contrário, implementa em software uma máquina que possui os mesmos recursos e funcionalidades do hardware real, inclusive as instruções privilegiadas.

O conceito de máquina virtual está associado ao conceito de microprogramação, o qual, segundo Deitel (1990), é associado ao trabalho do Prof. Maurice Wilkes[112], na década de 50. Ainda segundo Deitel, os conceitos apresentados no trabalho de Wilkes constituíram a base das atuais técnicas de microprogramação[113]. Contudo, somente aproximadamente 16

[111] Em relação à data da publicação.

[112] Wilkes, M.V. The Best Way to Design na Automatic Calculating Machine. Report of the Mancester University Computer Inaugural Conference. Electrical Engineering Department of Manchester University, Manchester, England, July 1951, ppp.16-18. Reprinted in Earl E.Swartzlander,Jr. (ed.), Computer Design Development – Principal Papers, Rochelle Park, N.J.: Hayden Book Co.,1976, pp.139-145.

[113] Na década de 60 o conceito de microprogramação era utilizado principalmente na implementação do conjunto de instruções das diversas arquiteturas desenvolvidas na época (DEITEL ,1990).

anos depois é que os conceitos passaram a ser utilizados em escala comercial (em computadores da linha /370). O conceito de microprogramação introduz uma camada de software entre o processador propriamente dito e a visão externa que se tem dele a partir do conjunto de instruções utilizado para programá-lo Assim, a linguagem de montagem (*assembly*) pode não necessariamente representar os padrões de bits que o processador utiliza para executar suas operações primitivas, mas sim representar internamente procedimentos com parâmetros, os quais efetivamente implementam a funcionalidade de cada instrução[114] (DEITEL,1990).

Quando uma máquina virtual implementa uma cópia da máquina real e executa sobre o mesmo hardware da máquina real (Figura 86), o comprometimento de performance não é tão significativo quanto o que ocorre quando a máquina virtual implementa um conjunto de instruções completamente diferente daquele que é suportado pela máquina real. O primeiro caso pode ser exemplificado pelas implementações recentes de sistemas como **VMWare** (VMWARE 2001, SUGERMAN, VENKITACHALAM e LIM,2001), **VirtualPC** (CONNECTIX,2001) e **Bochs** (BOCHS,2001).

A Figura 85 apresenta cópias de telas demonstrando: (a) processo de boot de uma máquina virtual **MS-DOS**; (b) formatação do disco rígido desta máquina virtual e (c) verificação (com scandisk) do disco formatado. A execução desta máquina virtual é concomitante com a execução do sistema operacional host (**Windows 2000**).

[114] Por exemplo, os processadores da família Intel Pentium 4, implementam as instruções da família CISC x86 através de micro-operações. Esta característica garante que uma máquina com um processador desta categoria vai poder executar um sistema operacional **MS-DOS** ou **Windows 3.1**. Este sistema não tem como saber que as instruções são microprogramadas e executam normalmente. O programador também não tem como distinguir as instruções microprogramadas daquelas efetivamente implementadas em hardware.

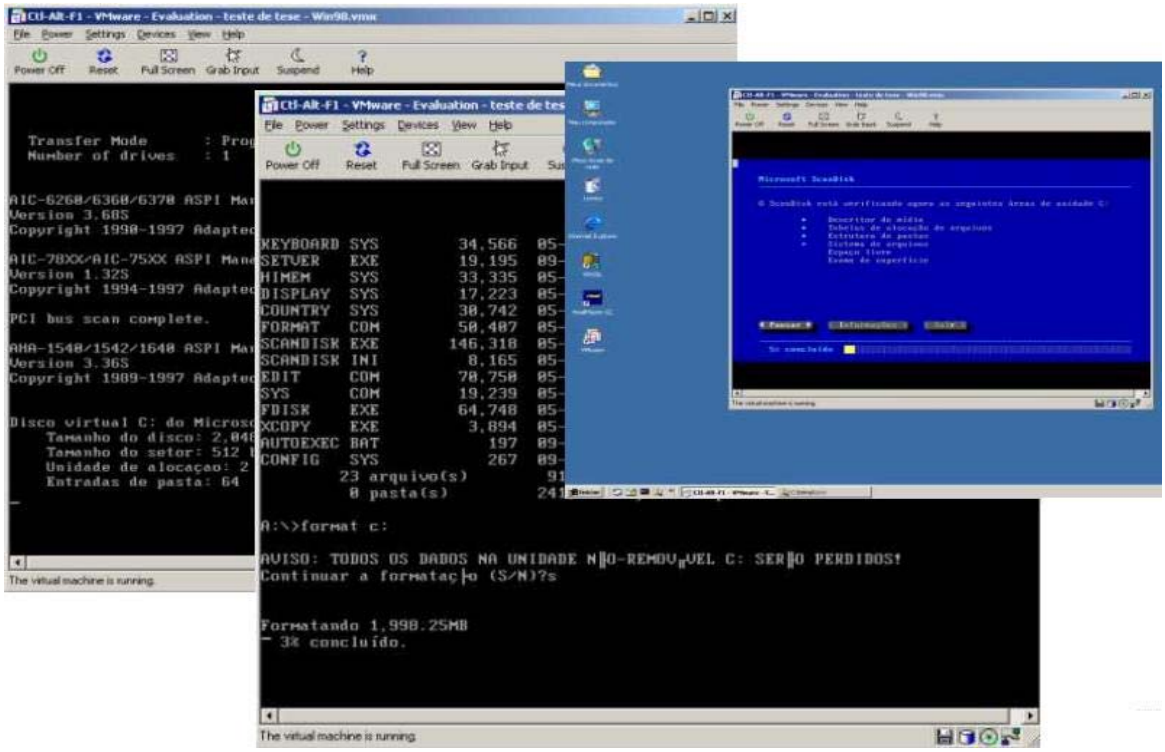


Figura 85 – Exemplo de funcionamento de uma máquina virtual Windows 98 executando sobre um host Windows 2000

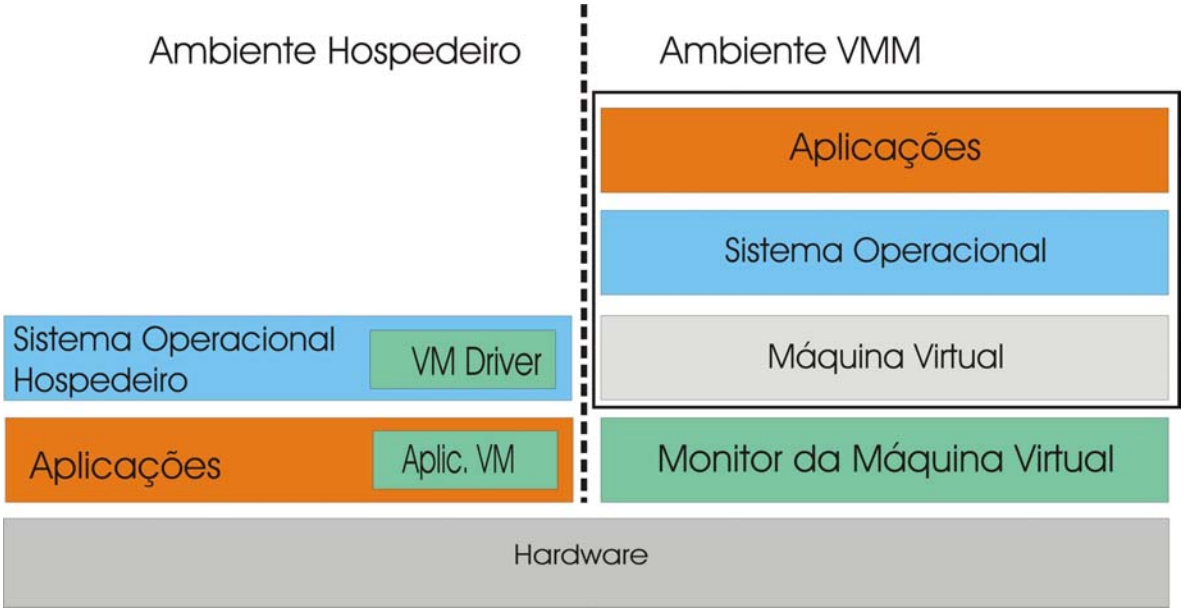


Figura 86 – Uma máquina virtual implementada sobre uma máquina real com mesmo conjunto de instruções.
FONTE:Adaptado de Sugerman,Venkitachalam e Lim (2001).

Quando a máquina virtual implementa um conjunto de instruções diferente daquele suportado pelo processador real, tem-se um completo ambiente virtual (Figura 87). São exemplos deste caso a especificação do processador **Java** (VENNERS,1999), o processador

Jalapeño (ALPERN et al.,2000) e a especificação da máquina **VirtualPC** (CONNECTIX,2001) para plataformas Macintosh.

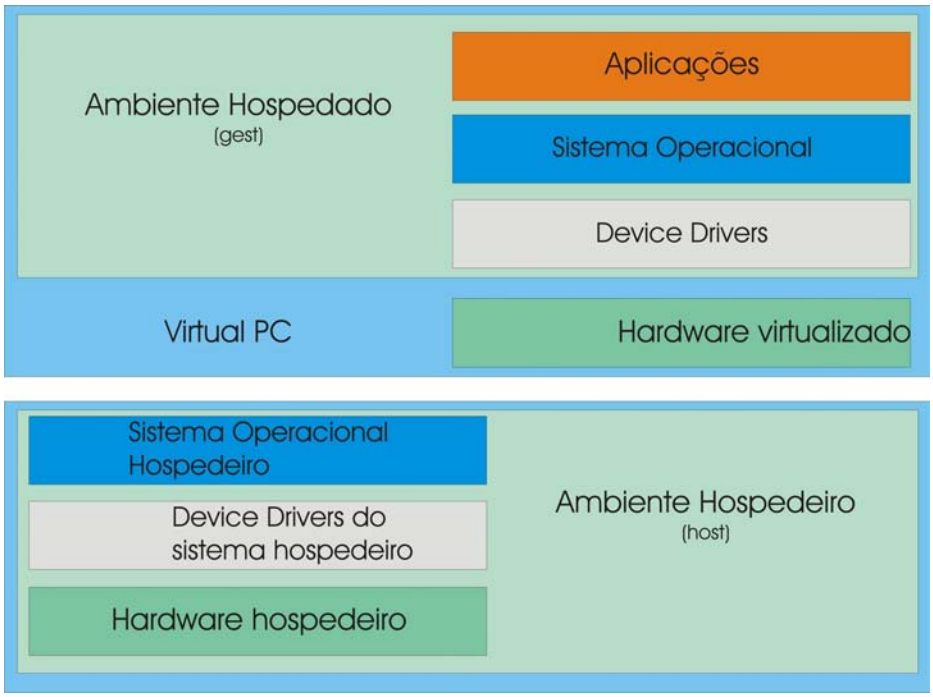


Figura 87 - Estrutura da Máquina Virtual VirtualPC.

A especificação da máquina Java, por exemplo, estabelece, entre outros, o formato das instruções e os modos de endereçamento. Como o processador alvo geralmente é diferente daquele especificado, utiliza-se um interpretador para implementar a máquina virtual. O código C do interpretador que implementa estas instruções e a funcionalidade correspondente em um processador real[115], pode ser imaginado como uma camada de microprogramação do ponto de vista do usuário que programa utilizando a linguagem Java.

Um sistema operacional que seja construído para executar sobre o processador Java vai utilizar o conjunto de instruções conforme a especificação deste processador (por exemplo: JavaOS descrito em Saulpaugh e Mirho (1999). Se um determinado byte code vai ser implementado como um procedimento (ou um conjunto) ou não, não interfere na construção do sistema operacional, ou das aplicações desenvolvidas para executar sobre

[115] A técnica de fazer uma máquina comportar-se como se fosse outra é referida na literatura (DEITEL,1990) como: emulação. Esta técnica foi amplamente utilizada pelos fabricantes de computadores quando da introdução de novas linhas de processadores, tendo em vista manter a compatibilidade com código desenvolvido para linhas anteriores e, como visto anteriormente, continua a ser empregada atualmente.

este processador[116]. No entanto, quando se utiliza um processador Java real, esta analogia com a camada de microprogramação desaparece, uma vez que o código binário dos byte code estará sendo executado efetivamente pelo processador real.

Assim como a máquina virtual Java, outras propostas de máquinas virtuais tem sido descritas na literatura, como: **Máquina Pascal Concorrente** (BRINCH HANSEN,1970, BRINCH HANSEN,1975), **VM/370** (CREASY,1981), **Dis** (DORWARD,1997).

A estrutura de máquinas virtuais interpôs uma nova camada de software entre o sistema operacional e o hardware. Apesar de ser uma inovação estrutural, uma vez que esta proposta permitia que vários sistemas fossem executados sobre um mesmo hardware (CREASY,1981), apresentava como característica uma tendência em degradar a performance da máquina real. Segundo Deitel (1990), em função deste fato, no passado, várias partes do código do sistema operacional mais freqüentemente utilizadas eram implementadas em microcódigo como uma forma de obter-se melhor performance do sistema. Alguns exemplos de funções implementadas desta forma são:

- Tratamento de interrupções;
- Manutenção das diversas estruturas de dados do núcleo;
- Primitivas de sincronização que controlam o acesso a estruturas compartilhadas;
- Operações sobre bits de uma palavra, objetivando melhorar a performance de algoritmos que operam sobre *strings*;
- Troca de contexto;
- Seqüências de chamadas e retorno de procedimentos.

A abordagem de máquinas virtuais apresenta algumas vantagens, as quais são relacionadas a seguir:

- Suporte à execução de múltiplos sistemas operacionais simultaneamente sobre o mesmo hardware;
- Independência de plataforma;
- Análise e depuração de sistemas operacionais;
- Ensino e pesquisa;
- Compatibilidade com versões anteriores.

[116] Em função desta afirmação, também podemos abstrair o fato de os conjuntos de instruções utilizados pelos processadores conhecidos atualmente (Pentium Intel, PIC, Strong ARM,etc) utilizarem ou não microprogramação em suas reais implementações.

A estratégia de máquinas virtuais também pode ser utilizada para análise e depuração de sistemas operacionais e/ou aplicativos. Rosenblum (1997) descreve um ambiente de simulação de uma máquina de arquitetura MIPS, o qual permite a execução de um sistema operacional real. A partir disso, é possível analisar-se os impactos (ROSENBLUM,1995) que inovações tecnológicas em processadores e periféricos podem causar num ambiente real.

A estratégia de simulação de um hardware real também é descrita em Mattos e Tavares (1999). Porém neste caso, o enfoque é didático e tem por objetivo demonstrar os efeitos da execução de programas reais sobre o sistema operacional e vice-versa.

Novoselsky (2000) descreve uma máquina virtual que possui como instruções o modelo XSL (*extensible style sheet language*). O enfoque proposto tem por objetivo separar operações de tempo de compilação daquelas de tempo de execução. A idéia é que um compilador possa fazer uma série de verificações em tempo de compilação, otimizando o código gerado e, assim, os recursos (e tempo) do usuário no momento da exibição dos documentos.

Segundo Hazzah (1997), a razão pela qual a Microsoft adotou o modelo de máquinas virtuais no **Windows** foi para suportar as antigas aplicações desenvolvidas para ambiente **DOS**. Isto porque uma aplicação **DOS** assume que ela é a única aplicação que está sendo sendo executada em uma máquina. Além disso, aplicações **DOS** acessam o hardware diretamente, usam toda a memória disponível e o tempo de processador também. Como no **Windows** há outras aplicações executando em paralelo, a solução encontrada foi criar uma máquina virtual para cada aplicação DOS, de tal forma que seja possível:

- Receber o controle do processador quando é realizado algum acesso ao hardware e redirecionar este acesso para os serviços do **Windows**;
- Mapear a memória real em memória virtual para permitir que programas DOS continuem acessando toda a memória disponível;
- Suspender uma máquina virtual para entregar o processador a outras aplicações num ambiente multitarefa.

Ainda segundo Hazzah (1997), a virtualização do hardware (espaço de endereçamento, espaço de entrada/saída, operações de interrupção e registradores de hardware) só é possível a partir dos processadores 386, os quais incluem suporte para tradução de endereços, paginação por demanda, trapping de entrada/saída, instruções e interrupções. Assim, um módulo do **Windows** denominado VMM – *Virtual Machine Monitor*, utilizando

estas facilidades de hardware cria várias máquinas virtuais independentes, cada uma das quais com seu próprio ambiente virtual de execução.

Todas as aplicações Win32 e Win16 executam em uma única máquina virtual denominada *System VM*, enquanto cada aplicação **DOS** executa em sua própria máquina virtual. A conseqüência imediata desta solução é que o processo não só é complicado, mas também envolve um grande overhead, a ponto de uma aplicação executando sob o **Windows** poder apresentar uma certa latência[117], no atendimento das interrupções, de até 20 vezes[118] em relação ao tempo de atendimento da mesma aplicação executando sob o **DOS** (HAZZAH,1997, pág.107).

Kamibayashi et al. (1982) apresentam um enfoque diferente para projeto de maquinas virtuais. Ao contrário do movimento constatado nos projetos anteriores, a proposta dos autores é levar para o firmware funções do núcleo do sistema operacional. Segundo os autores, através da experiência na implementação de uma máquina abstrata que implementa o núcleo de um sistema em firmware, as seguintes vantagens são obtidas:

- Melhoramento na relação entre custo e compartilhamento de carga de trabalho entre hardware e software no projeto de um sistema operacional;
- Consolidação de um ambiente de programação para projetistas de sistemas operacionais;
- Melhoramento da performance do computador;
- Estabelecimento de um ambiente de execução com maior confiabilidade.

O núcleo do projeto **HEART** executa as seguintes funções:

- Multiplexação de processos e escalonamento;
- Gerenciamento de domínios;
- Gerenciamento de segmentos e mapeamento de espaço de endereçamento;
- Cooperação entre processos.

O esquema de proteção de domínios é implementado em **HEART** através do mecanismo de capabilities, anteriormente descrito. Um aspecto que chama a atenção é que, segundo os autores, cada módulo foi implementado em linguagem Pascal. A versão em firmware é implementada de acordo com as seguintes políticas:

---

[117] Segundo Hazzah (pág 105,1997) , latência de interrupção corresponde ao atraso entre a sinalização da interrupção pelo hardware e o início da execução da rotina de atendimento.

[118] Segundo Hazzah (pág 105,1997), para minimizar o efeito da latência de interrupção no Windows, deve-se implementar as rotinas de atendimento na forma de VxD (*device drivers* virtuais).

- Os espaço de código e dados são separados da área dos processos para proteger o sistema de acesso não autorizado;
- O microcódigo do sistema é projetado para utilizar registradores internos da máquina que não são disponíveis ao nível do conjunto de instruções, para os quais as linguagens de alto nível geram código. O hardware utilizado (PFU-1500) possui 128 registradores internos que podem ser utilizados ao nível de microprogramação[119].

Kamibayashi et al. (1982) afirmam que o uso da técnica de microprogramação possibilita ganhos de performance em função da velocidade de execução das primitivas em hardware, além de permitir a implementação do conceito de máquina de núcleo de sistema operacional. Isto significa dizer que o processador sobre o qual serão desenvolvidas as aplicações terá, em seu conjunto de instruções, instruções especiais que implementam funções primitivas de núcleo de sistema operacional, interferindo de maneira importante na forma de concepção do sistema operacional para este hardware.

10.2.10 Estruturas Baseadas no Conceito de Objetos

Estruturas orientadas a objetos têm base na construção dos vários componentes do sistema operacional como uma coleção de objetos. A estrutura baseada em objetos tem a construção do sistema operacional com base no conceito de objetos, onde os serviços do sistema são implementados como uma coleção de objetos definidos como segmentos protegidos, através dos quais os estados internos são acessados e alterados (ZANCANELLA e NAVAUX,1993).

Nesta abordagem, geralmente o núcleo do sistema tem a função de gerenciamento dos objetos, gerenciamento de interações entre objetos e a responsabilidade de proteger os objetos contra acessos indevidos. Uma característica desta abordagem é que ela requer uma disciplina para organizar os serviços do sistema operacional como uma coleção de objetos. Outra possível característica é a existência de uma thread de controle de modo que objetos coletivamente possam ser tratados como uma unidade única (ZANCANELLA e NAVAUX,1993).

Segundo Nicol et al. (1989) apud Zancanella e Navaux (1993) o sistema **Cosmos** caracteriza um exemplo de implementação baseada neste modelo, onde todos os recursos são representados como abstração do modelo de objetos, incluindo o núcleo do sistema. Uma característica deste modelo é a imutabilidade dos objetos, o que significa que ditos

[119] A implementação do sistema **HEARTS** utilizou 56 destes registradores sendo: (i) 13 para acesso a informações do banco de dados do núcleo; (ii) 14 como registradores temporários; (iii) 4 como registradores para passagem de parâmetros; (iv) 12 para implementar a estrutura de chamadas ao núcleo e, (v) 13 outros registradores gerais. (KAMIBAYASHI et al.,1982).

objetos, assim que criados, não podem ser alterados. Uma conseqüência imediata deste enfoque é a produção de um grafo de versões com a história do objeto.

O sistema **Aurora** (ZANCANELLA e NAVAUX,1993) apresenta uma outra proposta de sistema operacional orientado a objetos baseado em arquiteturas de multiprocessadores. O conceito de migração de objetos é utilizado para implementar os serviços do sistema operacional e distribuir a carga do sistema entre os processadores. Além disso, o sistema suporta o conceito de herança dinâmica e trata todos os objetos de maneira uniforme e independente de localização.

O sistema **Aurora** utiliza o modelo estrutural de **Apertos**, o qual implementa a idéia de que o nível de objetos e o nível de abstração das linguagens possam ser separadamente descritos e implementados dentro da mesma estrutura. Assim, Aurora utiliza o conceito de separação entre objetos e meta-objetos, onde o objeto representa um repositório de dados, enquanto o meta-objeto define não somente a semântica de seu comportamento, mas a abstração da classe (ZANCANELLA E NAVAUX,1993).

A proposta do sistema **PURE** (BEUCHE,1997) é o desenvolvimento de um sistema operacional configurável de modo a possibilitar ao projetista da aplicação escolher a funcionalidade necessária do sistema. Para tanto, a filosofia que norteia o projeto é de que um sistema operacional é uma família de programas e a orientação a objetos é a disciplina de implementação fundamental. Com isto, evita-se o desenvolvimento de um sistema monolítico e a orientação a objetos permite uma implementação de uma estrutura de sistema altamente modular. Este aspecto caracteriza o projeto **PURE** como uma ferramenta para construção de sistemas operacionais. Todo o sistema é representado por um conjunto de pequenas bibliotecas de módulos úteis a determinada situação. Isto permite o desenvolvimento de um sistema operacional sob medida para uma aplicação.

Segundo Beuche et al. (1999), a vantagem deste conceito de família é permitir o desenvolvimento do software aplicativo, de forma que futuras ampliações sejam facilitadas. Contudo, à medida que a complexidade do projeto aumenta, começa a se tornar difícil o processo de seleção dos componentes apropriados. Beuche (1997) relata o desenvolvimento do sistema **PEACE**, o qual foi totalmente escrito em C++ demonstrando a possibilidade de se construírem sistemas operacionais eficientes, utilizando-se a orientação a objetos como filosofia de implementação. Este sistema possui 170 classes, as quais podem ser

combinadas para produzir 20 membros diferentes de uma família, permitindo a geração de código para quatro tipos diferentes de processadores. Contudo, esta flexibilidade produz mais de 80 conjuntos diferentes de código fonte, o que implica num potencial problema de manutenção (BEUCHE, 1997).

Para desenvolver um sistema operacional adequado a determinada situação, tanto os requisitos funcionais, correspondentes a capacidades demandadas – ex: ter ou não preempção, como os requisitos não-funcionais (questões relacionadas à implementação), devem ser considerados. Em determinadas situações, decisões sobre configuração global podem causar efeitos secundários tanto globais como parciais no sistema gerado. A maior parte destas decisões está relacionada aos requisitos não funcionais, como seleção de estratégia de tratamento de erros ou se são necessárias otimizações para diminuir o tamanho ou a velocidade do código gerado (BEUCHE et al.,1999).

O código do programa, que pode ser influenciado por este tipo de decisão de configuração, pode envolver o sistema todo. Portanto, o maior problema relacionado à implementação de aspectos específicos de um programa é que esta estratégia não é adequada ao conceito de decomposição funcional. Isto porque, na geração do código final, haverá código relacionado a aspectos de configuração específicos ligado a código que não necessita nenhuma espécie de especialização (BEUCHE et al.,1999).

A técnica de programação orientada a aspectos (AOP) se propõe a solucionar estes problemas através do principio da separação de interesses (concerns), onde são separados o código de um componente do código de um aspecto. Assim, o código de um aspecto pode consistir de vários programas (*aspect programs*) onde cada um implementa um aspecto específico utilizando uma linguagem orientada a problemas (*problem-oriented*). Posteriormente, um *aspect weaver* analisa o código dos componentes e dos aspectos, interpreta-os, localiza pontos de junção e une (*weaves*) as partes para formar uma única entidade (KICZALES et al.,1997).

Segundo Beuche et al. (1999), no caso de projetos altamente configuráveis, como PURE, estas facilidades são extremamente úteis tendo em vista permitirem a redução do processo de configuração a uma atividade de seleção de diferentes aspectos (*program aspects*). Sem o uso de AOP, o processo de configuração pode conduzir à geração de código-fonte poluído com um grande número de diretivas de compilação.

Gutiérrez et al. (1997) descrevem o projeto de pesquisa **Oviedo3** que pretende construir um sistema operacional completamente orientado a objetos baseado em um substrato de uma máquina abstrata também orientada a objetos. Esta máquina provê a base do modelo de objetos e o suporte para o restante do sistema. O coração do sistema **Oviedo3** é o sistema operacional **SO4** (Sistema Operacional para Oviedo3).

O sistema oferece a abstração de um único espaço de objetos virtualmente infinito, onde os objetos existem indefinidamente e onde os objetos localizados em diferentes máquinas cooperam de forma transparente através de mensagens. Entre outras estratégias, o projeto propõe uma arquitetura reflexiva como um mecanismo de colaboração entre a máquina e o sistema operacional. Concluem os autores que esta estratégia facilita a construção de facilidades básicas de um sistema operacional, como persistência ortogonal, distribuição de objetos, concorrência e segurança baseada em *capabilities*.

De acordo com Zancanella (1993), outros exemplos de sistemas que utilizam a técnica de orientação a objetos em seus projetos são: **Hydra** (WULF ,1974), **Chorus** (BANINO,1985), **Amoeba** (MULLENDER et al.,1990).

10.2.11 Estruturas Baseadas em Reflexão Computacional

Separação de interesses é uma estratégia que procura separar os requisitos funcionais dos requisitos operacionais de uma aplicação. A estratégia de reflexão computacional é uma forma de separação de interesses onde os aspectos funcionais e não funcionais são separados através do uso de objetos base e meta-objetos.

Segundo Sztajnberg (1999), os objetos–base implementam em seus métodos a funcionalidade da aplicação, enquanto os meta-objetos implementam os procedimentos de controle que determinam o comportamento dos objetos base a ele associados. Os objetos base podem não estar cientes da existência dos meta-objetos, não sendo obrigatório que cada objeto possua um meta-objeto associado. As chamadas aos métodos do objeto base são desviadas de modo a ativar métodos na meta-classe, permitindo, dessa forma, a supervisão e a modificação do comportamento do código funcional residente no objeto base. Além disso, também é possível adicionar funcionalidades ao objeto base. A execução do objeto base é suspensa até que o meta-objeto termine a sua execução, o que permite ao meta-objeto ter completo controle das atividades do objeto base (Figura 88).

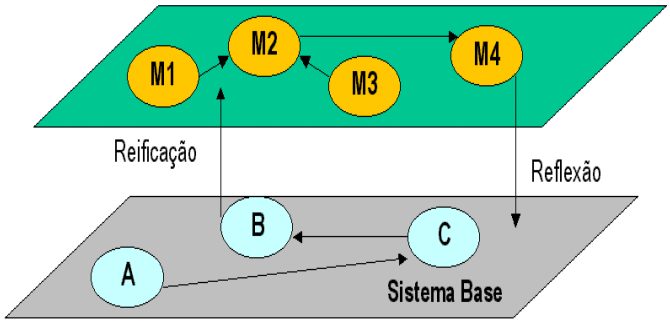


Figura 88 - Objetos A, B e C implementam a solução de um problema específico. Objetos meta-nível M1..M4 controlam a execução dos abjetos A, B e C e implementam parte de seu comportamento.
FONTE: Adaptado de Golm (1997).

Segundo Stankovic (1998) informações reflexivas constituem-se em meta dados sobre o sistema. Estes meta dados podem ser sobre as aplicações, bem como sobre a performance e outras propriedades do sistema. Todos os sistemas operacionais são reflexivos em algum grau. Contudo, somente a implementação da noção de reflexão como um dos princípios centrais de projeto do sistema é que tornará possível a construção de sistemas mais flexíveis.

Segundo Zancanella e Navaux (1993), as estruturas reflexivas caracterizam-se como sistemas capazes de acessar sua própria descrição e modificá-la, de modo a alterar seu próprio comportamento. Estas estruturas apontam para uma nova forma de construir-se aplicações, tanto ao nível de sistema operacional, quanto ao nível de programas de aplicação, apesar dos esforços ainda serem, em sua grande maioria, desenvolvidos em ambiente de pesquisa acadêmica. Pode-se citar como exemplo o sistema **Apertos** (YOKOTE,1992).

Fujinami apud Zancanella e Navaux (1993,pág.31), descreve um modelo denominado **Muse**[120]. Neste modelo, um objeto é uma simples abstração de um recurso computacional do sistema e pertence a um nível de abstração mais elevado, denominado de meta-espaço. O meta-espaço é que provê um ambiente de execução. **Muse** é baseado em programação concorrente orientada a objetos, onde as aplicações, os dispositivos e mesmo a memória são definidos como objetos ou coleções de objetos. Neste modelo, cada objeto consiste de: memória local, métodos que acessam a memória local e processadores virtuais que executam os métodos (ZANCANELLA e NAVAUX,1993).

[120] O nome Muse foi posteriormente alterado para Apertos (YOKOTE,1992 apud ZANCANELLA e NAVAUX,1993).

Segundo Zancanella e Navaux (1993) o conceito de processadores virtuais distingue os objetos **Muse** de outras estruturas baseadas em objetos, pois permite que cada objeto tenha seu próprio ambiente de execução e permite a introdução do conceito de computação reflexiva.

Segundo Sztajnberg (1999), o termo reflexão, dentro de um contexto genérico, possui dois sentidos. O primeiro com o significado de introspecção ou meditação e o segundo no sentido de refletir um evento, como a imagem em um espelho. Estes dois sentidos podem ser aplicados na utilização do conceito de reflexão na área de engenharia de software, tendo em vista permitir que uma aplicação possa executar algum processamento em benefício próprio, para modificar ou ajustar a sua própria estrutura ou comportamento.

Uma aplicação que utilize a técnica de reflexão computacional é dividida em um nível base, onde são implementadas as funções básicas, e um meta-nível, onde os aspectos não funcionais são implementados.

Genericamente, o processo de reflexão envolve os seguintes estágios distintos[121]:

- O primeiro estágio consiste em obter uma descrição abstrata da aplicação, tornando-a suficientemente concreta para permitir operações sobre ela. Esta operação é denominada reification (materialização);
- Um meta-nível utiliza a descrição concreta do nível inferior e executa algum tipo de operação computacional sobre ele, podendo alterá-lo ou não;
- O controle retorna ao nível inferior carregando consigo as eventuais alterações efetuadas pelo meta-nível. Assim, as operações subseqüentes deste nível refletirão as alterações que porventura tenham sido realizadas no meta-nível.

Cazzola (1998) identifica duas abordagens relativas à forma como a reflexão computacional pode ser implementada: estrutural e comportamental.

A reflexão estrutural pode ser definida como a capacidade que uma linguagem possui para permitir uma completa materialização (*reification*), tanto do programa que está sendo executado, quanto de seus tipos abstratos de dados.

As linguagens funcionais (como Lisp) e linguagens lógicas (como Prolog) possuem comandos que permitem manipular a representação de um programa. Tais comandos são baseados na natureza interpretativa das linguagens e isto pode tornar mais fácil a introdução do processo de reflexão estrutural. Por outro lado, a reflexão estrutural em linguagens não

[121] Uma análise aprofundada dos mecanismos e do uso da reflexão computacional foge ao escopo deste trabalho. Portanto, esta discussão limita-se a uma síntese conceitual a respeito.

puras orientadas a objetos, é realizada de uma das seguintes formas (embora seu potencial seja limitado): em tempo de compilação, como em OpenC++, ou introduzindo estruturas representando o código do programa. No primeiro método todas as ações reflexivas são estáticas. No último os limites são representados por quais aspectos do código são materializados (CAZZOLA,1998).

A reflexão comportamental pode ser definida como a capacidade da linguagem de permitir uma completa materialização de sua própria semântica, bem como uma completa materialização dos dados que ela usa para executar o programa atual. A manipulação é realizada em 2 fases: look-up dos métodos (ação *shift-up*) e aplicação de mensagens (ação *shift-down*). Na primeira fase, quando a entidade base envia uma mensagem, a meta-entidade a intercepta e procura por uma meta-computação relativa àquela mensagem. Na última fase, a meta entidade aplica, se necessário, a meta-computação à mensagem requisitada e retorna o resultado para a entidade base. Desse modo, o fluxo computacional desloca-se do nível base para o meta-nível e novamente para o nível base (CAZZOLA,1998).

Segundo Zancanella e Navaux (1993), o conceito de computação reflexiva no contexto do projeto **Muse** provê a habilidade de alterar o comportamento dos objetos dinamicamente. Cada objeto é suportado por um ou mais meta-objetos, que constituem seus meta-espaços. Um meta-espaço pode ser visto como uma máquina virtual para o objeto, ou um sistema operacional otimizado para o objeto.

Ainda segundo Zancanella e Navaux (1993),

> *"... a arquitetura **Muse** contém os conceitos de estrutura de kernel coletivo e estruturas baseadas em objetos, porém, diferentemente da estrutura de kernel coletivo, um objeto é, fundamentalmente, uma entidade e diferentemente da estrutura baseada em objetos, um objeto é definido como uma ferramenta de computação reflexiva."*

As técnicas de reflexão computacional têm sido utilizadas em vários campos, por exemplo, para desenvolvimento de sistemas operacionais (YOKOTE,1992,GOWING e CAHILL,1995), sistemas tolerantes a falhas (ANCONA et al.,1995, FABRE et al.,1995) e compiladores (LAMPING et al.,1992). Alguns destes trabalhos em sistemas operacionais incluem reflexão como mecanismo para permitir a extensão e ajuste de funções. Outros trabalhos enfocam problemas mais restritos, tais como; tolerância a falhas, sistemas de tempo-real, sincronização de sistemas concorrentes (MITCHELL et al.,1997) e esquemas para tratamento de exceções.

Segundo Golm e Kleinöder (1997), a adaptação de serviços do sistema operacional com uma granularidade muito fina não é possível nos atuais sistemas baseados em camadas. A estruturação do sistema em termos de nível base e meta-nível, abre a possibilidade da implementação de serviços do sistema de forma mais adequada. Na abordagem **Meta Java**, os serviços são implementados como meta objetos e podem ser substituídos por outros meta objetos na medida em que a aplicação necessitar.

Segundo Cazzola (1998), vários modelos de reflexão computacional (meta-classes, meta-objetos, message reification e channel reification) têm sido desenvolvidos e cada modelo tem diferentes facilidades ausentes nos demais modelos, o que os torna mais adequados para tarefas específicas. O tipo de reflexão suportada pelo modelo reflexivo especifica que aspecto do sistema pode ser monitorado e alterado pelas meta-entidades. Contudo, os tipos de reflexão não são mutuamente exclusivos. Ao contrário, um modelo reflexivo pode suportar mais de um tipo de reflexão, mas o problema é que nem todas as linguagens orientadas a objetos apresentam as facilidades necessárias para suportar todos os tipos de reflexão de uma forma simples.

10.2.12 Estruturas Baseadas em Conhecimento

As estruturas baseadas em conhecimento têm como característica a existência de uma base de conhecimento, a partir da qual se propõe a criação de um suporte mais inteligente para as aplicações e melhores ambientes para o usuário.

Segundo Li et al. (1995), a idéia de um sistema operacional baseado em conhecimento não é nova.

> *"Fleish e Blair já haviam reconhecido que a solução para os problemas detectados em sistemas operacionais (os quais serão discutidos nos capítulos seguintes) poderia ser viabilizada através do que eles chamaram, respectivamente, de: Sistema Operacional Inteligente e Sistema Operacional Baseado em Conhecimento." Li et al. (1995).*

Além disso, complementam Li et al. (1995), outros sistemas experimentais também seguiram esta direção, como: **UC** (VILENSKY,ARENS e CHIN,1984) , que propunha melhorar a interface de interação com o sistema operacional; **PIMOS** (CHIKAYAMA,1988), que apresentou uma máquina de inferência paralela, e as propostas de Korner(1990) e Pasquale (1988) no sentido de implementar mecanismos de gerenciamento inteligente de sistemas distribuídos.

Segundo Zancanella e Navaux (1993), o modelo de objetos é utilizado como ponto de partida ao desenvolvimento da estrutura baseada em conhecimento, de modo que as informações semânticas mantidas na base de dados, e tidas como a chave para prover a inteligência e a integração requerida pelo sistema operacional, dizem respeito à caracterização dos objetos e ao inter-relacionamento entre os mesmos.

O modelo proposto por Nicol (1987), assemelha-se ao modelo de redes semânticas de objetos, onde o conhecimento é representado de forma explícita e concisa. Novos fatos sobre objetos podem facilmente ser adicionados à rede semântica e todo o relacionamento para um dado objeto pode ser determinado sem uma pesquisa extensiva.

O projeto **Genera** (GENERA,1990), descreve uma solução baseada em um produto de hardware denominado Symbolics Common Lisp. **Genera** é um ambiente completo. Todas as atividades são constituídas por funções Lisp, baseadas em uma biblioteca genérica (Figura 89). Abaixo deste nível de software há o hardware, de tal forma que o usuário (ou programador) não tem acesso ao mesmo. Em outras palavras, o processador implementa a máquina LISP em hardware. Esta prática não é recente. Sansonnet et al. (1982), relatam a implementação de um processador LISP em hardware (utilizando microprogramação) e destacam os ganhos de performance que esta estratégia permite em relação às técnicas tradicionais de interpretação de máquinas abstratas.

O sistema Genera possui uma forma de registro histórico de quase tudo o que nele acontece (histórico de comandos, resultados, janelas e assim por diante). Estes históricos tornam possível a utilização de uma saída como entrada de um novo comando e esta é a chave da principal característica do sistema – o compartilhamento de conhecimento e informação entre todas as atividades resulta em um sistema que apresenta um comportamento relativamente inteligente. O conjunto de todos os estados de todos os processos é chamado de ambiente. Qualquer função executando em qualquer processo pode usar qualquer porção de dados do ambiente diretamente. Não há necessidade de programas especiais ou arquivos intermediários para recuperar este tipo de informação (GENERA,1990).

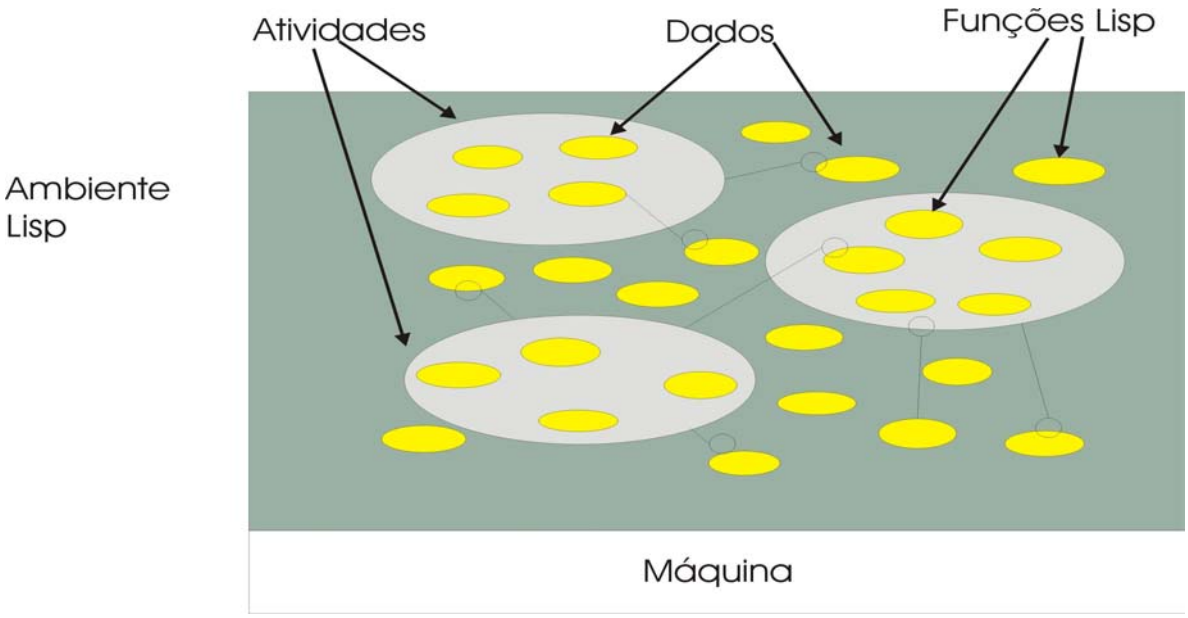


Figura 89 - O ambiente Lisp consistindo de todas as funções e objetos em memória virtual.
FONTE: Genera (1990).

O ambiente todo é chamado de *world* e é salvo em disco de tal forma a estar sempre pronto para ser inicializado (*boot*). Este modelo de mundo contém as funções, variáveis, janelas, processos, históricos, etc. Quando o usuário escreve um novo programa, ele passa a fazer parte do ambiente como uma extensão ao Genera, enriquecendo-o. Algumas características deste ambiente são (GENERA,1990):

- Operações genéricas: quando um usuário necessita adicionar 2 números, por exemplo, não há necessidade de identificar o tipo das variáveis envolvidas (inteiro, real, etc.). Utiliza-se a função genérica de soma (+) e o sistema encarrega-se de somar adequadamente as variáveis. Para garantir a eficiência do sistema, este tipo de operação é tratado por componentes de hardware especiais;
- Operações genéricas definidas pelo usuário: através da utilização de linguagens orientadas a objetos (Flavors, Clos), o sistema permite a utilização dos conceitos de encapsulamento, herança, etc., o que permite a criação de estruturas abstratas de forma relativamente simples;
- Gerenciamento automático de memória: os programadores não necessitam explicitamente alocar ou liberar memória. O espaço de memória é disponibilizado aos programas implicitamente, à medida da necessidade, e liberado automaticamente quando não é mais necessário, através de um garbage collector implementado através da utilização de hardware dedicado;
- Ligação dinâmica: em ambientes convencionais, para se gerar um programa executável são necessários vários passos. Em Genera, é necessário somente o passo de compilação. Uma vez que o código é verificado, ele é automaticamente incluído na biblioteca do sistema;
- Integração ao nível de dados: todas as funções e dados existem em memória virtual, a qual é compartilhada por todos os processos;

- Arquitetura aberta: todas as partes do sistema são disponíveis para inspeção, reuso, extensão ou substituição, ao contrário da maioria dos sistemas operacionais que possuem um núcleo protegido contra modificações e, eventualmente, não disponível aos programas de aplicação;
- Extensibilidade: o sistema é construído de forma a adequar-seàs preferências e características individuais dos usuários;
- Auto-revelação: as informações sobre o funcionamento interno do sistema estão sempre disponíveis.

Sendo uma máquina Lisp, a questão de proteção deste ambiente que disponibiliza suas estruturas internas ao usuário é garantida da seguinte forma (GENERA,1990):

- A integração ao nível de dados significa que, ao invés de conter um conjunto de bits desconexo que, na verdade, somente a aplicação sabe o que significa, a memória em Genera contém objetos de dados estruturados. Isto garante que programas automaticamente compartilhem, além dos objetos, a semântica associada, o que torna impossível para um programa fazer uso indevido das estruturas de dados de outro programa;
- O hardware provê consistência de tipo, limites de arrays e gerência de memória em tempo de execução. Em sistemas tradicionais, os programas têm que ser protegidos uns dos outros tendo em vista a possibilidade de problemas de programação, tais como: ponteiros não inicializados, acesso além da área de memória alocada, liberação de memória ainda em uso, etc. A verificação por hardware em máquinas Symbolics, evita estas fontes tradicionais de problemas;
- Usando conceitos de orientação a objetos, a substituição de parte do sistema ocorre através da sobreposição (herança) de partes do código original, ao invés de sua remoção. Assim, o sistema original permanece disponível para outros programas.

Além da possibilidade de expansão acima descrita, o sistema foi projetado para informar ao usuário o que está acontecendo. Isto conduz ao princípio de maximização da informação (*maximally informative interfaces*), como suporte à exploração e investigação. Além disso, o sistema fornece feedback e documentação sobre o seu estado, o que permite ao usuário ter uma noção mais precisa do que pode estar acontecendo. A qualquer momento, um programa pode ser suspenso e seu estado interno consultado através de uma ferramenta de depuração. Contudo, alguns problemas decorrentes deste tipo de ambiente são identificados (GENERA,1990):

- Como não há uma distinção formal entre o sistema e o usuário, o segundo pode sentir-se tentado a resolver um problema utilizando primitivas ao invés de facilidades mais apropriadas ao problema;
- Documentação: o fato de que o usuário pode utilizar qualquer subconjunto de facilidades faz com que aumente o tamanho e a complexidade do conjunto de documentação disponível. Um outro aspecto importante é que a documentação é sobre funções Lisp, o que demanda um conhecimento mais aprofundado do usuário;
- Questões de compatibilidade: na medida em que o sistema evolui, surgem questões de compatibilidade entre novas versões do sistema e o código gerado pelo usuário baseado em bibliotecas prescritas.

Larner (1990) descreve a implementação de uma estrutura de blackboard baseada em sistemas operacionais para controle de tempo real denominada **DosBart**. Segundo Englemore e Morgan (1988), arquiteturas do tipo blackboard são um conceito estabelecido no campo da inteligência artificial. Entre as várias qualidades que esta abordagem apresenta, pode-se destacar as seguintes:

- Modularidade na representação de conhecimento e técnicas de inferência;
- Liberdade em raciocinar sobre suas próprias estruturas de controle;
- Existência de mecanismos para implementar uma variedade de estratégias de raciocínio e representação de conhecimento.

Larner (1990) complementa que as abstrações fornecidas pela abordagem são adequadas para tratar com muitas das questões relativas ao controle em tempo real. Neste caso, controle em tempo real significa que o estado do sistema é ativamente controlado de modo a proceder de acordo com uma seqüência de valores desejados, ou permitidos, durante o tempo em que o próprio sistema está executando. Além disso, as ações de controle são tomadas para garantir que estas seqüências sejam decididas no mesmo período de execução. Ou seja, o sistema adapta-se dinamicamente às alterações percebidas no ambiente controlado.

Sobre esta última característica, algumas questões são apresentadas por Larner (1990), quais sejam:

- Tratamento de eventos internos e externos;
- Controle de ativação, suspensão e reativação de tarefas;
- Gerenciamento de prazos (*deadlines*) e prioridades;
- Simultaneidade de ações; e
- Alocação de recursos.

Um aspecto que Larner (1990) destaca é que o sistema **DosBart** não é baseado em um sistema operacional distribuído, mas distribui-se sobre várias plataformas usando as facilidades fornecidas pelos sistemas operacionais hospedeiros.

Este enfoque de um lado aproveita as vantagens da técnica de blackboard e, de outro, os benefícios da distribuição de capacidade de processamento. Contudo, devem ser considerados aspectos relativos à sincronização, replicação de dados, consistência, protocolos de comunicação e algoritmos distribuídos (LARNER,1990).

Por outro lado, a execução paralela de fontes de conhecimento (*source knowledge*) caracteriza um poderoso princípio conceitual de arquiteturas blackboard, apesar de não ser muito freqüente a sua implementação. Normalmente, as atividades de inferência são consideradas atômicas e executadas até o fim antes da próxima atividade de inferência ser iniciada. Através do paralelismo inerente às arquiteturas de sistemas operacionais, via fatias de tempo, e considerando a disponibilidade de uma rede interligando outros processadores, o projeto **DosBart** implementa a possibilidade de execução de várias linhas de inferência em paralelo (LARNER,1990).

Um aspecto relevante nesta proposta é o conceito de *knowledge activity* (KA). Uma KA utiliza um módulo provador de teoremas para criar novos objetos ou para implementar qualquer tipo de estratégia de solução ou controle desejados. Por exemplo, uma rotina de monitoramento e diagnóstico no modelo de linha de produção tenta provar que um determinado servidor está com defeito. Quando a prova resultar verdadeira, imediatamente atividades de resposta são disparadas tendo em vista reorganizar a linha de produção. Segundo Larner (1990), embora este nível de liberdade possa ser perigoso, ele foi adotado tendo em vista não restringir a capacidade do *blackboard* a estratégias de controle pré-determinadas. Outro aspecto relevante é que a experiência na construção do sistema **DosBart** serviu para demonstrar, entre outras, a aplicabilidade de linguagens e técnicas de inteligência artificial (*blackboard* em particular) na construção de *frameworks* de controle em tempo real versáteis e distribuídos (Larner,1990).

Li et al. (1995) apresentam um sistema operacional (experimental) inteligente denominado **KZ2**. O texto a seguir está baseado numa compilação das informações apresentadas no trabalho "*An Introduction to Intelligent Operating System KZ2*".

Segundo Li et al. (1995), o sistema **KZ2** caracteriza-se como a nova geração de sistemas operacionais capazes de gerenciar recursos de sistemas maciçamente paralelos, apresentar uma interface amigável de comunicação homem-máquina e controlar a execução de programas baseando-se em processamento de conhecimento e processamento paralelo. As funções implementadas até o momento da publicação incluíam:

- Um escalonador de tarefas inteligente;
- Uma interface homem-máquina inteligente, e
- Um mecanismo para suporte a processamento paralelo/distribuído e a processamento de conhecimento.

O sistema KZ2 apresenta as seguintes características principais:

- Multi-computadores: ele suporta computação distribuída e processamento paralelo baseado em um sistema com massivo número de computadores conectados (fracamente ou fortemente acoplados) através de redes que cooperativamente suportam a execução de aplicações;
- Processamento de conhecimento: utiliza conhecimento para gerenciar os recursos do sistema e para controlar a comunicação homem-máquina. Além disso, possui um mecanismo para processamento eficiente de conhecimento disponível tanto para o sistema como para as aplicações;
- Inteligência: utiliza técnicas de IA para resolver problemas, particularmente as indeterminações encontradas em gerenciamento de recursos de sistemas computacionais maciçamente paralelos, e apresentar um comportamento mais inteligente. A capacidade de aprendizagem do sistema pode também melhorar (*improve*) suas funções e a própria performance;
- Orientado a novatos: o processamento paralelo/distribuído que, em algum grau, é difícil tanto para usuários novatos como para especialistas, pode ser transparente. Além disso, uma interface em linguagem natural permite que o usuário não necessite estar familiarizado com as linguagens de comando dos sistemas operacionais tradicionais.

A estratégia dos projetistas do **KZ2** para quebrar a barreira entre hardware de sistemas computacionais maciçamente paralelos e usuários/aplicações, pode ser resumida a partir da seguinte afirmação: "kernel out e Shell in". Segundo os autores, até o momento da publicação o sistema possuía um escalonador para uso eficiente dos recursos do processador, uma interface em linguagem natural para manipulação do sistema e um mecanismo de inferência paralelo. Tanto o escalonador como a interface são baseados em processamento de conhecimento e inferência.

O núcleo do sistema operacional **KZ2** possui as seguintes funções principais:

- Sistema de gerenciamento da base de conhecimento;
- Escalonador Inteligente de tarefas distribuídas.

Os principais módulos de serviços do sistema são:

- Serviços de interface em linguagem natural;
- Serviços de inferência paralela;
- Serviços de redes heterogêneas.

Cabe destacar que os serviços podem ser utilizados tanto pelo sistema operacional como pelas aplicações. Além disso, três tipos de conhecimento são incluídos no módulo de serviços associados à interface de linguagem natural:

- Conhecimento lingüístico: consiste de conhecimento de vocabulário e padrões de sentenças. O conhecimento de vocabulário inclui a formação, uso e significado semântico de palavras e é

representado em frames. O conhecimento de padrões de sentenças (organizado na forma de uma BNF) representa como empregar palavras/frases para construir sentenças e qual o seu significado;

- Conhecimento de domínio: é essencial ao processamento de requisições em linguagem natural. Cada domínio tem seus próprios conceitos e alguns podem ter diferentes significados de um domínio para outro;
- Conhecimento de comando de aplicações: descreve as formas e significados semânticos de todos os comandos disponibilizados às aplicações. Por exemplo, se a aplicação é **Unix**, todos os comandos do sistema operacional, suas regras de formação e significado semântico, são incluídos na base.

Segundo Li et al. (1995) as pesquisas atuais em processamento de linguagem natural não dedicam muita atenção à representação declarativa de conhecimento de domínio e conhecimento de comando de aplicações. Neste sentido o projeto **KZ2** ataca o problema expressando-os declarativa e separadamente, além de disponibilizar um método para estender estes conhecimentos em particular. As requisições de processamento em linguagem natural são atendidas de forma hierárquica em quatro fases:

- Entendimento de linguagem natural: requisições são inicialmente traduzidas para uma forma interna que representa o entendimento inicial;
- Compilação do domínio: esta forma interna é refinada para adequar-se ao domínio utilizando a base de domínio;
- Máquina de inferência: de acordo com o conhecimento de interface de comando de aplicação, a forma interna é convertida pela máquina de inferência em uma seqüência de comandos;
- A seqüência de comandos é executada pela aplicação e um módulo de aprendizado é utilizado para facilitar a tarefa de aquisição de conhecimento interativamente.

O módulo de escalonamento inteligente possui dois tipos de conhecimento:

- Conhecimento de objetos: inclui fatos básicos sobre o componente correspondente. Por exemplo, se o objeto é um processador, são incluídas informações sobre velocidade, tamanho da memória daquele processador em particular, etc. Além disso, são agregadas informações sobre a taxa de ocupação deste processador, a qual é adquirida dinamicamente durante o ciclo de vida do objeto. Além disso, uma rede de crenças (*believes*) é fornecida para refletir a qualidade dos relacionamentos entre tais conhecimentos;
- Regras de escalonamento: estas regras são definidas a partir da experiência dos projetistas de sistemas operacionais. A forma das regras é: C -> A {B}, onde C contém as condições, A contém as ações ou conclusões e B é o peso máximo do relacionamento entre a proposição A e a evidência C.

Todo conhecimento é representado por uma linguagem baseada na teoria da evidência (regras) de Dempster-Shafer. Além disso, há um gerenciador de estados global, o qual observa o comportamento do sistema, formula hipóteses sobre seu estado e as refina

em conhecimento armazenado. Este gerenciador de estados global possui três módulos, quais sejam:

- Interpretador de E/S: é responsável pela coleção de informações do sistema. Após coletar um certo montante de informações observando a comunicação entre os módulos ele gera hipóteses sobre o estado do sistema, tentando explicar as informações recebidas. Cada hipótese é associada a um valor de crença (*belief*) baseado em uma rede de crenças. Quando um valor de crença supera um valor de limiar (*threshold*) significa que há evidências suficientes para suportar esta hipótese e esta torna-se um fato com um valor de crença associado. Esta informação é agregada à base de conhecimento como parte do conhecimento durante a vida de determinado objeto. Por outro lado, à medida que o tempo vai passando, se não há evidências coletadas para suportar uma determinada hipótese ou fato, o valor de crença vai diminuindo gradualmente de acordo com uma regra de atenuação. Caso o valor de crença atinja um limiar inferior, o fato é removido da base de conhecimento;
- Utilizando o conhecimento de estado global e as regras de escalonamento, o gerente de decisões avalia, otimiza e finalmente obtém uma solução aceitável para o problema de escalonamento, o qual não somente atende às demandas de tempo de resposta do usuário, mas também atende aos requisitos de performance e balanceamento de carga do sistema;
- O módulo de tomada de decisões é responsável por raciocinar e obter todas as possíveis soluções de acordo com as regras de escalonamento e fatos na base de conhecimento. O avaliador avalia estas soluções de acordo com um conjunto de critérios (funções de custo) para gerar um conjunto aproximado de escalonamentos cujo custo é aceitável. Se há algum escalonamento aceitável, o módulo tomador de decisões irá aceitá-lo. Caso contrário, o módulo otimizador tentará encontrar algum que seja ou aceitável ou aproximadamente aceitável, baseado nos teoremas e regras de otimização.

Finalmente, o esquema de escalonamento é enviado para o interpretador de E/S para disparar a execução das tarefas. Contudo, devido à imprecisão do conhecimento de estado e da autonomia de cada sistema local, conflitos de escalonamento são inevitáveis. Neste caso, um módulo colaborador resolve tais conflitos através de estratégias de negociação e arbitramento.

O mecanismo de inferência paralelo adota Grafos de Expressões de Execução e Árvores de Execução Paralela para decompor um esquema de controle. Além disso, utiliza um mecanismo de sincronização baseado em produtor/consumidor para controlar a utilização de variáveis compartilhadas no sentido de evitar a produção de resultados inconsistentes durante a execução de esquemas de controle.

Durante o levantamento bibliográfico foram identificadas na página do autor Xie Li (LI ,2003) as seguintes referências bibliográficas:

- KZ10: A knowledge-based Operating System. Publicado no Jornal Chinês de Avanços em Pesquisas em Software (Chinese Journal of Advance Software Research, Vol.1 no. 4, 1994).

- General Natural Language for Operating Systems publicado em SIGART Bulletin, vol.3 no.4, 1993;
- An Adaptive Strategy Integrating Locking With Optimistic Concurrency control publicado em Journal of Compute Sci. & Technnol. Vol.8, no.4, 1993;
- A Knowledge-Based Open Model for Human Computer Interaction publicado em Science in China (Series A), vol 36, no.8, 1993;
- Open model methodology: A New Approach to the Development of User Interfaces based on Knowledge Processing publicado em Software Engineering Journal (England), vol.7,no.3, 1992;
- Accommodating Domain-Independence: A new Approach to the Development of General Natural Language Interfaces publicado em Engineering Application of AI (England) vol.4, no.2, 1992
- KZ1: A Prototype of Intelligent Operating System publicado no Journal of Computer Sci. & Tech. Vol.6, no.3, 1991,

demonstrando que as pesquisas na China, na década de 90, direcionaram esforços no sentido de produzir-se um sistema operacional baseado em conhecimento[122].

10.2.13 Estruturas Baseadas em Agentes

Segundo Kalakota e Whinston (1996), "a metáfora de agentes tem suas raízes em trabalhos realizados no início dos anos 60. A área de Inteligência Artificial utilizou o termo para referir-se a programas que atuam em favor de seus usuários".

Booker et al. (1999) complementam que:

> *"Em 1995, Wooldridge e Jennings publicaram o livro: "Agentes Inteligentes: Teoria e Pratica" (1995) o qual provavelmente contém a primeira definição compreensível de agentes inteligentes, suas propriedades e comportamentos. Ao mesmo tempo, outros modelos não diretamente relacionados à computação estavam sendo desenvolvidos, tais como o projeto Ubiquitous Computing do Palo Alto Research Center.".*

O conceito mais comumente adotado define agente como sendo um sistema que percebe e age em ambientes dinâmicos e imprevisíveis, que coordena capacidades de interoperação entre componentes complementares e que executa serviços úteis com um alto grau de autonomia (BÁRBARA HAYES-ROTH apud BOOKER et al.,1999) Esta definição é complementada a partir das seguintes noções:

- Autonomia: os agentes são capazes de determinar seu próprio estado e o estado do ambiente, bem como de tomar decisões dinâmicas baseadas nestes estados sem a necessidade de intervenção direta de pessoas ou outros componentes;

[122] Não foram encontrados links, na página do autor, para a obtenção dos trabalhos e os contatos realizados através do serviço de comutação de bibliotecas (por e-mail) não permitiram a obtenção de tais informações em tempo hábil para completar esta seção do trabalho.

- Reatividade: os agentes têm a capacidade de perceber o ambiente em que eles estão localizados, utilizando sensores potencialmente imperfeitos, e são capazes de responder e adaptar-se a mudanças neste ambiente;
- Proatividade: além de respostas a mudanças ambientais, os agentes podem apresentar um comportamento orientado por metas tendo em vista cumprir suas tarefas;
- Sociabilidade (*social ability*): os agentes são capazes de se comunicarem com outros agentes e pessoas, de forma a coordenar e negociar planos e estratégias.

Há outras características geralmente propostas como desejáveis, mas não fundamentais, as quais são relacionadas a seguir:

- Continuidade temporal: geralmente considera-se que os agentes executam continuamente e não como os programas tradicionais, que são ativados para executarem tarefas específicas e são encerrados depois de executados;
- Memória: alguns agentes, particularmente os pertencentes à categoria de assistentes pessoais, possuem capacidade de aprender as preferências dos usuários a partir da experiência, de modo a aumentar seu grau de autonomia e utilidade;
- Mobilidade: agentes que são capazes de multiplicarem-se em várias instâncias deles mesmos, para execução em máquinas remotas (com arquiteturas potencialmente diferentes), podem incrementar a sua social ability e agilizar a execução das tarefas a eles destinadas;
- Personalidade (*personality*): agentes educacionais e recreativos (*recreational*) podem apresentar um caráter de crença (*believable*) e estados emocionais;
- Racionalidade: espera-se que os agentes executem suas tarefas de modo que eles não influenciem negativamente outros agentes ou pessoas, talvez em alguns casos através da recusa em executar tarefas que possam ser classificadas como anti-sociais ou perigosas.

Segundo Booker et al. (1999), os atributos acima apresentados podem ser usados para identificar sub-categorias cujas propriedades comportamentais e ambientais diferem. A categoria de agentes de software (contrastando com projetos de robôs autônomos) inclui:

- Assistentes especializados (*expert assistants*) ou guias turísticos (*tour guides*);
- Agentes sintéticos (*synthetic agents*) que operam em mundos virtuais simulados;
- Assistentes pessoais (*personal information assistants*);
- *Softbots* ou Robôs em Software.

A proposta de uso da tecnologia de agentes aparece em pelo menos 2 trabalhos de pesquisa recentes: o projeto **Abos** (SVAHNBERG,1998) e o projeto **AgentOS** (CHEN,2000). Basicamente, a proposta é construir o núcleo do sistema operacional com um conjunto mínimo de recursos que permita a gerência e a intercomunicação entre pequenos módulos autônomos denominados agentes.

Em **ABOS** (SVAHNBERG,1998), os agentes executam em espaços de memória independentes. Com isto, pretende-se atingir um nível de extensibilidade e flexibilidade bastante razoável. Como os agentes são autônomos e podem adaptar-se com o tempo, é possível a criação de um sistema operacional mais flexível utilizando esta tecnologia.

Segundo Svahnberg (1998) a estrutura do **ABOS** é formada por um conjunto de módulos pequenos e autônomos (agentes). Diferentemente dos núcleos tradicionais, onde todos os módulos compartilham o mesmo espaço de memória, os agentes em **ABOS** executam como processos separados. Isto permite modificar a estrutura do sistema sem necessidade de recompilar ou mesmo reinicializar o sistema operacional. **ABOS** é organizado como um conjunto de níveis ao redor de um núcleo. O nível dos serviços aumenta na medida em que há uma transposição de nível mais distante do núcleo.

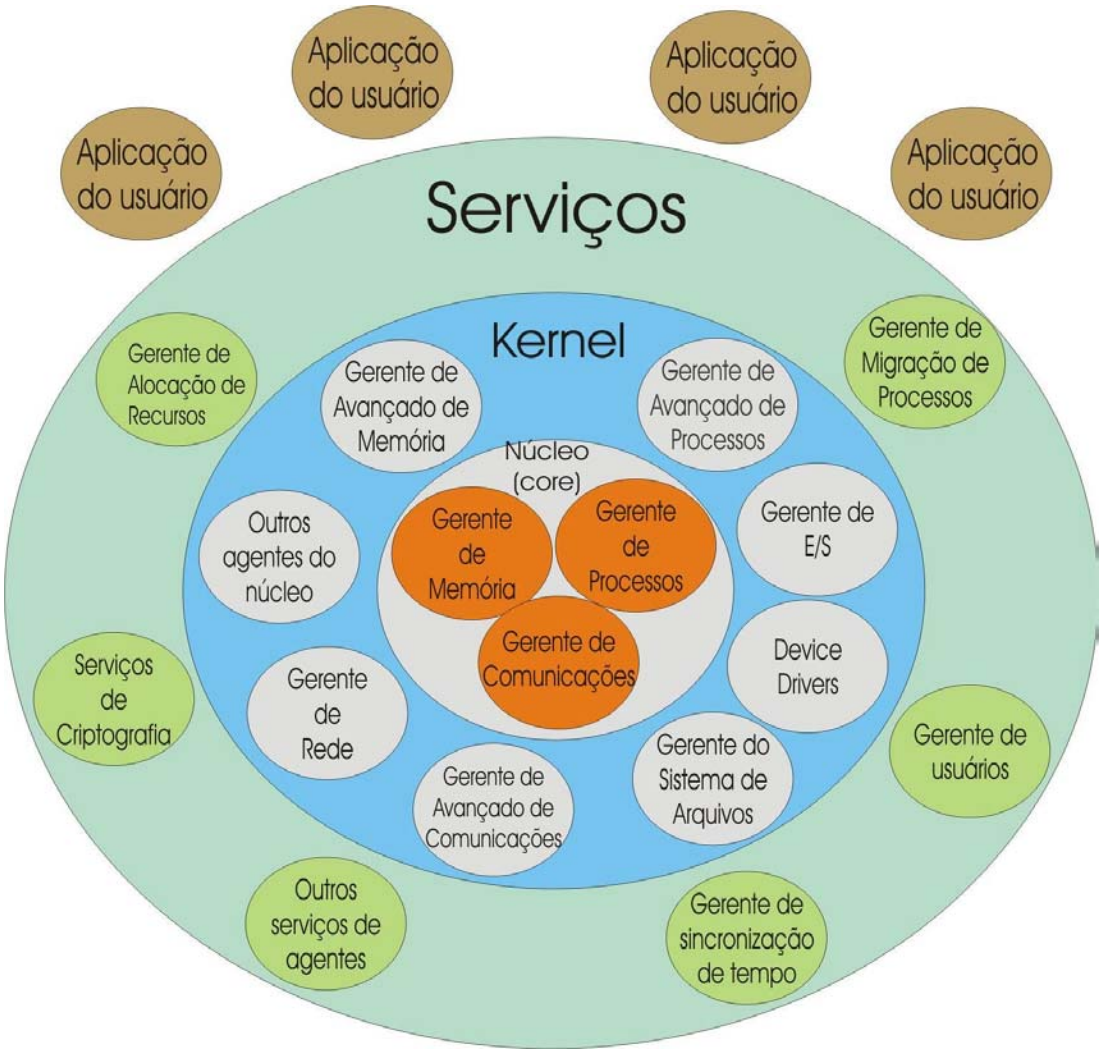


Figura 90 – Estrutura organizacional do sistema ABOS.
FONTE: Adaptado de Svahnberg(1998).

O núcleo suporta as funções básicas de gerenciamento de processos, memória e comunicação entre processos. Não há comportamento inteligente neste nível. A função do núcleo é agir como uma interface com o processador e com a MMU. Cada um dos gerentes no núcleo base pode ter vários gerentes correspondentes no nível do kernel, de modo a prover inteligência adaptada para cada tarefa em particular. Os device drivers e sistema de arquivos também são gerenciados no nível de *kernel*. O nível de serviço provê funcionalidade adicional que não é parte do *kernel*, mas necessária a um sistema operacional como migração de processos e gerenciamento de usuários. As aplicações de usuários executam sobre este nível de serviço (SVAHNBERG,1998).

Já o projeto **Agentos** (CHEN,2000) caracteriza-se como um sistema operacional virtual que suporta o desenvolvimento de aplicações orientadas a objetos e distribuídas através da utilização de agentes. Assim, **Agentos** é um agente-hospedeiro (*agent-host*) que provê suporte para a execução de agentes implementados em Java. O sistema é composto por um pequeno núcleo que estabelece um modelo de execução para agentes, um mecanismo de comunicação baseado em eventos com transparência de rede e um modelo para acesso aos serviços disponíveis.

10.2.14 Estruturas Híbridas

A estratégia adotada pela Microsoft no projeto do sistema operacional **Windows 2000** envolve o conceito de arquitetura híbrida (AHL,2001). Esta arquitetura utiliza conceitos do modelo cliente-servidor, em camadas, orientada a objetos e suporta multiprocessamento simétrico (Figura 91).

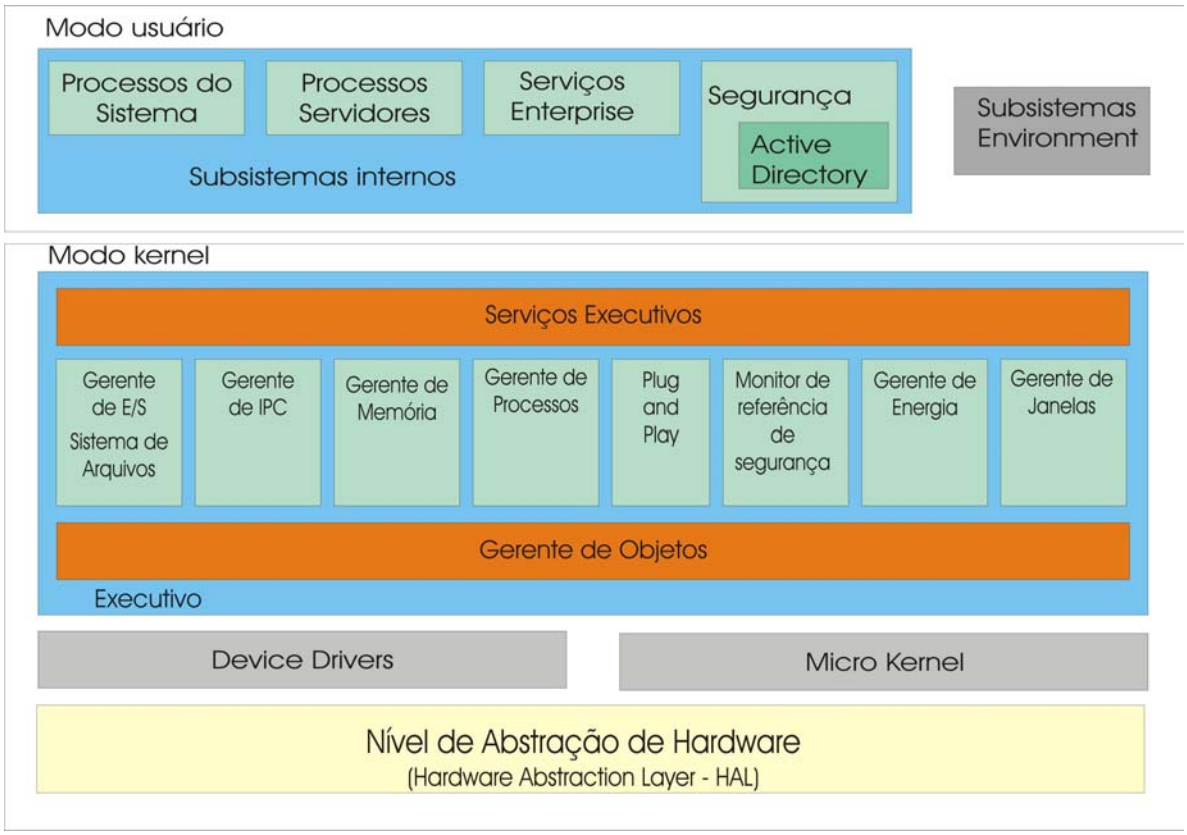


Figura 91- Abordagem híbrida do sistema operacional Windows 2000 Server.

FONTE: Adapado de Ahl (2001).

Um micronúcleo executa sobre um HAL baseado na filosofia do *kernel Mach,* e sobre este micronúcleo, também em modo supervisor, executam o gerente de objetos, o gerente de entrada/saída, o gerente de processos e o sistema de arquivos. Ao conjunto de servidores, juntamente com o micronúcleo, dá-se o nome de ***Windows NT Executive***. A diferença entre os servidores do **NT** e os módulos do **Linux** reside no fato de que os primeiros executam em modo supervisor, porém cada um em um espaço de endereçamento separado, o que protege o núcleo de eventuais problemas em um servidor. O núcleo do **NT** também é orientado a objetos, como o **Mach**, o que significa que todas as abstrações e serviços do sistema são representados como objetos (AHL,2001). Segundo Dumon (1997), o serviço de gerência de memória é muito semelhante ao do Mach, apenas com o nome dos objetos alterados.

Segundo Oney (1999), todo o conhecimento do sistema operacional sobre como o computador está conectado aos vários dispositivos físicos repousa sobre o HAL o qual detém informações sobre como as interrupções funcionam em uma plataforma específica,

como são implementados os spin-locks e como deve ser endereçado o espaço de entrada/saída. Ao invés de acessar o hardware diretamente, os drivers WDM[123] fazem chamadas a primitivas do HAL para solicitar os serviços necessários ao efetivo acesso aos periféricos. Esta solução permite o desenvolvimento de device drivers independentes de plataforma e de barramento (*bus*), o que caracteriza uma vantagem, em se tratando do desenvolvimento de um sistema operacional para várias plataformas de hardware diferentes (ONEY,1999).

Uma outra proposta está baseada nas observações obtidas a partir de pesquisas realizadas nos projetos **FUZOS** – *Fuzzy Operating System* e **SIIIS** - *Software Industry Intelligent Information Systems* (PATKI, RAGHUNATHAN e KHURSHID,1997), realizados na Índia, as quais indicam que as novas estratégias para construção de sistemas operacionais deverão seguir os moldes da construção de sistemas RISC, ou seja, através do particionamento do sistema em duas seções (PATKI e RAGHUNATHAN,1996 e PATKI,1996):

- Um esqueleto deve ser integrado na forma de extensões de hardware e/ou firmware para melhorar os índices de *Machine Intelligent Quotient* (MIQ);
- A outra porção do sistema operacional poderia ser implementada puramente em software.

E Patki e Raghunathan (1996) e Patki (1996), vão além propondo que, para sustentar esta estratégia, os projetistas de hardware e desenvolvedores de arquiteturas VLSI, precisam desenvolver novas estruturas em silício para suportar:

- Processamento de operações difusas ao nível de instruções de hardware;
- Tipos de dados primitivos para operações de transferência ao nível de registradores que permitam adequada manipulação de tuplas (valor no conjunto, valor de pertinência);
- Memória associativa *on-chip* para suportar informações do processador.

Schubert (1997) descreve o projeto **Cheops** onde são aplicados conceitos de orientação a objetos e reflexão computacional para suportar adaptabilidade dinâmica com uma granularidade bastante fina. O sistema está baseado na idéia de cobrir a lacuna que existe entre modelos e abstrações utilizados durante o desenvolvimento (projeto e implementação) e as entidades identificáveis no sistema em execução. Mantendo-se a maioria da informação sobre estruturação das classes, objetos e seus relacionamentos, é

[123] Segundo Oney (1999), o WDM (*Windows Driver Model*) estabelece um *framework* para a construção de *device drivers* que operam nos sistemas operacionais **Windows 98** e **Windows 2000** da Microsoft.

possível aplicar extensões e modificações no sistema em execução, da mesma forma como ocorre na sua descrição (ao nível de código fonte).

Com base nos projetos acima apresentados, pode-se antecipar que, a partir do momento em que novas arquiteturas de processadores, incorporando os conceitos de lógica difusa e redes neurais, venham a ser disponibilizadas, um grande espectro de novas aplicações deverá surgir e, naturalmente, novas gerações de sistemas operacionais deverão ser construídas para suportar esta evolução.

10.2.15 Considerações finais

O presente apêndice apresentou uma revisão das diversas abordagens de construção de núcleos de sistemas operacionais. O levantamento realizado permitiu a identificação dos principais modelos, suas características, vantagens e desvantagens com as respectivas justificativas além de um elenco de sistemas enquadrados em cada categoria. Estas informações foram compiladas e resumidas no capítulo 2(seção 2.4 ,pág.65).

10.3 Apêndice 3 – Exemplo de log de erro no sistema (DrWatson –Windows)

Este apêndice apresenta um extrato de um arquivo de log de erro (**drwtsn32.log**) obtido durante a execução de um aplicativo em ambiente **Windows**.

Microsoft (R) Windows 2000 (TM) Versão 5.00 DrWtsn32
Copyright (C) 1985-1999 Microsoft Corp. Todos os direitos reservados.
Exceção de aplicativo:
Aplicativo: (pid=316)
Data e hora: 22/7/2003 às 14:49:27.007
Número da exceção: c0000005 (violação de acesso)

----> Informação do sistema <----
Nome do computador: XYB-732
Nome de usuário: Administrador
Número de processadores: 1
Tipo de processador: x86 Family 6 Model 8 Stepping 3
Versão do Windows 2000: 5.0
Versão atual: 2195
Service Pack: None
Tipo atual: Uniprocessor Free
Empresa registrada: FURB - Universidade Regional de Blumenau
Proprietário registrado: Tese

----> Lista de tarefas <----

0 Idle.exe	8 System.exe	160 smss.exe
184 csrss.exe	180 winlogon.exe	232 services.exe
244 lsass.exe	380 svchost.exe	428 svchost.exe
536 regsvc.exe	552 MSTask.exe	752 Explorer.exe
764 msiexec.exe	876 tp4mon.exe	900 internat.exe
496 spoolsv.exe	316 WINWORD.exe	492 AgentSvr.exe
540 drwtsn32.exe		
0 _Total.exe		

(30000000 - 30836000) (77F80000 - 77FFD000) (77DB0000 - 77E0A000)
....

(50700000 - 50712000) (0AB00000 - 0ABB6000) (76B70000 - 76BDF000) (77820000 - 77827000)
(0A9E0000 - 0AB00000)

Despejo de estado para o identificador do segmento 0x264
eax=0a5ea904 ebx=0a5ea940 ecx=0000014b edx=7361646e esi=0a5ea7f4 edi=0a5ed3cc
eip=0aa36659 esp=0012fc90 ebp=0012fcb0 iopl=0 nv up ei pl zr na po nc
cs=001b ss=0023 ds=0023 es=0023 fs=003b gs=0000 efl=00000246

função: GramStatsDlg

0aa36637	c745fc00000000	mov	dword ptr [ebp+0xfc],0x0	ss:00bad286=????????
0aa3663e	8b4d0c	mov	ecx,[ebp+0xc]	ss:00bad286=????????
0aa36641	8b5110	mov	edx,[ecx+0x10]	ds:00a7d721=????????
0aa36644	8955f0	mov	[ebp+0xf0],edx	ss:00bad286=????????
0aa36647	837df000	cmp	dword ptr [ebp+0xf0],0x0	ss:00bad286=????????
0aa3664b	7421	jz	0aa3f16e	
0aa3664d	8b4508	mov	eax,[ebp+0x8]	ss:00bad286=????????
0aa36650	33c9	xor	ecx,ecx	
0aa36652	668b4802	mov	cx,[eax+0x2]	ds:0b067edb=????
0aa36656	8b55f0	mov	edx,[ebp+0xf0]	ss:00bad286=????????
FALHA -> 0aa36659	**390a**	**cmp**	**[edx],ecx**	**ds:7361646e=????????**
0aa3665b	7411	jz	0aa3f16e	
0aa3665d	8b45f0	mov	eax,[ebp+0xf0]	ss:00bad286=????????
0aa36660	8945fc	mov	[ebp+0xfc],eax	ss:00bad286=????????
0aa36663	8b4df0	mov	ecx,[ebp+0xf0]	ss:00bad286=????????
0aa36666	8b5104	mov	edx,[ecx+0x4]	ds:00a7d721=????????
0aa36669	8955f0	mov	[ebp+0xf0],edx	ss:00bad286=????????
0aa3666c	ebd9	jmp	0aa3ea47	
0aa3666e	837df000	cmp	dword ptr [ebp+0xf0],0x0	ss:00bad286=????????
0aa36672	7526	jnz	0aa3d09a	
0aa36674	6a38	push	0x38	
0aa36676	6a40	push	0x40	

----> Rastreamento regressivo da pilha<----
FramePtr ReturnAd Param#1 Param#2 Param#3 Param#4 Nome da função
0012FCB0 0AA32499 0A5EA904 0A5EA940 00000000 00000000 !GramStatsDlg
0012FCD4 3024149F 0A5E0001 00000001 0A5EA86E 0A5EA904 !GramCheck
0012FE40 3024206F 0203D3CC 0A5ED3E4 0A5ED3E8 0012FE6C !<nosymbols>
0012FE74 30235B57 0A5EA7F4 0A5ED3CC 00000001 00000000 !<nosymbols>
0012FEA4 30226C46 0009B5C2 00000204 3003A4D2 00000000 !<nosymbols>
0012FEC4 302265DB FFEFFFFF 00100000 30807DE0 00000000 !<nosymbols>
00000000 00000000 00000000 00000000 00000000 00000000 !<nosymbols>

----> Despejo simplificado da pilha<----

```
0012fc90  fb 01 00 00 9f 00 00 00 - 15 00 00 00 fe 01 00 00  ................
0012fca0  6e 64 61 73 00 00 00 00 - fe 01 00 00 28 b0 1e 00  ndas........(...
0012fcb0  d4 fc 12 00 99 24 a3 0a - 04 a9 5e 0a 40 a9 5e 0a  .....$....^.@.^.
0012fcc0  00 00 00 00 00 00 00 00 - 01 00 00 00 00 00 00 00  ................
0012fcd0  00 00 73 00 40 fe 12 00 - 9f 14 24 30 01 00 5e 0a  ..s.@.....$0..^.
0012fce0  01 00 00 00 6e a8 5e 0a - 04 a9 5e 0a 40 a9 5e 0a  ....n.^...^.@.^.
0012fcf0  cc d3 5e 0a 00 00 00 00 - f4 a7 5e 0a 84 00 00 00  ..^.......^.....
0012fd00  b8 1a 42 00 88 15 3b 00 - 1a 00 00 00 78 00 00 00  ..B...;.....x...
0012fd10  ee 03 00 00 bf 02 00 00 - a5 06 00 00 00 00 00 00  ................
0012fd20  00 00 00 00 00 00 00 00 - 00 00 00 00 8a 01 06 00  ................
0012fd30  00 00 00 00 00 00 00 00 - 72 01 00 00 61 01 00 00  ........r...a...
0012fd40  78 fd 12 00 b2 51 e1 77 - 00 00 00 00 04 00 00 00  x....Q.w........
0012fd50  00 00 00 00 00 00 82 00 - 00 00 ac 01 00 00 15 00  ................
0012fd60  00 00 c0 b2 14 00 52 00 - 00 00 00 00 10 00 00 00  ......R.........
0012fd70  00 00 00 00 00 00 00 00 - fc fd 12 00 d2 54 e1 77  .............T.w
0012fd80  8a 01 06 00 84 00 00 00 - 00 00 00 00 00 00 00 00  ................
0012fd90  00 00 00 00 39 cf 1d 30 - 8a 01 06 00 fe b2 09 00  ....9..0........
0012fda0  41 00 00 00 47 00 00 00 - cc 85 41 0d 00 00 00 00  A...G.....A.....
0012fdb0  43 00 00 00 ff ff ff ff - 00 00 00 00 01 00 00 00  C...............
0012fdc0  e8 d3 5e 0a 01 00 00 00 - 00 00 00 00 00 00 00 00  ..^.............
```

Despejo de estado para o identificador do segmento 0x3f8

```
eax=003b0650  ebx=00000d00  ecx=000083e8  edx=00000000  esi=00f1ff98  edi=77e1909c
eip=77e1414f  esp=00f1ff38  ebp=00f1ff58  iopl=0         nv up ei pl zr na po nc
cs=001b  ss=0023            ds=0023       es=0023       fs=0038       gs=0000
efl=00000246
```

função: DispatchMessageW

```
77e14137 90                nop
77e14138 ffff              ???
77e1413a ffff              ???
77e1413c 7407              jz      DrawFrame+0xae4 (77e22545)
77e1413e e477              in      al,77
77e14140 8707              xchg    [edi],eax       ds:77e1909c=53ec8b55
77e14142 e477              in      al,77
77e14144 b89a110000        mov     eax,0x119a
77e14149 8d542404          lea     edx,[esp+0x4]   ss:0199d50f=46454752
77e1414d cd2e              int     2e
77e1414f c21000            ret     0x10
77e14152 55                push    ebp
77e14153 8bec              mov     ebp,esp
77e14155 53                push    ebx
77e14156 56                push    esi
77e14157 8b7508            mov     esi,[ebp+0x8]           ss:0199d52e=61657361
77e1415a 8b450c            mov     eax,[ebp+0xc]           ss:0199d52e=61657361
77e1415d 57                push    edi
77e1415e 33ff              xor     edi,edi
77e14160 0fb74e2a          movzx   ecx,word ptr [esi+0x2a] ds:0199d56f=6f69
77e14164 81e1ff3fffff      and     ecx,0xffff3fff
77e1416a 7521              jnz     UnregisterClassA+0x13f (77e1c68d)
```

----> Rastreamento regressivo da pilha<----

```
FramePtr ReturnAd Param#1  Param#2  Param#3  Param#4  Nome da função
00F1FF58 30A62BF7 00F1FF98 00000000 00000000 00000000 user32!DispatchMessageW
00F1FFB4 77E92CA8 00000000 00000000 3019004A 00000000 !MsoFCustomAWDatabases
00F1FFEC 00000000 00000000 00000000 00000000 00000000 kernel32!CreateFileA
```

Despejo de estado para o identificador do segmento 0x1b0

```
eax=77542bdf  ebx=00000002  ecx=00000000  edx=00000000  esi=77f87e6c  edi=00000002
eip=77f87e77  esp=0a8aff24  ebp=0a8aff70  iopl=0         nv up ei pl zr na po nc
cs=001b       ss=0023       ds=0023       es=0023       fs=0038       gs=0000
efl=00000246
```

função: ZwWaitForMultipleObjects

```
77f87e6c b8e9000000        mov     eax,0xe9
77f87e71 8d542404          lea     edx,[esp+0x4]           ss:0b32d4fb=????????
77f87e75 cd2e              int     2e
77f87e77 c21400            ret     0x14
77f87e7a 668b08            mov     cx,[eax]                ds:77542bdf=8b55
77f87e7d 40                inc     eax
77f87e7e 40                inc     eax
77f87e7f 8945a4            mov     [ebp+0xa4],eax          ss:0b32d546=????????
77f87e82 6685c9            test    cx,cx
77f87e85 75f3              jnz     RtlExpandEnvironmentStrings_U+0x26 (77f8e57a)
77f87e87 663930            cmp     [eax],si                ds:77542bdf=8b55
77f87e8a 75ee              jnz     ZwFsControlFile+0x54 (77f8bf7a)
77f87e8c 40                inc     eax
77f87e8d 40                inc     eax
77f87e8e 8945a4            mov     [ebp+0xa4],eax          ss:0b32d546=????????
```

----> Rastreamento regressivo da pilha<----

```
FramePtr ReturnAd Param#1  Param#2  Param#3  Param#4  Nome da função
0A8AFF70 77E9E68A 0A8AFF48 00000001 00000000 00000000 ntdll!ZwWaitForMultipleObjects
0A8AFFB4 77E92CA8 00000000 7FFDEBF8 00000000 00000000 kernel32!WaitForMultipleObjects
0A8AFFEC 00000000 00000000 00000000 00000000 00000000 kernel32!CreateFileA
```

10.4 Apêndice 4 – Exemplo de log obtido através de device driver (Windows)

Este apêndice apresenta um extrato de um arquivo de log de operações de acesso a disco ocorrido durante o processo de estabelecimento de conexão com a internet, através do serviço de dial-up do Windows98. Os dados foram obtidos por meio de um virtual device driver desenvolvido com o propósito de receber notificações de operações de acesso a disco e registrá-las em um arquivo de log.

Seq.	Tempo	Programa	Oper	Path	Status	Complemento
0	0.00012720	MSGSRV32	At	C:\WIN\USER.DAT	OK	GetAttr
1	0.00060720	MSGSRV32	At	C:\WIN\USER.DAT	OK	SetAttr
2	0.00030320	MSGSRV32	Op	C:\WIN\USER.DAT	OK	OPEXISTING WRITEONLY COMPATIBILITY
3	0.00001200	MSGSRV32	Sk	C:\WIN\USER.DAT	OK	Beg. Off.: 32 / New Off.: 32
4	0.00004080	MSGSRV32	Wr	C:\WIN\USER.DAT	OK	Off.: 32 Len.: 32
5	0.00000880	MSGSRV32	Sk	C:\WIN\USER.DAT	OK	Beg. Off.: 28704 / New Off.: 28704
6	0.00051760	MSGSRV32	Wr	C:\WIN\USER.DAT	OK	Off.: 28704 Len.: 28672
. . . . .						
17	0.00001040	MSGSRV32	Sk	C:\WIN\USER.DAT	OK	Beg. Off.: 1171488 / New Off.: 1171488
18	0.00180560	MSGSRV32	Wr	C:\WIN\USER.DAT	OK	Off.: 1171488 Len.: 61440
19	0.00006960	MSGSRV32	Cl	C:\WIN\USER.DAT	OK	CL_FINAL
20	0.00010480	MSGSRV32	At	C:\WIN\USER.DAT	OK	GetAttr
21	0.00054320	MSGSRV32	At	C:\WIN\USER.DAT	OK	SetAttr
22	0.00019200	Explorer	Rd	C:\WIN\SYSTEM\COMCTL32.DLL	OK	Off.: 340992 Len.: 4096
. . . . .						
25	0.01217520	Explorer	Rd	C:\WIN\SYSTEM\BROWSEUI.DLL	OK	Off.: 386048 Len.: 4096
26	0.00098480	Explorer	**Op**	C:\WIN\APPLICATION DATA\\INTERNET EXPLORER\QUICK LAUNCH\ATALHO PARA PROVEDOR.LNK	OK	OPEXISTING READONLY DENYNONE
27	0.00007200	Explorer	**Rd**	C:\WIN\APPLICATION DATA\\INTERNET EXPLORER\QUICK LAUNCH\ATALHO PARA PROVEDOR.LNK	OK	Off.: 0 Len.: 4096
28	0.00005200	Explorer	**Cl**	C:\WIN\APPLICATION DATA\\INTERNET EXPLORER\QUICK LAUNCH\ATALHO PARA PROVEDOR.LNK	OK	CL_FINAL
29	0.00040480	Explorer	Op	C:\WIN\APPLICATION DATA\\INTERNET EXPLORER\QUICK LAUNCH\ATALHO PARA PROVEDOR.LNK	OK	OPEXISTING READONLY DENYNONE
30	0.00005760	Explorer	Rd	C:\WIN\APPLICATION DATA\\INTERNET EXPLORER\QUICK LAUNCH\ATALHO PARA PROVEDOR.LNK	OK	Off.: 0 Len.: 4096
31	0.00004000	Explorer	Cl	C:\WIN\APPLICATION DATA\\INTERNET EXPLORER\QUICK LAUNCH\ATALHO PARA PROVEDOR.LNK	OK	CL_FINAL
32	0.00029760	Explorer	At	C:\WIN	OK	GetAttr
33	0.00006480	Explorer	At	C:\WIN\RUNDLL32.EXE	OK	GetAttr
34	0.00031760	Explorer	Op	C:\WIN\RUNDLL32.EXE	OK	OPEXISTING READONLY DENYWRITE
35	0.00004640	Explorer	Rd	C:\WIN\RUNDLL32.EXE	OK	Off.: 0 Len.: 64
36	0.00001920	Explorer	Sk	C:\WIN\RUNDLL32.EXE	OK	Beg. Off.: 128 / New Off.: 128
37	0.00002160	Explorer	Rd	C:\WIN\RUNDLL32.EXE	OK	Off.: 128 Len.: 248
38	0.00003840	Explorer	Cl	C:\WIN\RUNDLL32.EXE	OK	CL_FINAL
39	0.00013680	Explorer	Directory	C:\WIN	OK	CHECK
40	0.00006480	Explorer	FindOp	C:\WIN\WIN.INI	OK	WIN.INI
41	0.00001280	Explorer	FindClose	C:\WIN\WIN.INI	OK	
42	0.00036960	Explorer	Op	C:\WIN\WIN.INI	OK	OPEXISTING READWRITE DENYWRITE
43	0.00004160	Explorer	Ioctl	C:	OK	Subfunction: 08h
44	0.00001360	Explorer	At	C:\WIN\WIN.INI	OK	Get Modify
45	0.00000880	Explorer	Sk	C:\WIN\WIN.INI	OK	End Off.: 0 / New Off.: 9942
46	0.00001040	Explorer	Sk	C:\WIN\WIN.INI	OK	Beg. Off.: 0 / New Off.: 0
47	0.00020320	Explorer	Rd	C:\WIN\WIN.INI	OK	Off.: 0 Len.: 9942
48	0.00004320	Explorer	Cl	C:\WIN\WIN.INI	OK	CL_FINAL
49	0.00022400	???	At	C:\WIN	OK	GetAttr
50	0.00005360	???	At	C:\WIN\RUNDLL32.EXE	OK	GetAttr
51	0.00027200	???	Directory	C:\WIN\RUNDLL32.EXE	OK	QUERY
52	0.00031280	???	Op	C:\WIN\RUNDLL32.EXE	OK	OPEXISTING READONLY DENYWRITE
53	0.00002160	???	Rd	C:\WIN\RUNDLL32.EXE	OK	Off.: 0 Len.: 64
54	0.00000960	???	Sk	C:\WIN\RUNDLL32.EXE	OK	Beg. Off.: 128 / New Off.: 128
. . . . .						
57	0.00001840	???	Rd	C:\WIN\RUNDLL32.EXE	OK	Off.: 128 Len.: 448
58	0.00026160	???	Directory	C:\WIN\RUNDLL32.EXE	OK	QUERY
59	0.00005280	???	Rd	C:\WIN\RUNDLL32.EXE	OK	Off.: 12288 Len.: 1024
60	0.00029280	???	Rd	C:\WIN\SYSTEM\SHELL32.DLL	OK	Off.: 1028096 Len.: 4096
61	0.00013440	???	Rd	C:\WIN\SYSTEM\SHELL32.DLL	OK	Off.: 1032192 Len.: 4096
. . . . .						
66	0.00005360	???	Rd	C:\WIN\SYSTEM\MSVCRT.DLL	OK	Off.: 241664 Len.: 1536
67	0.00003920	???	At	C:\WIN\SHLWAPI.DLL	NOTFOUND	GetAttr
68	0.00006640	???	At	C:\WIN\SYSTEM\SHLWAPI.DLL	OK	GetAttr
69	0.00072880	???	Directory	C:\WIN\SYSTEM\SHLWAPI.DLL	OK	QUERY
70	0.00027600	???	Rd	C:\WIN\SYSTEM\SHLWAPI.DLL	OK	Off.: 271360 Len.: 4096
71	0.00009600	???	Rd	C:\WIN\SYSTEM\SHLWAPI.DLL	OK	Off.: 271360 Len.: 4096
72	0.00014320	???	Rd	C:\WIN\SYSTEM\SHLWAPI.DLL	OK	Off.: 1024 Len.: 4096

```
73  0.00007760  ???       Rd  C:\WIN\SYSTEM\SHLWAPI.DLL  OK  Off.: 1024 Len.: 4096
. . . . . .
76  0.00011840  ???       Rd  C:\WIN\SYSTEM\SHELL32.DLL  OK  Off.: 1028096 Len.: 4096
77  0.00012720  ???       Rd  C:\WIN\SYSTEM\SHELL32.DLL  OK  Off.: 4096 Len.: 4096
. . . . . .
80  0.00012720  ???       Rd  C:\WIN\SYSTEM\MSVCRT.DLL  OK  Off.: 221184 Len.: 4096
81  0.00013280  ???       Rd  C:\WIN\SYSTEM\MSVCRT.DLL  OK  Off.: 225280 Len.: 4096
. . . . . .
84  0.00040560  ???       Rd  C:\WIN\SYSTEM\SHELL32.DLL  OK  Off.: 1044480 Len.: 3584
85  0.00015840  ???       Rd  C:\WIN\SYSTEM\SHLWAPI.DLL  OK  Off.: 259072 Len.: 4096
. . . . . .
107 0.00003200  Rundll32   Rd  C:\WIN\RUNDLL32.EXE  OK  Off.: 8192 Len.: 512
108 0.01226880  Rundll32   Rd  C:\WIN\SYSTEM\RNAUI.DLL  OK  Off.: 49152 Len.: 4096
109 0.00009680  Rundll32   Rd  C:\WIN\SYSTEM\RNAUI.DLL  OK  Off.: 53248 Len.: 3072
. . . . . .
115 0.00008480  Rundll32   Rd  C:\WIN\SYSTEM\COMDLG32.DLL  OK  Off.: 4096 Len.: 4096
116 0.00004160  Rundll32   At  C:\WIN\NETAPI32.DLL  NOTFOUND  GetAttr
117 0.00007840  Rundll32   At  C:\WIN\SYSTEM\NETAPI32.DLL  OK  GetAttr
118 0.00204960  Rundll32   Directory  C:\WIN\SYSTEM\NETAPI32.DLL  OK  QUERY
119 0.00026960  Rundll32   Rd  C:\WIN\SYSTEM\TAPI32.DLL  OK  Off.: 53248 Len.: 4096
120 0.00013920  Rundll32   Rd  C:\WIN\SYSTEM\TAPI32.DLL  OK  Off.: 57344 Len.: 4096
. . . . . .
124 0.00004160  Rundll32   At  C:\WIN\SECUR32.DLL  NOTFOUND  GetAttr
125 0.00005840  Rundll32   At  C:\WIN\SYSTEM\SECUR32.DLL  OK  GetAttr
126 0.00183120  Rundll32   Directory  C:\WIN\SYSTEM\SECUR32.DLL  OK  QUERY
127 0.00018880  Rundll32   Rd  C:\WIN\SYSTEM\SECUR32.DLL  OK  Off.: 49152 Len.: 2560
128 0.00012880  Rundll32   Rd  C:\WIN\SYSTEM\SECUR32.DLL  OK  Off.: 49152 Len.: 4096
. . . . . .
149 0.00017840  Rundll32   Rd  C:\WIN\SYSTEM\WSOCK32.DLL  OK  Off.: 24576 Len.: 2560
150 0.00004240  Rundll32   At  C:\WIN\WS2_32.DLL  NOTFOUND  GetAttr
151 0.00006320  Rundll32   At  C:\WIN\SYSTEM\WS2_32.DLL  OK  GetAttr
152 0.00187040  Rundll32   Directory  C:\WIN\SYSTEM\WS2_32.DLL  OK  QUERY
. . . . . .
169 0.00323600  Rundll32   Rd  C:\WIN\SYSTEM\RNAUI.DLL  OK  Off.: 61440 Len.: 512
170 0.00152400  Rundll32   Rd  C:\WIN\SYSTEM\RNAUI.DLL  OK  Off.: 12288 Len.: 4096
171 0.00028560  Rundll32   At  C:\WIN  OK  GetAttr
172 0.00002560  Rundll32   At  C:\WIN\RNAAPP.EXE  NOTFOUND  GetAttr
173 0.00005440  Rundll32   At  C:\WIN\SYSTEM\RNAAPP.EXE  OK  GetAttr
174 0.00036960  Rundll32   Op  C:\WIN\SYSTEM\RNAAPP.EXE  OK  OPEXISTING READONLY DENYWRITE
175 0.00933520  Rundll32   Rd  C:\WIN\SYSTEM\RNAAPP.EXE  OK  Off.: 0 Len.: 64
176 0.00001200  Rundll32   Sk  C:\WIN\SYSTEM\RNAAPP.EXE  OK  Beg. Off.: 128 / New Off.: 128
. . . . . .
190 0.00066720  ???       Rd  C:\WIN\SYSTEM\RNAAPP.EXE  OK  Off.: 32768 Len.: 2560
191 0.00001440  ???       GetDiskInfo  C:  OK  Free Space
192 0.00395600  ???       Wr  C:\WIN\WIN386.SWP  OK  Off.: 58720256 Len.: 0
193 0.00017360  ???       Rd  C:\WIN\SYSTEM\SHELL32.DLL  OK  Off.: 1028096 Len.: 4096
194 0.00011040  ???       Rd  C:\WIN\SYSTEM\SHELL32.DLL  OK  Off.: 1032192 Len.: 4096
. . . . . .
375 0.00054800  Tapisrv    Op  C:\WIN\SYSTEM\MSNSSPC.DLL  OK  OPEXISTING READONLY DENYWRITE
376 0.00001520  Tapisrv    Sk  C:\WIN\SYSTEM\MSNSSPC.DLL  OK  Beg. Off.: 0 / New Off.: 0
377 0.00000720  Tapisrv    Sk  C:\WIN\SYSTEM\MSNSSPC.DLL  OK  End Off.: 0 / New Off.: 122880
378 0.00000720  Tapisrv    Sk  C:\WIN\SYSTEM\MSNSSPC.DLL  OK  Beg. Off.: 0 / New Off.: 0
379 0.00000720  Tapisrv    Sk  C:\WIN\SYSTEM\MSNSSPC.DLL  OK  Beg. Off.: 0 / New Off.: 0
380 0.00000720  Tapisrv    Sk  C:\WIN\SYSTEM\MSNSSPC.DLL  OK  End Off.: 0 / New Off.: 122880
381 0.00000720  Tapisrv    Sk  C:\WIN\SYSTEM\MSNSSPC.DLL  OK  Beg. Off.: 0 / New Off.: 0
382 0.00015440  Tapisrv    Rd  C:\WIN\SYSTEM\MSNSSPC.DLL  OK  Off.: 0 Len.: 0
383 0.00741680  Tapisrv    Rd  C:\WIN\SYSTEM\MSNSSPC.DLL  OK  Off.: 0 Len.: 4096
384 0.00436480  Tapisrv    Rd  C:\WIN\SYSTEM\MSNSSPC.DLL  OK  Off.: 32768 Len.: 4096
385 0.00006320  Tapisrv    Cl  C:\WIN\SYSTEM\MSNSSPC.DLL  OK  CL_FOR_PROCESS
386 0.00002800  Tapisrv    Cl  C:\WIN\SYSTEM\MSNSSPC.DLL  OK  CL_FINAL
387 0.00369440  Tapisrv    Rd  C:\WIN\SYSTEM\MSNSSPC.DLL  OK  Off.: 12288 Len.: 4096
388 0.00012800  Tapisrv    At  C:\WIN\SYSTEM\DIGEST.DLL  OK  GetAttr
389 0.00032080  Tapisrv    Directory  C:\WIN\SYSTEM\DIGEST.DLL  OK  QUERY
390 0.00711040  Tapisrv    Rd  C:\WIN\SYSTEM\DIGEST.DLL  OK  Off.: 26112 Len.: 4096
391 0.00008400  Tapisrv    Rd  C:\WIN\SYSTEM\DIGEST.DLL  OK  Off.: 30208 Len.: 1536
392 0.00026240  Tapisrv    Rd  C:\WIN\SYSTEM\DIGEST.DLL  OK  Off.: 26112 Len.: 4096
393 0.00782880  Tapisrv    Rd  C:\WIN\SYSTEM\DIGEST.DLL  OK  Off.: 1536 Len.: 4096
394 0.00012560  Tapisrv    Rd  C:\WIN\SYSTEM\DIGEST.DLL  OK  Off.: 1536 Len.: 4096
395 0.00068320  Tapisrv    Rd  C:\WIN\SYSTEM\DIGEST.DLL  OK  Off.: 5632 Len.: 4096
396 0.01824720  Tapisrv    Rd  C:\WIN\SYSTEM\DIGEST.DLL  OK  Off.: 39936 Len.: 1024
. . . . . .
419 0.00244240  Tapisrv    Directory  C:\WIN\SYSTEM\WSOCK32.DLL  OK  QUERY
420 0.00011040  Tapisrv    At  C:\WIN\SYSTEM\CRYPT32.DLL  OK  GetAttr
421 0.00047520  Tapisrv    Directory  C:\WIN\SYSTEM\CRYPT32.DLL  OK  QUERY
422 0.00043760  Tapisrv    Rd  C:\WIN\SYSTEM\CRYPT32.DLL  OK  Off.: 405504 Len.: 4096
423 0.00025360  Tapisrv    Rd  C:\WIN\SYSTEM\CRYPT32.DLL  OK  Off.: 409600 Len.: 4096
424 0.00023280  Tapisrv    Rd  C:\WIN\SYSTEM\CRYPT32.DLL  OK  Off.: 413696 Len.: 4096
425 0.00043760  Tapisrv    Rd  C:\WIN\SYSTEM\CRYPT32.DLL  OK  Off.: 405504 Len.: 4096
426 0.00025040  Tapisrv    Rd  C:\WIN\SYSTEM\CRYPT32.DLL  OK  Off.: 4096 Len.: 4096
427 0.00009760  Tapisrv    Rd  C:\WIN\SYSTEM\CRYPT32.DLL  OK  Off.: 4096 Len.: 4096
428 0.00025200  Tapisrv    Rd  C:\WIN\SYSTEM\SCHANNEL.DLL  OK  Off.: 99840 Len.: 4096
429 0.00136240  Tapisrv    Rd  C:\WIN\SYSTEM\SCHANNEL.DLL  OK  Off.: 1536 Len.: 4096
430 0.00014720  Tapisrv    Rd  C:\WIN\SYSTEM\SCHANNEL.DLL  OK  Off.: 1536 Len.: 4096
431 0.02181360  Tapisrv    Rd  C:\WIN\SYSTEM\MSASN1.DLL  OK  Off.: 45056 Len.: 4096
432 0.00043200  Tapisrv    Rd  C:\WIN\SYSTEM\MSASN1.DLL  OK  Off.: 49152 Len.: 1536
433 0.00028240  Tapisrv    Rd  C:\WIN\SYSTEM\CRYPT32.DLL  OK  Off.: 417792 Len.: 4096
. . . . . .
571 0.00001840  Tapisrv    GetDiskInfo  C:  OK  Free Space
572 0.00000640  Tapisrv    GetDiskInfo  C:  OK  Free Space
573 0.00804480  Rnaapp     Rd  C:\WIN\SYSTEM\TAPI32.DLL  OK  Off.: 8192 Len.: 4096
574 0.00829120  Tapisrv    Rd  C:\WIN\SYSTEM\TAPISRV.EXE  OK  Off.: 53248 Len.: 4096
575 0.00108800  Tapisrv    Rd  C:\WIN\SYSTEM\TAPISRV.EXE  OK  Off.: 20480 Len.: 4096
576 0.00011760  Tapisrv    At  C:\WIN\SYSTEM\TSP3216L.TSP  OK  GetAttr
577 0.00098640  Tapisrv    Directory  C:\WIN\SYSTEM\TSP3216L.TSP  OK  QUERY
. . . . . .
610 0.00046720  Tapisrv    Op  C:\WIN\TELEPHON.INI  OK  OPEXISTING READWRITE DENYWRITE
```

```
611  0.00003280  Tapisrv    Ioctl  C:                                  OK      Subfunction: 08h
612  0.00001520  Tapisrv    At     C:\WIN\TELEPHON.INI                 OK      Get Modify
613  0.00000880  Tapisrv    Sk     C:\WIN\TELEPHON.INI                 OK      End Off.: 0 / New Off.: 428
614  0.00000720  Tapisrv    Sk     C:\WIN\TELEPHON.INI                 OK      Beg. Off.: 0 / New Off.: 0
615  0.00557360  Tapisrv    Rd     C:\WIN\TELEPHON.INI                 OK      Off.: 0 Len.: 428
616  0.00005200  Tapisrv    Cl     C:\WIN\TELEPHON.INI                 OK      CL_FINAL
617  0.00008320  Tapisrv    At     C:\WIN\SYSTEM\UNIMDM.TSP            OK      GetAttr
618  0.00034800  Tapisrv    Op     C:\WIN\SYSTEM\UNIMDM.TSP    OK      OPEXISTING READONLY COMPATIBILITY
619  0.00002320  Tapisrv    Ioctl  C:                                  OK      Subfunction: 08h
620  0.00001040  Tapisrv    At     C:\WIN\SYSTEM\UNIMDM.TSP            OK      Get Modify
621  0.00541040  Tapisrv    Rd     C:\WIN\SYSTEM\UNIMDM.TSP            OK      Off.: 0 Len.: 64
622  0.00001200  Tapisrv    Sk     C:\WIN\SYSTEM\UNIMDM.TSP            OK      Beg. Off.: 128 / New Off.: 128
623  0.00001440  Tapisrv    Rd     C:\WIN\SYSTEM\UNIMDM.TSP            OK      Off.: 128 Len.: 64
624  0.00002480  Tapisrv    Rd     C:\WIN\SYSTEM\UNIMDM.TSP            OK      Off.: 192 Len.: 1255
625  0.00000880  Tapisrv    Sk     C:\WIN\SYSTEM\UNIMDM.TSP            OK      Beg. Off.: 2848 / New Off.: 2848
626  0.00542880  Tapisrv    Rd     C:\WIN\SYSTEM\UNIMDM.TSP            OK      Off.: 2848 Len.: 32768
627  0.00085520  Tapisrv    Rd     C:\WIN\SYSTEM\UNIMDM.TSP            OK      Off.: 35616 Len.: 8960
628  0.00008320  Tapisrv    At     C:\WIN\SYSTEM\WAN.TSP               OK      GetAttr
. . . . . .
696  0.00022160  Mprexe     At     C:\WIN\SYSTEM.INI                   OK      GetAttr
697  0.00072240  Mprexe     Op     C:\WIN\SYSTEM.INI                   OK      OPEXISTING READWRITE DENYWRITE
698  0.00006720  Mprexe     Ioctl  C:                                  OK      Subfunction: 08h
699  0.00003440  Mprexe     At     C:\WIN\SYSTEM.INI                   OK      Get Modify
700  0.00001120  Mprexe     Sk     C:\WIN\SYSTEM.INI                   OK      End Off.: 0 / New Off.: 2239
701  0.00001120  Mprexe     Sk     C:\WIN\SYSTEM.INI                   OK      Beg. Off.: 0 / New Off.: 0
702  0.00008400  Mprexe     Rd     C:\WIN\SYSTEM.INI                   OK      Off.: 0 Len.: 2239
703  0.00007840  Mprexe     Cl     C:\WIN\SYSTEM.INI                   OK      CL_FINAL
704  0.00090320  Mprexe     Op     C:\WIN\MATTOS.PWL           OK      OPEXISTING READWRITE DENYREADWRITE
705  0.00001520  Mprexe     Sk     C:\WIN\MATTOS.PWL                   OK      Beg. Off.: 0 / New Off.: 0
706  0.01793840  Mprexe     Rd     C:\WIN\MATTOS.PWL                   OK      Off.: 0 Len.: 594
707  0.00001920  Mprexe     Sk     C:\WIN\MATTOS.PWL                   OK      Beg. Off.: 594 / New Off.: 594
. . . . . .
730  0.00005440  Mprexe     Cl     C:\WIN\MATTOS.PWL                   OK      CL_FINAL
731  0.00019600  Rundll32   Rd     C:\WIN\SYSTEM\TAPI32.DLL            OK      Off.: 45056 Len.: 4096
732  0.00013760  Rundll32   Rd     C:\WIN\SYSTEM\TAPI32.DLL            OK      Off.: 36864 Len.: 4096
733  0.00012720  Rundll32   Rd     C:\WIN\SYSTEM\TAPI32.DLL            OK      Off.: 40960 Len.: 4096
. . . . . .
755  0.00010160  Explorer   FindOp      C:\WIN\WIN.INI                 OK      WIN.INI
756  0.00001280  Explorer   FindClose   C:\WIN\WIN.INI                 OK
757  0.00038480  Explorer   Op     C:\WIN\WIN.INI              OK      OPEXISTING READWRITE DENYWRITE
758  0.00003760  Explorer   Ioctl  C:                                  OK      Subfunction: 08h
759  0.00001280  Explorer   At     C:\WIN\WIN.INI                      OK      Get Modify
760  0.00000880  Explorer   Sk     C:\WIN\WIN.INI                      OK      End Off.: 0 / New Off.: 9942
761  0.00001120  Explorer   Sk     C:\WIN\WIN.INI                      OK      Beg. Off.: 0 / New Off.: 0
762  0.00017840  Explorer   Rd     C:\WIN\WIN.INI                      OK      Off.: 0 Len.: 9942
763  0.00004160  Explorer   Cl     C:\WIN\WIN.INI                      OK      CL_FINAL
764  0.00014240  Explorer   At     C:\MEUS DOCUMENTOS\SYSINTERNALS1.LOG NOTFOUND       GetAttr
765  0.00006880  Explorer   Ioctl  C:                                  OK      Subfunction: 08h
766  0.00013840  Explorer   FindOp      C:\*.*                         OK      DISCO51-.C
767  0.00002000  Explorer   FindClose   C:\*.*                         OK
. . . . . .
801  0.00174720  Explorer   At     C:\WIN\RECENT\SYSINTERNALS1.LOG.LNK  NOTFOUND  GetAttr
802  0.09264080  Explorer   At     C:\WIN\RECENT\SYSINTERNALS1.LOG.LNK  NOTFOUND  GetAttr
803  0.00164960  Explorer   At     C:\WIN\RECENT\SYSINTERNALS1.LOG.LNK NOTFOUND GET_ATTRIB_CREATION_DATETIME
804  0.00663920  Explorer   Op     C:\WIN\RECENT\SYSINTERNALS1.LOG.LNK NOTFOUND OPEXISTING READONLY DENYNONE
805  0.03187920  Explorer   Wr     C:\_RESTORE\LOGS\VXDALT1.LOG        OK      Off.: 0 Len.: 4096
806  0.04253840  Explorer   Op     C:\WIN\RECENT\SYSINTERNALS1.LOG.LNK OK       CREATENEW REPLACEEXISTING
READWRITE DENYNONE
807  0.00049280  Explorer   Wr     C:\WIN\RECENT\SYSINTERNALS1.LOG.LNK  OK      Off.: 0 Len.: 373
808  0.00008000  Explorer   Cl     C:\WIN\RECENT\SYSINTERNALS1.LOG.LNK  OK      CL_FINAL
. . . . . .
815  0.00028480  Explorer   Sk     C:\WIN\TEMPOR~1\CONTENT.IE5\INDEX.DAT OK Beg. Off.: 0 / New Off.: 0
816  0.00000880  Explorer   Sk     C:\WIN\TEMPOR~1\CONTENT.IE5\INDEX.DAT OK End Off.: 0 / New Off.: 2310144
817  0.00000720  Explorer   Sk     C:\WIN\TEMPOR~1\CONTENT.IE5\INDEX.DAT OK  Beg. Off.: 0 / New Off.: 0
818  0.00011200  Explorer   Sk     C:\WIN\HISTËR~1\HISTORY.IE5\INDEX.DAT    OK  Beg. Off.: 0 / New Off.: 0
819  0.00000880  Explorer   Sk     C:\WIN\HISTËR~1\HISTORY.IE5\INDEX.DAT    OK  End Off.: 0 / New Off.: 524288
. . . . . .
842  0.00071840  Explorer   Op C:\WIN\HISTËRICO\DESKTOP.INI    OK  OPEXISTING READWRITE DENYWRITE
843  0.00003600  Explorer   Ioctl      C:                                  OK  Subfunction: 08h
844  0.00002720  Explorer   At         C:\WIN\HISTËRICO\DESKTOP.INI        OK  Get Modify
845  0.00000880  Explorer   Sk     C:\WIN\HISTËRICO\DESKTOP.INI     OK  End Off.: 0 / New Off.: 113
846  0.00000880  Explorer   Sk         C:\WIN\HISTËRICO\DESKTOP.INI    OK  Beg. Off.: 0 / New Off.: 0
847  0.00700080  Explorer   Rd         C:\WIN\HISTËRICO\DESKTOP.INI        OK  Off.: 0 Len.: 113
848  0.00006080  Explorer   Cl         C:\WIN\HISTËRICO\DESKTOP.INI        OK  CL_FINAL
849  0.00013200  Explorer   At         C:\WIN\HISTËRICO\DESKTOP.INI        OK  GetAttr
850  0.01461040  Explorer   Rd         C:\WIN\SYSTEM\SHDOCVW.DLL       OK  Off.: 607744 Len.: 4096
. . . . . .
911  0.00012320  MSGSRV32   At     C:\WIN\USER.DAT          OK  GetAttr
912  0.00062400  MSGSRV32   At     C:\WIN\USER.DAT          OK  SetAttr
913  0.00030800  MSGSRV32   Op     C:\WIN\USER.DAT    OK  OPEXISTING WRITEONLY COMPATIBILITY
914  0.00001120  MSGSRV32   Sk     C:\WIN\USER.DAT          OK  Beg. Off.: 32 / New Off.: 32
915  0.00004240  MSGSRV32   Wr     C:\WIN\USER.DAT          OK  Off.: 32 Len.: 32
. . . . . .
926  0.00006480  MSGSRV32   Cl     C:\WIN\USER.DAT                  OK  CL_FINAL
```

10.5 Apêndice 5 – Exemplo de log obtido através de Sar (Unix)

AIX penha 3 4 000000775700 12/20/01

15:42:31	iget/s	lookuppn/s	dirblk/s
15:42:36	0	3	20
15:42:41	0	1	0
15:42:46	0	0	0
. . . . .			
15:44:36	1	34	49
15:44:41	0	2	5
15:44:46	1	72	133
15:44:51	0	2	6
15:44:56	0	21	24
15:45:01	1	66	69
15:45:06	0	30	24
15:45:11	0	12	18
15:45:16	0	0	6
15:45:21	0	8	14
15:45:26	1	34	42
15:45:31	1	50	61
. . . . .			
15:46:41	1	798	2645
15:46:46	0	925	1355
15:46:51	0	915	663
15:46:56	0	913	635
15:47:01	0	1114	1688
15:47:06	0	1225	9918
15:47:11	0	1340	6633
15:47:16	0	1210	1785
15:47:21	0	1125	1423
. . . . .			
15:57:26	1	33	43
15:57:31	0	10	0
Average	0	340	188

15:42:31	bread/s	lread/s	%rcache	bwrit/s	lwrit/s	%wcach	pread/s	pwrit/s
15:42:36	0	0	0	0	0	0	0	0
15:42:41	0	0	0	0	0	0	0	0
15:42:46	0	0	0	0	0	0	0	0
.....								
15:57:31	0	0	0	0	0	0	0	0
Average	0	0	0	0	0	0	0	0

15:42:31	scall/s	sread/s	swrit/s	fork/s	exec/s	rchar/s	wchar/s
15:42:36	106	14	10	0.00	0.00	3549	1
15:42:41	47	9	7	0.00	0.00	1024	3
15:42:46	55	10	7	0.20	0.00	5157	1
15:42:51	79	18	6	0.00	0.00	4457	1
15:42:56	644	102	25	1.40	1.20	25756	3
15:43:01	117	10	7	0.20	0.20	3457	1
15:43:06	114	12	9	0.20	0.20	3984	1
. . . . .							
15:44:31	126	16	9	0.20	0.20	4242	2
15:44:36	358	42	9	0.40	0.20	8142	6
15:44:41	51	9	6	0.00	0.00	1574	1
15:44:46	1458	166	52	0.80	0.80	55623	7
15:44:51	76	13	9	0.00	0.00	7044	2
15:44:56	408	69	21	1.20	0.40	15246	10
15:45:01	665	89	12	1.00	0.80	21398	1
15:45:06	334	49	8	0.20	0.20	15148	6
.5:48:11	3194	393	548	0.40	0.20	25379	2
. . . . .							
15:53:01	3334	403	580	0.40	0.20	17823	1
15:53:06	3622	468	575	1.20	1.00	16404	8
15:53:11	3396	430	567	0.60	0.00	18548	1
15:53:16	3439	421	589	0.00	0.00	16647	4
15:53:21	3992	638	957	0.20	0.20	19128	1
15:53:26	4044	659	939	0.20	0.00	15998	5
15:53:31	4044	660	922	0.00	0.00	16391	0
15:53:36	4135	695	930	1.00	0.40	21760	9
. . . . .							
15:57:26	330	29	9	0.40	0.20	3658	2
15:57:31	135	11	6	0.40	0.40	6254	2
Average	999	125	172	0.15	0.08		

15:42:31	ksched/s	kproc-ov	kexit/s
15:42:36	0	0	0
15:42:41	0	0	0
15:42:46	0	0	0
15:42:51	0	0	0
. . . . .			
15:45:31	0	0	0
15:45:36	0	0	0
15:49:16	0	0	0
15:54:16	0	0	0
. . . . .			
15:57:16	0	0	0
15:57:21	0	0	0
15:57:26	0	0	0
15:57:31	0	0	0
Average	0	0	0

15:42:31	iwch/s	msg/s	sema/s
15:42:36	0	0.00	0.00
15:42:41	0	0.00	0.00
15:42:46	0	0.00	0.00
15:42:51	0	0.00	0.00
15:42:56	0	0.00	0.00
15:43:01	0	0.00	0.00
15:43:06	0	0.00	0.00
. . . . .			
15:48:26	0	0.00	0.00
15:48:31	0	0.00	0.00
15:48:36	0	0.00	0.00
15:48:41	0	0.00	0.00
15:48:46	0	0.00	0.00
15:48:51	0	0.00	0.00
15:48:56	0	1.20	0.00
15:49:01	0	0.00	0.00
15:49:06	0	1.20	0.00
. . . . .			
15:54:31	0	0.00	0.00
15:54:36	0	0.00	4.80
15:54:41	0	0.00	7.20
15:54:46	0	0.00	5.99
15:54:51	0	0.00	0.00
15:54:56	0	0.00	0.00
. . . . .			
15:56:16	0	0.00	0.00
15:56:21	0	0.00	0.00
15:56:26	0	0.00	13.20
. . . . .			
15:57:26	0	0.00	0.00
15:57:31	0	0.40	0.00
Average	0	0.00	0.10

15:42:31	runq-sz	%runocc	swpq-sz	%swpocc
15:42:36	1.0	20		
15:42:41				
15:42:46				
15:42:51				
15:42:56	2.0	40		
. . . . .				
15:43:46				
15:43:51	1.0	20		
15:43:56			1.0	20
15:44:01	1.0	20		
15:44:06	1.0	40		
15:44:11				
15:44:16	1.0	20		
15:44:21	1.0	20		
15:44:26	2.0	20		
15:44:31				
15:44:36	1.0	20		
15:44:41				
15:44:46	1.3	60		
15:44:51				
. . . . .				
15:46:46	1.5	80	1.0	60
15:46:51	1.8	80	1.0	40
15:46:56	1.5	80	1.0	60

15:47:01	1.4	100	1.0	20
15:47:06	1.6	100	1.0	40
15:47:11	1.2	80	1.0	80
15:47:16	1.4	100	1.0	60
15:47:21	1.6	100	1.0	40
15:47:26	1.2	100	1.0	60
Average	1.8	10	1.0	2

15:42:31	slots	cycle/s	fault/s	odio/s
15:42:36	94563	0.00	0.80	1.80
15:42:41	94563	0.00	0.60	0.20
15:42:46	94563	0.00	18.80	0.00
15:42:51	94563	0.00	7.60	12.20
15:42:56	94563	0.00	185.63	5.99
15:43:01	94563	0.00	26.40	0.60
15:43:06	94563	0.00	19.60	0.00
15:43:11	94563	0.00	6.20	0.00
15:43:16	94563	0.00	0.00	0.00
15:43:21	94563	0.00	94.60	32.40
15:43:26	94563	0.00	10.40	0.80
15:43:31	94563	0.00	0.00	0.80
15:43:36	94563	0.00	10.80	0.60
15:43:41	94562	0.00	56.00	0.00
15:43:46	94560	0.00	34.40	0.20
15:43:51	94517	0.00	82.04	8.18
15:43:56	94517	0.00	1.60	20.00
15:44:01	94517	0.00	37.60	2.00
15:44:06	94517	0.00	211.00	14.20
15:44:11	94517	0.00	96.40	0.60
15:44:16	94517	0.00	128.20	4.60
15:57:26	93457	0.00	84.00	6.00
15:57:31	93457	0.00	34.80	0.00
Average	93989	0.00	9	2

15:42:31	%usr	%sys	%wio	%idle
15:42:36	1	1	1	97
15:42:41	2	0	0	98
15:42:46	9	0	0	90
15:42:51	2	1	8	89
15:42:56	7	8	1	84
15:43:01	1	1	0	97
15:43:06	1	1	0	98
15:43:11	3	1	0	97
15:49:46	24	53	0	23
15:49:51	21	53	0	26
15:49:56	19	59	2	20
15:50:01	22	55	2	21
15:50:06	22	54	0	24
15:50:11	26	49	0	24
15:50:16	29	54	0	17
15:57:11	17	6	8	69
15:57:16	2	2	4	92
15:57:21	3	1	0	96
15:57:26	4	3	5	88
15:57:31	2	0	0	98
Average	16	30	3	51

15:42:31	proc-sz	inod-sz	file-sz	thrd-sz
15:42:36	80/131072	731/3819	393/938	129/262144
15:42:41	80/131072	731/3819	393/938	129/262144
15:42:46	81/131072	731/3819	396/938	130/262144
15:42:51	81/131072	733/3819	397/938	130/262144
15:42:56	86/131072	732/3819	406/938	135/262144
15:43:01	86/131072	732/3819	407/938	135/262144
15:43:06	86/131072	732/3819	407/938	135/262144
15:57:21	92/131072	737/3819	457/938	145/262144
15:57:26	91/131072	737/3819	452/938	144/262144
15:57:31	93/131072	738/3819	458/938	146/262144

15:42:31	cswch/s
15:42:36	49
15:42:41	41
15:42:46	38
15:42:51	45
15:42:56	75
15:43:01	41
15:46:51	666
15:46:56	347
15:47:01	466
15:47:06	228
15:47:11	227
15:47:16	291
15:47:21	231
15:47:26	227
15:47:31	348
15:47:36	569
15:47:41	1052
15:47:46	838
15:54:31	692
15:54:36	274
15:54:41	731
15:54:46	1272
15:54:51	1030
15:54:56	484
15:55:01	912
15:55:06	1076
15:55:11	866
15:55:16	969
15:55:21	618
15:55:26	37
15:55:31	48
15:55:36	42
15:55:41	46
15:55:46	51
15:55:51	48
15:55:56	59
15:57:31	43
Average	50

15:42:31	rawch/s	canch/s	outch/s	rcvin/s	xmtin/s	mdmin/s
15:42:36	0	0	109	0	0	0
15:42:41	0	0	101	0	0	0
15:42:46	0	0	101	0	0	0
15:42:51	0	0	101	0	0	0
15:42:56	2	0	412	0	0	0
15:43:01	0	0	101	0	0	0
15:43:06	1	0	110	0	0	0
15:43:11	0	0	141	0	0	0
15:46:36	6	0	118	0	0	0
15:46:41	1	0	14086	0	0	0
15:46:46	0	0	20923	0	0	0
15:46:51	0	0	16942	0	0	0
15:46:56	0	0	20737	0	0	0
15:47:01	0	0	23494	0	0	0
15:47:06	0	0	41664	0	0	0
15:54:51	0	0	15170	0	0	0
15:54:56	0	0	7188	0	0	0
15:55:01	0	0	10626	0	0	0
15:55:06	0	0	15126	0	0	0
15:55:11	0	0	26832	0	0	0
15:55:16	0	0	15509	0	0	0
15:55:21	0	0	2843	0	0	0
15:55:26	0	0	101	0	0	0
15:55:31	0	0	101	0	0	0
15:55:36	0	0	101	0	0	0
15:55:41	0	0	100	0	0	0
15:55:46	0	0	101	0	0	0
15:55:51	0	0	101	0	0	0
15:55:56	1	0	169	0	0	0
15:56:01	0	0	101	0	0	0
15:56:06	0	0	187	0	0	0
15:56:11	0	0	101	0	0	0
15:56:16	0	0	101	0	0	0
15:56:21	0	0	101	0	0	0
15:56:26	0	0	201	0	0	0
15:56:31	0	0	101	0	0	0
15:56:36	0	0	100	0	0	0
15:56:41	0	0	880	0	0	0
15:56:46	0	0	101	0	0	0
15:56:51	0	0	101	0	0	0
15:56:56	0	0	229	0	0	0
15:57:01	0	0	133	0	0	0
15:57:06	0	0	101	0	0	0
15:57:11	0	0	190	0	0	0
15:57:16	0	0	183	0	0	0
15:57:21	0	0	101	0	0	0
15:57:26	0	0	181	0	0	0
15:57:31	0	0	101	0	0	0
Average	0	0	1138	0	0	0

10.6 Apêndice 6 – Modelo de árvore de decisões para algoritmos

A figura abaixo apresenta um extrato da última versão da árvore de decisões implementada no protótipo que, a partir das respostas dos alunos, constrói um algoritmo para a solução de um determinado problema.

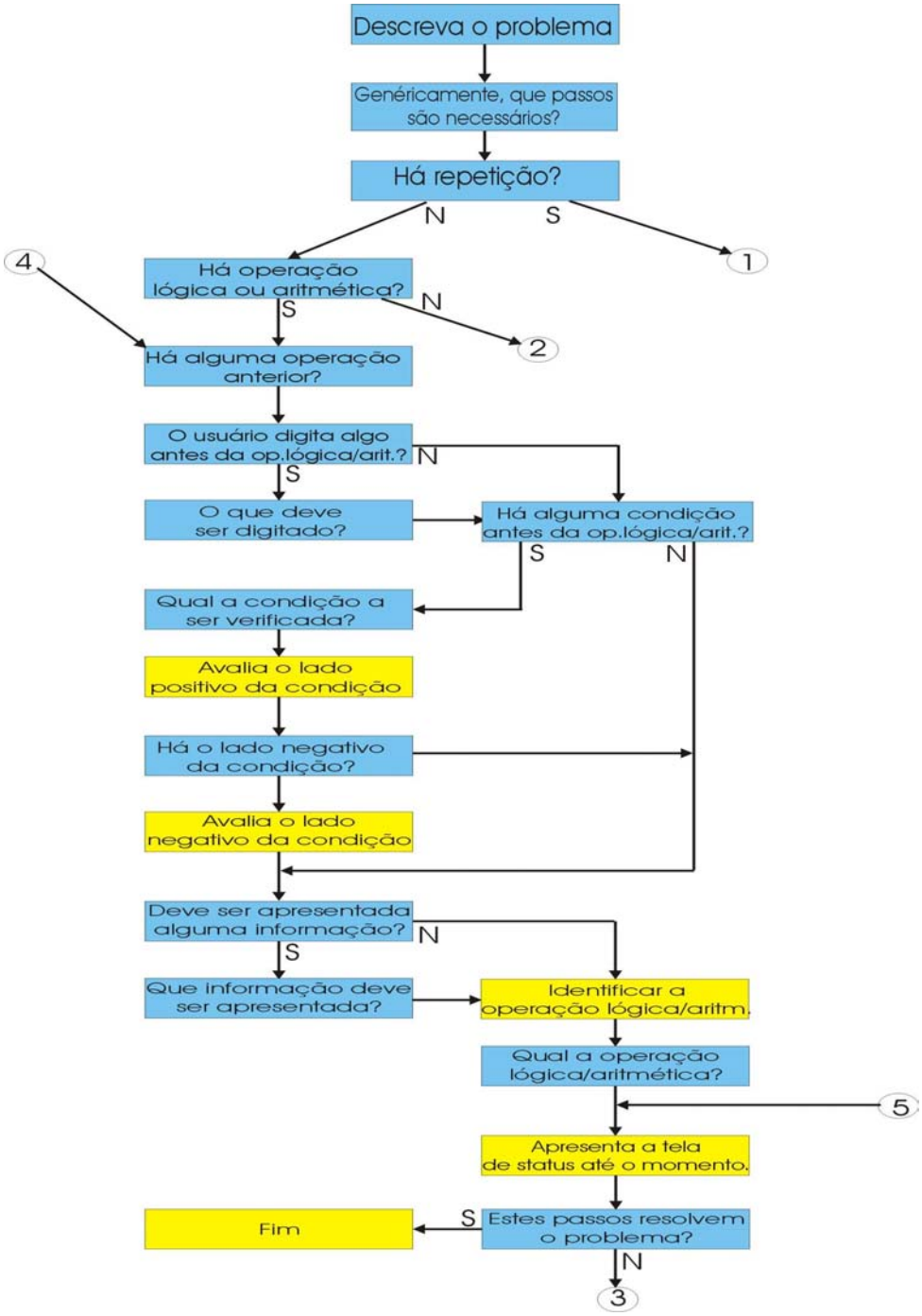


Figura 92 – Exemplo de árvore de decisões utilizada no protótipo de validação do modelo de aplicações

10.7 Apêndice 7 – Modelo de árvore de decisões para redações

A figura abaixo apresenta um extrato da última versão da árvore de decisões implementando o conhecimento do especialista na construção de redações. O resultado da execução desta estrutura é um meta-modelo conceitual descrevendo a estrutura do documento que está sendo redigido.

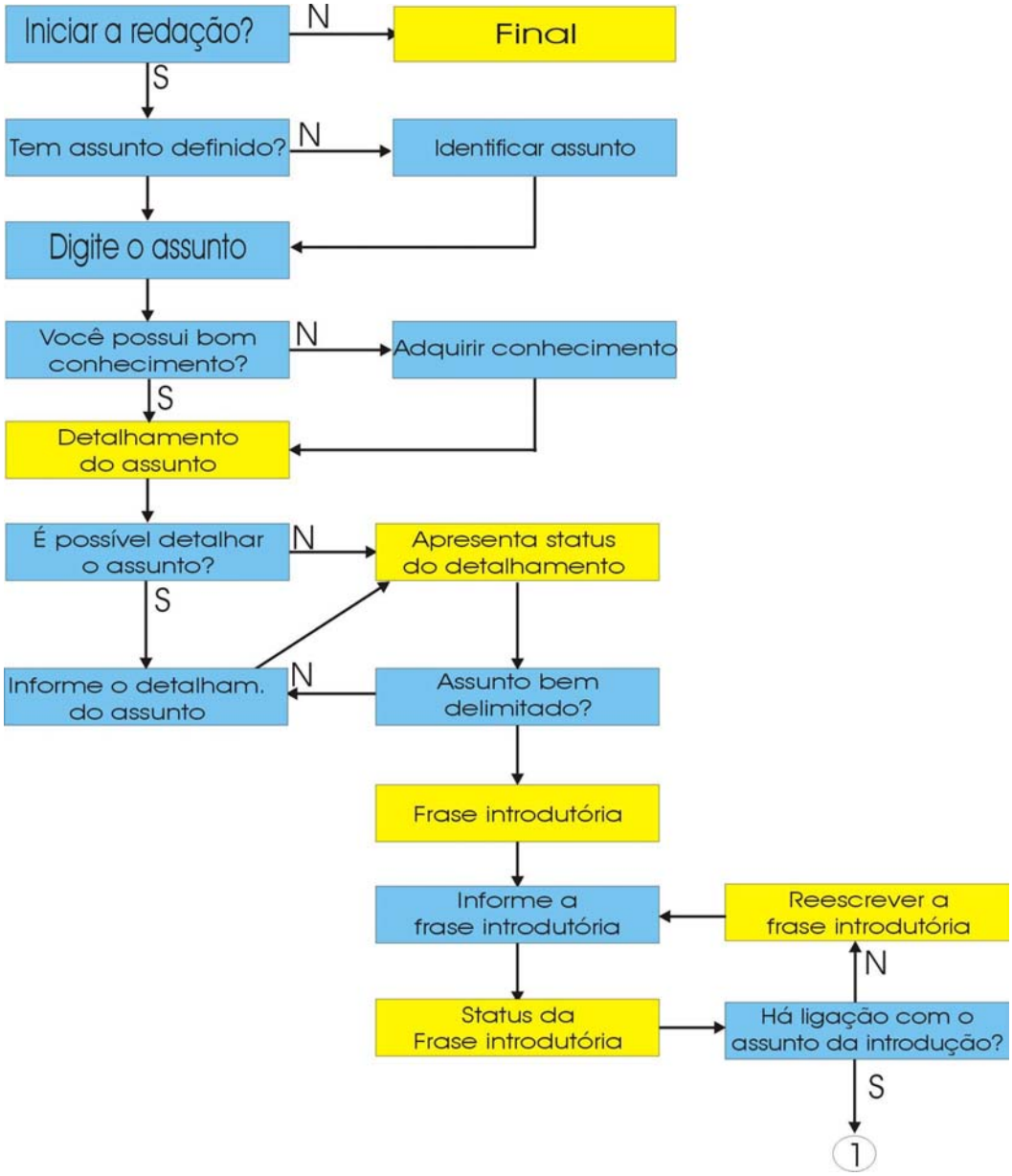
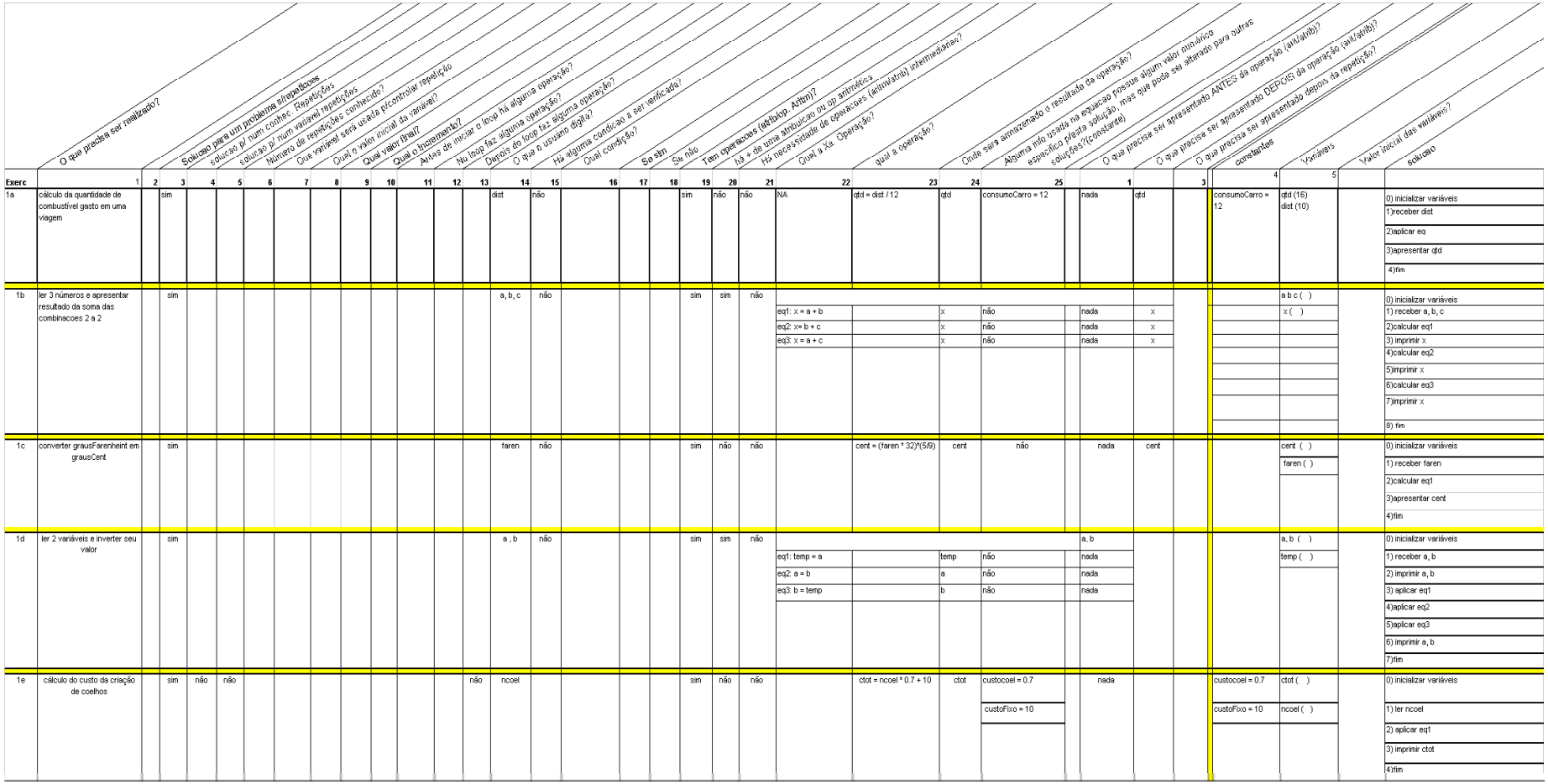


Figura 93 – Exemplo de árvore de decisões modelando conhecimento sobre construção de redações

10.8 Apêndice 8 – Planilha para identificação de regras



Exerc	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	1	3	4	5	solucao
	O que precisa ser realizado?	Solucao para um problema sRepetições	solucao p/ num conhec. Repetições	solucao p/ num variavel repetições	Numero de repetições conhecido?	Qua variavel será usada p/controlar repetição	Qual o valor inicial da variavel?	Qual valor final?	Qual o incremento?	Antes de iniciar o loop há alguma operação?	Nu loop faz alguma operação?	Depois do loop faz alguma operação?	O que o usuario digita?	Ha alguma condicao a ser verificada?	Qual condição?	Se sim	Se não	Tem operacoes (attb)op. Attn)?	Ha + de uma attribuicao ou op aritmetica	Ha necessidade de operacoes (arit/attrib) intermediarias?	Qual a Xa. Operação?	qual a operação?	Onde sera armazenado o resultado da operação?	Alguma info usada na equacao possue algum valor numérico específico p/esta solução, mas que pode ser alterado para outras soluções?(constante)	O que precisa ser apresentado ANTES da operação (ant/attrib)?	O que precisa ser apresentado DEPOIS da operação (ant/attrib)?	O que precisa ser apresentado depois da repetição?	constantes	Variáveis	Valor inicial das variaveis?	
1a	cálculo da quantidade de combustível gasto em uma viagem		sim											dist	não				sim	não	não	NA	qtd = dist / 12	qtd	consumoCarro = 12	nada	qtd		consumoCarro = 12	qtd (16) dist (10)		0) inicializar variáveis 1)receber dist 2)aplicar eq 3)apresentar qtd 4)fim
1b	ler 3 números e apresentar resultado da soma das combinacoes 2 a 2		sim											a, b, c	não				sim	sim	não	eq1: x = a + b eq2: x= b + c eq3: x = a + c		x x x	não não não	nada nada nada	x x x			a b c ( ) x ( )		0) inicializar variáveis 1) receber a, b, c 2)calcular eq1 3) imprimir x 4)calcular eq2 5)imprimir x 6)calcular eq3 7)imprimir x 8) fim
1c	converter grausFarenheint em grausCent		sim											faren	não				sim	não	não		cent = (faren * 32)*(5/9)	cent	não	nada	cent			cent ( ) faren ( )		0) inicializar variáveis 1) receber faren 2)calcular eq1 3)apresentar cent 4)fim
1d	ler 2 variáveis e inverter seu valor		sim											a , b	não				sim	sim	não	eq1: temp = a eq2: a = b eq3: b = temp		temp a b	não não não	nada nada nada	a, b			a, b ( ) temp ( )		0) inicializar variáveis 1) receber a, b 2) imprimir a, b 3) aplicar eq1 4)aplicar eq2 5)aplicar eq3 6) imprimir a, b 7)fim
1e	cálculo do custo da criação de coelhos		sim	não	não								não	ncoel					sim	não	não		ctot = ncoel * 0.7 + 10	ctot	custocoel = 0.7 custoFixo = 10	nada			custocoel = 0.7 custoFixo = 10	ctot ( ) ncoel ( )		0) inicializar variáveis 1) ler ncoel 2) aplicar eq1 3) imprimir ctot 4)fim

Figura 94 – Planilha de regras utilizadas para montagem da árvore de decisões