UNIVERSIDADE FEDERAL DE SANTA CATARINA
CURSO DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Helcio Hermes Hoffmann

UM AMBIENTE COMPUTACIONAL UBÍQUO ORIENTADO A MOBILIDADE

Dissertação submetida à Universidade Federal de Santa Catarina
como parte dos requisitos para obtenção de grau de
Mestre em Ciência da Computação

Prof. Rosvelter João Coelho da Costa
Orientador

Florianópolis, 26 de Fevereiro de 2003

UM AMBIENTE COMPUTACIONAL UBÍQUO ORIENTADO A MOBILIDADE

Helcio Hermes Hoffmann

Esta dissertação foi julgada adequada para a obtenção do título de Mestre em Ciências da computação Área de concentração Sistemas Distribuídos e aprovada em sua forma final pelo programa de pós-graduação em Ciência da Computação

_____

Prof. Fernando A. Ostini Gauthier, Dr

Coordenador do Curso

Banca Examinadora

_____

Prof. Rosvelter João Coelho da Costa, Dr.

Presidente da Banca (Orientador)

_____

Prof. João Bosco Mangueira Sobral, Dr.

Membro da Banca

_____

Prof.Mário Antonio Ribeiro Dantas, Dr.

Membro da Banca

DEDICATÓRIA

Aos meus pais, Valtraud e Venidor, aos sogros Hilda e Orlando, aos meus irmãos Herci, Herson e Hansi e suas esposas, e em especial a minha esposa Dolores pela dedicação, paciência e apoio por ela despendida sem medir esforços.

Aos meus professores, peças chave, em qualquer transformação, aos meus alunos e as equipes de professores das instituições aonde leciono, aos colaboradores da empresa VISOFT, que me apoiaram oportunamente quando havia necessidade.

AGRADECIMENTOS

Gostaria de agradecer aos meus amigos no auxilio que recebi com materiais e informações que ajudaram a escrever esta dissertação, e a todas as pessoas que de alguma forma contribuíram com o resultado final deste trabalho.

A Mateus Adelardo Fink peça fundamental no auxilio ao desenvolvimento do protótipo de sistema ora proposto, além de ser uma pessoa com qual me identifico muito, por seu espírito de luta, batalho e perseverança.

A meus amigos Carlos Alberto Barth, Djoni Koschanki, Geraldo Menegazo Varela, Marcio Elias Gonçalves, Marlon Volpi, e todos que de alguma forma contribuíram para realização deste trabalho.

E de maneira muito especial, ao meu orientador Prof Rosvelter João Coelho da Costa, pela orientação sempre oportuna, esclarecedora, inteligente, fazendo com que este trabalho pudesse ser realizado.

E claro, não poderia esquecer, agradeço muito a DEUS, pai Supremo.

Sumário

Lista de Figuras

Lista de Acrônimos

Aglets	-	**Agentes móveis (Agent + applets) de autoria da IBM**
API	-	**Application Programming Interface**
Applets	-	**Mini-aplicativos escritos na linguagem Java , capazes de serem executados em páginas da WWW**
CORBA	-	**Commom Object Request Broquer Archited**
Host	-	**Hospedeiro**
IDL	-	**Interface Difinition Langrage**
J-AAPI	-	**Java Aglet Application Programming Interface**
RMI	-	**Remote Method Invocation**
RPC	-	**Remote Procedure Call**
UML	-	**Unified Modeling Language**
URL	-	**Uniform Resource Locator**
GUI	-	**Grafic User Interface (Interfase gráfica com usuário)**

Resumo

Nesta dissertação propomos um ambiente computacional para solucionar problemas relacionados à mobilidade de aplicações de *software* e de seus usuários. Sua problemática e alternativas para solucionar problemas de locomoção de recursos computacionais dos usuários para diversos locais estão incluídos. Um usuário poderá utilizar esse sistema em qualquer máquina conectada à Internet em que o sistema esteja operante. Os dados gerados pelo usuário serão disponibilizados a partir de qualquer estação que compõe o ambiente.

Apresenta-se a análise do sistema proposto através da abordagem UML. Evidenciaram-se as principais características do sistema através de diagrama de casos de uso, diagrama de atividades e diagrama de estados. Para finalizar, descreve-se o desenvolvimento de um protótipo desse sistema.

Palavras-chave: Sistemas evasivos, mobilidade, usuários móveis, computação ubíqua.

Abstract

This dissertation proposes a computer environment that is able to solve problems related to mobile of software applications and users. A study on mobile computing is presented, which includes its problematic and also alternatives to solve problems in moving user computer resources to different places. A user will be able to make use of this system in any equipment connected to the Internet, in which the system is installed. The data produced by the user will be available from any station in the environment.

It is presented an analysis of the proposed system by using the UML approach. The main features of the system are shown by means of use case diagram, activity diagram and statechart diagram. The development of a prototype of this system is also described.

Key-Words: *Pervasive systems, mobility, mobile users, ubiquitous computing*.

Capítulo I

Introdução

Com a crescente influência da informática em todos os setores da economia, novos produtos de *hardware* e *software* surgem a cada momento. Avanços significativos na área de redes de computadores vem permitindo que gradualmente equipamentos de diversas naturezas possam ser interligados, deixando as antigas ilhas informatizadas para traz. A Internet tem papel cada vez mais importante na comunicação entre as pessoas, e torna-se cada vez mais popular para troca de informações entre elas.

Na história comercial da informática, inicialmente as informações eram concentradas em *main frames* (computadores de grande porte) em que os usuários utilizavam aplicações através de terminais, através dos quais, se conectavam e tornava possível a execução de aplicativos. Os terminais tinham a função exclusiva de conexão, sendo que não tinham poder de processamento.

Com o surgimento dos PC's (*Personal Computer*), os usuários puderam solucionar problemas do dia-a-dia, como podemos citar: editoração de textos, planilhas de cálculos, entre outras. Esses computadores trouxeram mais flexibilidade a seus usuários, pois permitiam a realização de tarefas mais pessoais, além de poderem ser ligados aos *main frames* e executarem as aplicações neles implantadas. Com o passar dos anos surgiram tecnologias que permitiram a ligação entre PC's, entre os quais, foi possível compartilhar recursos e torna-se possível a transferência de recursos entre eles, podendo realizar conexão através da Internet.

Com o surgimento da Internet, usuários de informática tiveram uma facilidade ainda maior de comunicação, troca de mensagens e documentos eletrônicos, não mais se limitando as redes internas. Com todos esses avanços os usuários conseguiram uma maior flexibilidade. Produtos como *Hand-Held, NoteBook* e *PalmTop*, aliados a evolução das redes de computadores, tornaram possível seu transporte e sua conexão em diversos pontos da rede.

Esses avanços possibilitaram estudos que mudaram as construções de sistemas, onde novas estruturas foram propostas. Com o surgimento de linguagem de programação, por exemplo, JAVA, houve uma maior flexibilidade entre as aplicações possibilitando integração entre diferentes plataformas de *hardware* e *software*. Com isto as necessidades e pretensões dos usuários tornaram-se cada vez mais elaboradas, possibilitando mobilidade na utilização de sistemas por usuários.

A mobilidade é muito freqüente na natureza e problemas de localização agravam-se quando falamos em usuários de informática. Um usuário solicita uma informação e deseja obter uma resposta, porém não necessariamente, onde foi feita a solicitação. Para solucionar este tipo de problemas encontramos várias abordagens, entre elas: o retorno de uma consulta pode ser através de e-mail [COR99]; computação ubíqua, esta solução mantém em uma lista todos usuários conectados ao sistema. Ao conectar novamente, será apresentado o resultado [RHO00].

Este trabalho propõe um ambiente padrão onde, independentemente de plataforma e localização geográfica, o usuário tenha condições de utilizar seu ambiente de trabalho. Os dados gerados pelo usuário poderão estar espalhados pela rede que compõem o sistema. Ao conectar-se em qualquer uma das estações, os recursos deixados em outro local deverão ser apresentados para que a sua utilização seja possível. Esse processo deverá ocorrer sem um mínimo de interferência do usuário, liberando o mesmo da tarefa de localizar documentos, ou ainda de transportá-los consigo.

Com tais processos automatizados acredita-se facilitar o ingresso de usuários no mundo da informática. Fazendo com que seu aprendizado inicial seja mais simples, pois o mesmo estará sempre utilizando o mesmo sistema, pelo menos aparentemente, pois poderá estar operando seu sistema em plataformas heterogênias.

1.1 Objetivos

O objetivo principal deste trabalho é especificar e realizar um protótipo de sistema que permita que usuários utilizem em diversos locais, sem se preocupar com a

localização de seus recursos e documentos. Este protótipo, ao ser invocado em qualquer computador conectado a rede, deverá disponibilizar tudo o que foi gerado pelo usuário em qualquer das estações. Quando o usuário selecionar um item, o sistema deverá transportá-lo até ele.

Os objetivos específicos são:

a) Estudar sistemas evasivos, usuários móveis, mobilidade, computação ubíqua e metodologias específicas para seu desenvolvimento;
b) Obter informações sobre as ferramentas de desenvolvimento que suportem plataformas operacionais de diversos fabricantes (multi-plataforma);
c) Escrever um protótipo de sistemas móvel na linguagem de programação JAVA, utilizando a tecnologia RMI (*Remote Method Invocation*).

1.2 Estrutura

Esta dissertação está estruturada em cinco capítulos que procuram demonstrar de forma lógica e seqüencial a construção desse trabalho:

- Primeiro capítulo

Apresenta a introdução do trabalho incluindo os objetivos e estrutura.

- Segundo capítulo

Encontram-se noções sobre mobilidade computacional, apresentando seus conceitos, características, problemas e efeitos. Agentes, agentes móveis e sistemas de agentes móveis também serão contemplados nesse capítulo. Apresenta informações tecnologias e técnicas utilizadas para a realização deste trabalho, em nosso caso RMI (Remote Method Invocation), que proporciona mobilidade aos nossos agentes.

- Terceiro capítulo

Apresenta a análise e modelagem do sistema proposto, sendo que esta foi construída em UML [FOW02, SCH99]. Serão apresentados diagramas de casos de uso e diagramas de atividade para auxiliar o leitor a compreensão desse trabalho.

- Quarto capítulo

Descreve o projeto e a realização do protótipo através de diagrama de classes e de seqüências.

- Quinto capítulo

Apresenta as conclusões e sugestões de continuação.

Capítulo II

Mobilidade

2.1 Introdução

Um requisito imprescindível aos sistemas cuja meta seja independência de plataforma e local geográfico é a sua capacidade de adaptação em tempo de execução. Os recursos variáveis, as falhas do sistema e as necessidades dos usuários devem sempre ser considerados quando tratamos de mobilidade. Para que os sistemas possuam mobilidade, devem se adaptar a novas situações de *Hardware* e *Software,* com o mínimo de interferência ou supervisão humana.

Situações como usuários migrando de um ambiente para outro, independente do tipo de conexão, largura de banda, configuração do monitor, entre outros, são comuns em ambientes que oferecem mobilidade. Para conseguir tais funcionalidades, este tipo de sistema deve ser construído com ferramentas que suportem ambientes hostis[1].

Sistemas tradicionais podem até detectar e contornar falhas com mecanismos neles implantados, mas dificilmente serão capazes de solucionar as causas do problema, pois apenas conhecem a solução do mesmo, e não as causas. Conseguem bloquear erros decorrentes de falhas, mas não conseguem reconhecer problemas como a degradação de desempenho sobre determinado tipo de comunicação.

Não foi difícil descobrir que a motivação para este tipo de aplicação foi inspirada em avanços tecnológicos com computadores portáteis tais como: *LapTop*, *PalmTop*, *Hand-Held,* entre outros. Estes equipamentos, por serem portáteis, tornaram essencial a conexão entre eles e outros equipamentos já existentes. Seus locais de conexão têm a necessidade de serem aleatórios, pois este tipo de equipamento foi justamente desenvolvido para dar mobilidade a seus usuários. Desta forma, atualmente, exige-se cada vez mais qualidade e eficiência nesses serviços.

---

[1] Ambientes de diferentes plataformas de *hardware* e *software.*

A facilidade cada vez maior e freqüente de obter conexão em diferentes pontos de rede é também motivo facilitador da mobilidade dos usuários. Ao se conectar em um novo ponto da rede, o usuário terá acesso aos recursos por ele mesmo disponibilizados em algum outro ponto. Para que isto aconteça, a transparência nas infra-estruturas (protocolos) que permitem este tipo de atividades faz-se necessária [BHA96, TAL95].

Aplicações em rede foram criadas para usuários não móveis. A mobilidade dos usuários vem sendo ampliada à medida em que surgem recursos para tal, o que começou a ser possível nos últimos anos graças às inovações tecnológicas criadas. *Softwares* como JAVA, transparentes à plataforma de desenvolvimento e também à plataforma de execução, auxiliaram estes avanços [ANA00].

Outro fator que impulsionou a mobilidade de usuários foi o surgimento da Internet, que foi originalmente criada para dar suporte a aplicações em rede tais como:

- Transferência de arquivos: um exemplo clássico é o FTP (*File Transfer Protocol*), vastamente utilizado para transferir arquivos de dados de qualquer natureza;
- E-mail (correspondência eletrônica): talvez até hoje a ferramenta mais utilizada para transferência de mensagens entre usuários da Internet;
- Conexão remota como emuladores de terminais de *Main Frame*, muito utilizado por instituições de grande porte.

Tais aplicações deram origem a um novo rumo na informática, pois agilizaram processos que até então eram dispendiosos e lentos. Um exemplo clássico disso é o correio. Com o advento do E-mail, principalmente as grandes instituições começaram a fazer uso dessa tecnologia. É claro que para que tal tecnologia tivesse aprovação da grande massa de usuários, tornou-se necessário dar a eles condições de utilizá-la. Isto aconteceu naturalmente com o passar dos anos, através dos avanços da tecnologia e da diminuição dos seus custos.

2.2 Mobilidade computacional

A mobilidade é muito freqüente na natureza e problemas de localização agravam-se quando falamos em usuários de informática.Em [NAS00], são declarados os seguintes tipos de problemas relacionados à mobilidade:

- Problemas com localização: de alguma maneira, o usuário deverá definir sua localização, ou melhor, o sistema deverá descobrir estes dados para o usuário. Para isso, seriam necessários aparelhos como GPS, indicadores da localização geográfica exata do ponto em que o mesmo foi acionado;
- Tempo de resposta: se o usuário está se deslocando, dependendo de sua velocidade e da distância do próximo posto de gasolina, a resposta poderá fazer parte do passado, pois o usuário pode ter passado pelo local solicitado;
- Perda de comunicação: se o usuário estiver se deslocando, e efetuar a consulta durante seu deslocamento e sair da área de abrangência do meio de comunicação, não obterá a resposta.

Para permitir ao usuário consultar, deslocar-se e receber o resultado da consulta, é possível agir, basicamente, de duas formas [COR99].

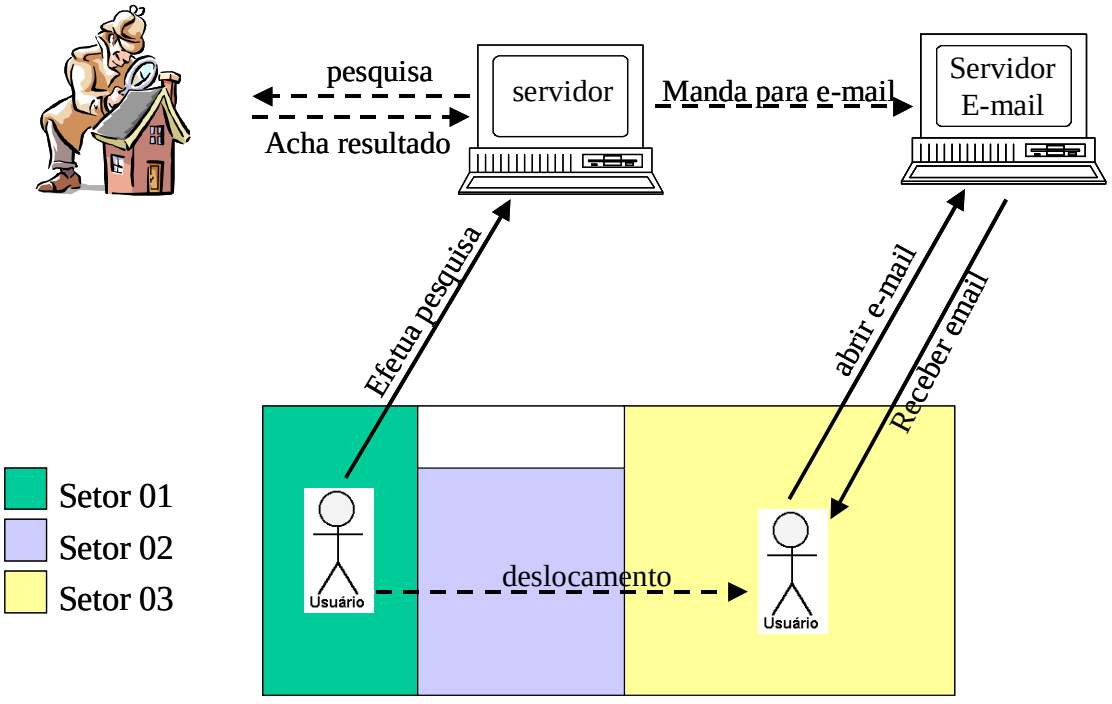


Figura 2.1 - Solicitação retorna por e-mail

Uma das alternativas, representada na figura 2.2, poderia ser utilizar o e-mail do usuário. Neste caso, o usuário teria que consultar sua caixa postal para obter a informação. Essa forma traria ao usuário algumas vantagens:

- O usuário poderia se desconectar da rede e reconectar-se quando e onde fosse necessário e possível, trazendo-lhe grande flexibilidade. Porém, o limita a uma tecnologia, pois teria que solicitar a consulta e ainda ele mesmo buscar o resultado da pesquisa efetuada, abrindo sua caixa postal;
- E-mail é uma tecnologia suportada por uma gama muito grande de aparelhos;
- Os softwares para gerenciamento de e-mail são bastante simples e comuns.

A segunda alternativa para solucionar o problema divide-se em dois casos:

1. Cada servidor mantém uma lista com todos os servidores ativos desse sistema. Quando o usuário se desconectar do sistema, seriam notificados todos os servidores que compõem o ambiente, através de um agente. O mesmo aconteceria se o usuário re-conectar ao sistema. Quando o agente terminar sua tarefa, permanecerá esperando no ultimo hospedeiro que visitou, aguardando a re-conexão do usuário ao sistema. Quando isto ocorrer, o agente migrará para o local em que o usuário se re-conectou e irá apresentar-lhe o resultado da consulta;
2. Um mecanismo centralizador estaria presente no domínio, mantendo as informações sobre o local de conexão do usuário (em uma rede fixa). Se o agente terminar sua busca e o usuário que a solicitou não estiver mais conectado ao sistema, o agente migra para o local onde o usuário conectou-se e suspende a execução. Quando o usuário retornar ao sistema, seu servidor notificará sua re-conexão retomando a execução.

Em outro mecanismo encontrado, denominado pelo autor [RHO00] de U*biquitous Computing* (computação ubíqua), os usuários poderão interagir com centenas de computadores em algum momento. Cada usuário, mesmo que oculto no ambiente, pode estar conectado a qualquer outro do sistema. É uma espécie de sala na qual existe uma série de sensores detectando quem está entrando e saindo da mesma. Segundo o autor, há uma série de aplicações implementadas com tal tecnologia. Fazendo uma analogia com o mundo real, todos gostariam de ser onipresentes e é normal que haja esta mesma vontade em sistemas. Isto é possível, mas, para tal, enfrentar-se-ão alguns problemas:

- Questão de privacidade: provavelmente, um dos maiores problemas e um dos mais importantes a ser resolvido. Como todos os integrantes se conhecem, essa questão torna-se complexa;
- Dificuldade com informações personalizadas: pode existir uma infinidade de usuários compartilhando recursos como, por exemplo, Banco de Dados. Além disso, muitos outros usuários podem estar conectando e desconectando-se a todo momento, tornando difícil o gerenciamento.

Em outra abordagem denominada *Pervasive computing Systems*, o autor sugere que o próprio sistema deve possuir a habilidade de auto-adaptação durante a conexão, execução, falhas na rede e transparência à plataforma. Cada vez mais se exige tal atitude dos sistemas, pois a intervenção humana para resolver problemas dessa natureza é sempre onerosa e, às vezes, até de difícil solução.

Muitos sistemas tratam adaptações simplesmente como exceções. Na verdade, o que deveria acontecer é os sistemas utilizarem seu processamento para resolução de problemas e não para se auto ajustar[CHE02].

2.2.1 O problema da mobilidade computacional

Para ilustrar como a mobilidade na camada de rede é um problema, enfatiza-se a distinção entre conceitos de *Name* (Nome) e *Address*(Endereço). O Nome é

identificador de localização independente de um *Host* (hospedeiro). Endereço é o retorno que indica o ponto em que está atracado este Hospedeiro. Para que tal ciclo aconteça, um cliente efetua uma pesquisa em um servidor passando como parâmetro um Nome, obtém-se do mesmo um retorno trazendo o endereço; então, tem-se o endereço do destino. O destino deverá ser cadastrado no servidor anteriormente, passando seu Nome e Endereço como parâmetro. Tal configuração pode ser vista na figura 2.3.

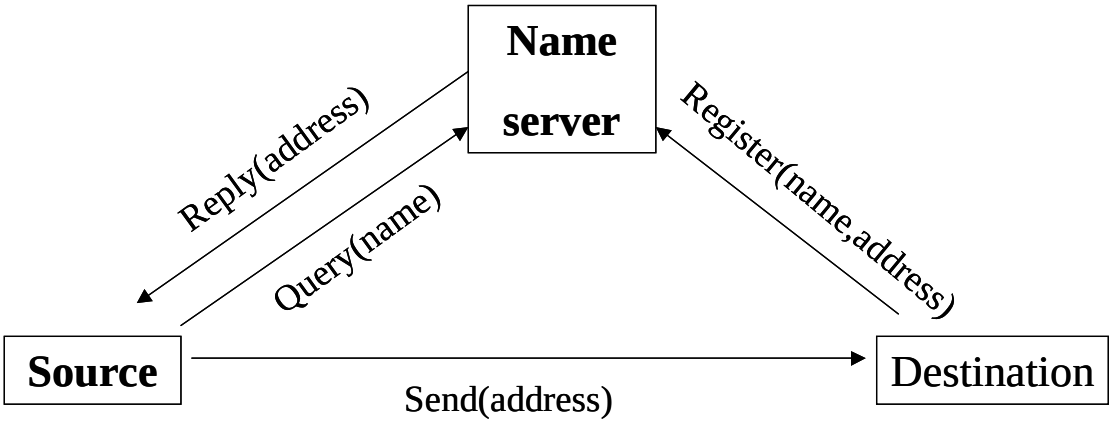


Figura 2.2 - Problema da mobilidade.

Fonte: [BHA96]

2.2.2 Efeitos da mobilidade computacional

Os efeitos da mobilidade são de extrema relevância [TAL95]. São eles:

- *Packed Delay* (Espera de um pacote): tempo que um pacote leva para trafegar da saída à chegada. Consiste em dois componentes:
    - *Propagation Delay* (tempo de propagação): a propagação do pacote depende do caminho do fluxo de dados por onde deverá passar.

- *Congestion Delay* (tempo de congestionamento): o tempo perdido pelo pacote na fila do caminho a ser percorrido.

- *Packed loos rate* (Taxa de pacote perdido): Fração de pacotes perdidos por fluxo.
- *Dellay jitter* (impaciência na espera): variação do tempo de chegada entre os pacotes.
- *Throughput* (transferência): Qualidade do fluxo, depende diretamente da quantidade de dados transportados em determinado tempo.

2.2.3 Objetos móveis

Estruturas conhecidas na informática como arquivos fazem parte desse conjunto. Elementos como processos também entram na lista. Na verdade, toda e qualquer estrutura de dados poderá ser chamada de objeto móvel [HER99].

2.3 Agentes

Entidades computacionais aptas a habitar um ambiente virtual complexo e dinâmico, raciocinando e agindo no mesmo de modo a realizar as tarefas e objetivos para os quais foram criados [WOO96].

Agentes possuem propósitos que são gerados por si próprios, ao invés de adotados de outros agentes. Estes se originam de motivações definidas como desejos ou preferências, podendo levar à geração e à adaptação de objetivos. Assim, agente motivado é um agente que possui sua própria agenda para solução e comportamento, de acordo com sua motivação interna [SOU97].

2.3.1 Agentes móveis

Agente móvel é um programa executável que pode migrar de uma máquina para outra máquina, em redes heterogêneas [GRA00].

Agentes móveis não são largamente utilizados na confecção de aplicações. Há três motivos para isso:

1. Necessitam de ambientes altamente dinâmicos, com alto poder de computação aliado a altas velocidades de transmissão;
2. Devido à sua grande flexibilidade, não é aconselhável implementar agentes móveis em aplicações que devam ser robustas e de alto desenpenho;
3. O mais importante é que não existem aplicações que somente poderão ser implementadas com agentes móveis. Existem sempre tecnologias mais tradicionais que, de uma outra forma, levam ao resultado desejado.

Sistemas de agentes móveis representativos podem ser categorizados em três grupos [GRA00]. O primeiro grupo pode ser escrito em diversas linguagens; o segundo, somente em JAVA (a linguagem mais popular para escrever agentes); o terceiro pode ser escrito em linguagem própria e também em JAVA.

1. Sistemas de múltiplas linguagens
   - Ara - Suporta agentes escritos em Tcl, C/C++ e Java. Ara possui uma instrução denominada *go*, que captura o estado do agente, transfere-o para máquina-destino e recomeça a execução exatamente no ponto do *go*;
   - D'Agent - Suporta agentes escritos em Tcl, C/C++, Scheme e Java. Como em ARA, D'Agent também possui a instrução *go*, salva todo o contexto do agente, transfere-o e o restaura na máquina-destino. A única diferença é que o servidor D'Agent suporta Multi-Tread;
   - Tacoma - Suporta agentes escritos em C/C++, ML, Perl, Pyton e uma série de outras linguangens, inclusive Java. Diferente de ARA e D

‘Agent, Tacoma não propicia a captura do estado do objeto de maneira tão facilitada. Ao migrar para outra máquina, o agente cria uma pasta, salva seu contexto, envia-a para nova máquina e somente então reinicia sua execução.

2. Sistemas baseados em JAVA – Java é a linguagem mais popular para desenvolver aplicações distribuídas. Tal sucesso se deve à sua transparência de plataforma, segurança, suporte à seriação, suporte a RMI.
    - Aglets - Pioneira na implementação de agentes em JAVA. Aglets utiliza-se de métodos para migrar. Quando um agente migra, invoca-se o método *onDispatch*, interrompem-se as tarefas do agente; serializa-se o agente e seu contexto; enviando-os à máquina destino. Lá, invoca-se o método *onArived*, que remonta o agente e invoca-se o método *run*, tornando à execução do agente;
    - Concórdia - Seu ponto mais forte é a segurança. Move o objeto agente e os dados como a grande maioria, mas não o estado de suas tarefas; possui um pacote com o itinerário de endereços que deve visitar, os quais podem ser ajustados por ele mesmo enquanto trafega;
    - Jumping Beans - implementa algo que chama de agência (*agency*). Cada uma está associada a um domínio (*domain*), possuidor de um servidor que autentica as agências ligadas a ele. Com isso, os agentes móveis podem migrar entre as agências. O que não está claro em sua literatura é se o contexto da tarefa é salvo ou não;
    - Voyager – ambiente de agentes móveis integrados com CORBA (*Commom Object Request Broquer Archited*). Apresenta um ambiente bastante conveniente para interagir de forma transparente com objetos remotos e para os mesmos se moverem de um lugar para outro. Quando um objeto se move, deixa para trás um sinal que redirecionará qualquer mensagem para a nova localização.
3. Outros sistemas

- Messengers – utiliza código móvel para construir sistemas distribuídos flexíveis, não especificamente sistemas de agentes móveis;
- Obliq – é um interpretador de scopo léxico de uma linguagem orientada a objetos. É um conjunto de campos contendo métodos, caminhos e valores. Um objeto pode ser criado ou clonado em uma máquina destino, ou ainda migrar até ele;
- Telescript – primeiro sistema de agentes móveis comercial. Nele, cada ponto da rede roda um servidor, que mantém um ou mais *places* (abre as portas para entrada de agentes habilitados, mediante autenticação e criptografia).

Relacionamos, a seguir, algumas propriedades dos agentes móveis [RHO00]:

- Agentes são autônomos: podem ser disparados em um sistema e encarregados de atingir determinadas metas, sem intervenção humana;
- Agentes pró-ativos: agentes possuem suas próprias tarefas e metas, podem agir independentemente de outros agentes ou do ambiente;
- Agentes são auto-descritivos: podem se auto-descrever, bem como disponibilizar tarefas a eles definidas;
- Agentes podem interagir: podem compartilhar tarefas entre si ou, ainda, um agente poderá repassar tarefas para outros, através de eventos assíncronos. Tal mecanismo pode ser implementado por Java RMI;
- Agentes podem se mover: devem conseguir mover-se de um lugar para outro.

2.4 RMI(*Remote Method Invocation*) - Invocação Remota de Métodos

O mecanismo de RMI consiste em uma evolução de RPC (*Remote Procedure Call*) que foi disponibilizada pela grande maioria das linguagens de programação desde

o começo da década de 80. A RPC permite que programas escritos de maneira procedural em linguagens, tais como C ou Pascal, invoquem uma função que se encontra em outra máquina, como se esta fizesse parte do próprio programa [DUA00, DEI01, WOL02, HÜB99, KON01].

Uma das diferenças entre RMI e RPC é que inexiste a necessidade dos programadores aprenderem a IDL (*Interface Definition Language*). Como RMI suporta somente linguagem JAVA, são requeridas somente as próprias interfaces de JAVA. Se houver a necessidade de comunicação com objetos diferentes de JAVA, a solução seria utilizar IDL JAVA (introduzida a partir da versão 1.2), desde que a linguagem que gerou este objeto suporte CORBA (*Commom Object Request Broquer Archited*).

Para que RMI se torne acessível remotamente, necessita ser registrada na máquina cliente. Uma vez que um método (ou serviço) de um objeto JAVA foi registrado como sendo acessível remotamente, o cliente pode pesquisar este serviço e obter uma referência que o permita utilizá-lo. Sua sintaxe é idêntica à invocação de métodos locais. A RMI implanta classes capazes de decompor, recompor e transportar qualquer tipo de classe e objeto. Com isto, podemos transportar os mesmos pela rede, com a vantagem do programador não se preocupar com a transmissão, pois isso está contemplado no *framework* que suporta RMI.

Um conjunto de pontos que faz com que sejam suportados objetos distribuídos em JAVA [JAV99] é que esta linguagem:

- suporta invocação remota de objetos em diferentes máquinas virtuais;
- suporta solicitação de resposta de servidores para aplets;
- integra o modelo de objetos distribuídos dentro da linguagem de programação JAVA;
- distingue diferenças entre o modelo de objetos distribuídos e o modelo da dplataforma local de objetos em JAVA;
- permite escrever aplicações distribuídas seguras da forma mais simples possível;
- preserva "*Type-safety*" (Segurança em seus tipos) em ambientes de execução das plataformas JAVA;

- suporta variações na semântica de referências em objetos remotos. Ex: referencias não persistentes, referências persistentes e ativação lenta.

2.4.1 Funcionamento

O funcionamento de RMI consiste, basicamente, em dois programas: o cliente e o servidor. O servidor instancia objetos remotos, referencia-os com um nome e faz um "*BIND*" deles numa porta, onde estes objetos esperam por clientes que invoquem seus métodos. Já o cliente referencia remotamente um ou mais métodos de um objeto remoto. RMI fornece os mecanismos para que a comunicação entre cliente e servidor seja possível [DUA00, JAV99].

Aplicações distribuídas precisam, portanto, executar as seguintes ações:

- Localizar Objetos Remotos - Uma aplicação pode usar dois mecanismos para obter referências de objetos remotos. Ela pode registrar o objeto remoto com a ferramenta de nomes do RMI, chamada "rmiregistry", ou pode passar e retornar referências aos objetos remotos como parte de sua operação normal;

- Comunicar-se com Objetos Remotos - Os detalhes de comunicação entre objetos remotos são tratados pelo RMI, ou seja, para o programador, a comunicação remota é semelhante a uma chamada ao método local;

- Carregar *"bytecodes"* de objetos móveis - Como o RMI permite que objetos remotos sejam passados como parâmetros numa função, fornece os mecanismos necessários para carregar o código dos objetos remotos.

A figura 2.4 ilustra como uma aplicação RMI distribuída usa o "rmiregistry" para obter uma referência ao objeto remoto. O servidor chama o registro para associar um nome ao objeto remoto. O cliente procura o objeto remoto pelo seu nome no registro do servidor e, então, invoca um método nele. A ilustração também mostra que o RMI pode utilizar um servidor "web" para carregar os "bytecodes", de servidor para cliente e vice-versa, de acordo com as necessidades da aplicação.

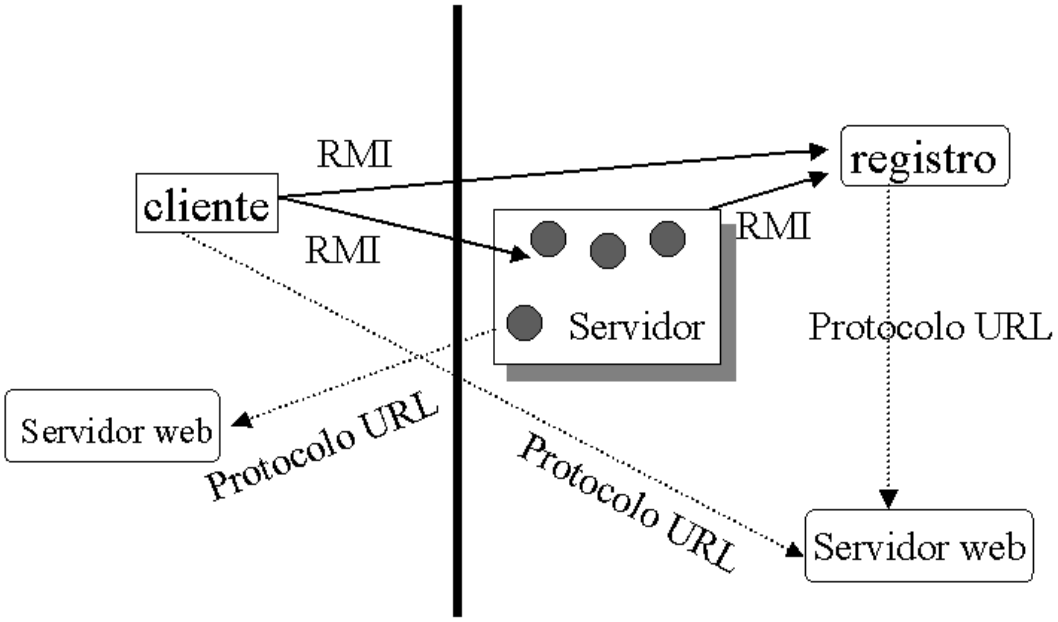


Figura 2.3 - Aplicação de RMI distribuído

Os passos abaixo descritos auxiliam no entendimento de como realiza-se a implementação de RMI [ROC02].

1. Definir a interface;
2. Implementar os objetos remotos;
3. Implementar um servidor para os objetos;
4. Compilar os objetos remotos;
5. Gerar *stubs* e *skeletons*;
6. Escrever, compilar e instalar o(s) cliente(s);
7. Instalar o *stub* no(s) cliente(s);
8. Iniciar o RMI Registry no servidor;

9. Iniciar o servidor de objetos;

10. Iniciar os clientes, informando o endereço do servidor.

Capítulo III

Análise e Modelagem do Sistema Móvel

3.1 Introdução

Encontra-se neste capítulo o estudo de uma aplicação com mobilidade computacional denominada “SISTEMA DE FERRAMENTAS COM RECURSOS MÓVEIS”.

Em uma instituição, tem-se a necessidade de disponibilizar ferramentas para seus colaboradores. Agendas, editores de texto são exemplos simples de ferramentas vastamente utilizadas em instituições. É claro que, dependendo das necessidades de cada instituição e do perfil dos usuários, outras ferramentas poderão compor esta lista. Um problema ocorre quando o usuário tiver a necessidade de utilizar suas aplicações em outra máquina. Para isto, o mesmo deverá levar uma cópia de seus documentos.

Este capítulo propõe uma solução para este tipo de problema. Quando um usuário disparar a execução de qualquer ferramenta disponível, o sistema se encarrega de apresentar a ele uma lista de dados previamente gerados com a ferramenta selecionada. Estes dados poderão estar espalhados em diversas máquinas que compõem o ambiente do usuário, mas isto deverá ser transparente ao usuário.

Criou-se um agente que verifica e gera uma lista dos dados de acordo com a ferramenta selecionada pelo usuário. Ao selecionar um dos dados da lista, o sistema se encarrega de transportá-lo até a estação onde o usuário o solicitou, para, então, executar a ferramenta e abrir o documento selecionado.

3.2 Visão geral

Nome do sistema

Sistema de ferramentas com recursos móveis (SIFREM).

Padrões de nomes adotados

- Ferramentas: programas que executam tarefas (Ex: agenda e editor de texto);
- Recursos: dados gerados por ferramentas (móveis);
- Estação: equipamento pelo qual o usuário utiliza o sistema;
- Ambiente: gera as interfaces de menus por onde o usuário interage;
- Sistema: é composto de todas as estações que o usuário do sistema utilizou.

Descrição do sistema

Uma instituição tem a necessidade de oferecer diversas ferramentas para seus colaboradores.

Usuários utilizam uma série de ferramentas em seu dia-a-dia.

Usuários têm a necessidade de utilizá-las em locais geograficamente dispersos.

Usuários geram recursos e têm a necessidade de editá-los em locais geograficamente dispersos.

Fatores de risco

- Ausência de conexão de rede no exato momento do transporte de recursos;
- Conexão ineficiente, baixa velocidade de transmissão durante o transporte;

- Queda de conexão durante o transporte de recursos;
- Ausência de instalação mínima para execução do sistema.

Descrição dos atores

- **Usuário -** pessoas que utilizarão o sistema proposto, para o qual serão disponibilizados recursos e ferramentas, através de menu.
- **Agente Calibra** – ao constatar tratar-se de nova estação e percorrerá todas as estações do ambiente, notificando-as da existência de nova estação.
- **Agente Procura –** responsável por encontrar recursos e suas respectivas estações, de acordo com a ferramenta selecionada pelo usuário.
- **Agente Transporte** – responsável pelo transporte dos recursos selecionados pelo usuário.

3.3 Diagramas de casos de uso

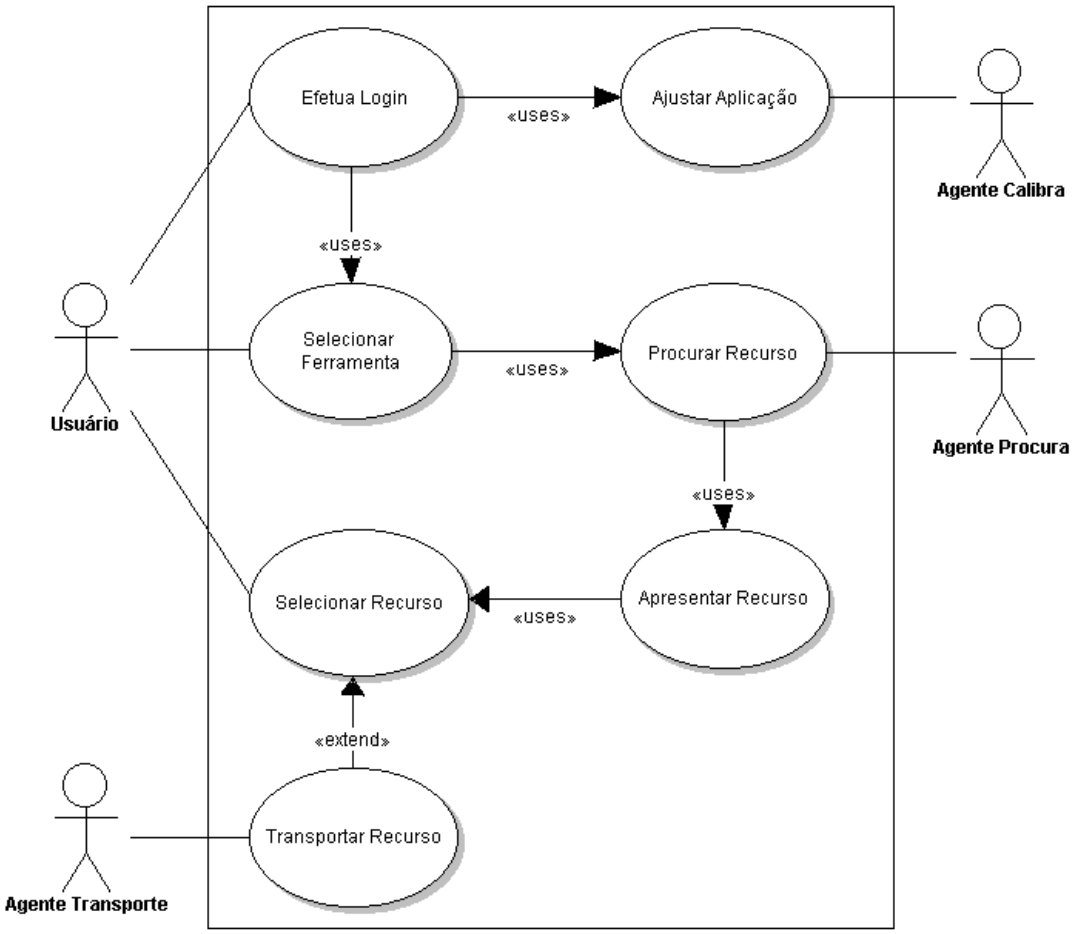


Figura 3.1 - Diagrama de casos de uso

O diagrama de casos de uso, representado na figura 3.1, demonstra a interação dos atores com o sistema, bem como identifica os próprios:

- Usuário
- Agente Calibra
- Agente Procura
- Agente Transporte

O ator usuário citado anteriormente é o único ator que interage com o sistema. Os outros atores, apesar de serem sub-sistemas do sistema proposto, são considerados atores, pois estendem ações do próprio usuário.

O ator usuário interage diretamente com o sistema em três situações:

1. Caso de uso **Efetuar Login**: neste caso será solicitado ao usuário que entre com seus dados (nome e senha). O sistema fará uma verificação, buscando este usuário na estação local. Se o usuário já utilizou o sistema nesta estação, e seus dados forem válidos, o sistema progride chamando o caso de uso **Seleciona Ferramenta**, descrito abaixo. Porém, se este usuário não for encontrado nesta estação, o sistema lhe apresentará uma tela na qual ele poderá optar por:
    a. Criar um novo usuário: o sistema solicitará ao mesmo que informe um endereço de Hospedeiro (alguma estação que já tenha sido utilizada). Após isto, será também solicitado ao usuário o endereço da estação que está sendo operada neste momento. Após este processo, o sistema progride chamando o caso de uso **Ajustar Aplicação**, o qual está descrito abaixo;
    b. Entrar com seus dados novamente: neste caso, será reapresentada a tela de entrada de dados, onde o usuário entrará novamente com seu nome e senha (reinicia o caso de uso **Efetuar Login**).
2. Caso de uso **Selecionar Ferramenta**: mediante apresentação de um menu, o usuário seleciona a ferramenta desejada. Após esta seleção, o sistema acionará o caso de uso **Procurar Recurso**, descrito abaixo.
3. Caso de Uso **Selecionar Recurso**: mediante apresentação de uma lista que será gerada pelo caso de uso **Apresentar Recurso**, o usuário poderá selecionar o recurso que desejar editar. Após esta seleção, o sistema irá verificar se o recurso selecionado encontra-se na estação local. Caso não esteja, o sistema acionará o caso de uso **Transportar Recurso**. Ao término das operações acima, o sistema irá abrir a ferramenta e o recurso para o usuário.

Os outros quatro casos de uso não possuem intervenção direta do usuário. Esses são casos disparados para prover os recursos necessários para o funcionamento do sistema proposto.

1. Caso de Uso **Ajustar Aplicação**: Este caso de uso é acionado após o usuário ter disparado a aplicação, ter entrado com seu nome e senha e o sistema constatar tratar-se de uma nova estação. Com esta operação, o sistema chamará o `Agente Calibra.` Este agente divulgará esta nova estação para todas as estações por onde este usuário passou e utilizou a sistema. Terminado o processo do `Agente Calibra,` o sistema progride chamando o caso de uso **Escolhe Recurso** descrito acima.
2. Caso de Uso **Procurar Recurso**: Este caso de uso é acionado após o usuário ter selecionado uma ferramenta disponibilizada para ele através de um menu. O sistema progride, disparando o `Agente Procura.` Esse agente irá percorrer todas as estações que compõem o ambiente do usuário. Ao encontrar recursos referentes à ferramenta selecionada e que pertencem ao usuário conectado, o agente irá montar uma lista de recursos que será disponibilizada ao usuário em forma de lista, veja o caso de uso **Apresentar Recurso**.
3. Caso de uso **Apresentar Recurso**: Este se encarrega de exibir os recursos encontrados pelo `Agente Procura`, para que o usuário tenha a opção de escolha.
4. Caso de Uso **Transportar Recurso**: Este caso de uso aciona o `Agente Transporte`, que se encarrega de transportar o recurso da estação de onde foi encontrado para a estação na qual foi solicitado. Além disso, apaga o recurso da estação de origem, evitando redundância de recursos.

Efetuar Login

Este caso descreve o processo pelo qual o usuário obtém acesso ao sistema.

Atores

- Usuário

Fluxo dos Eventos

Fluxo básico

1. O caso inicia-se quando o usuário disparar sua aplicação. Para dispará-la, deve-se ter em mídia a aplicação inicial (esta aplicação inicial contempla o mínimo para que o sistema possa ser utilizado).
2. Caso esta estação esteja sendo utilizada por este sistema pela primeira vez, o sistema acionará o caso de uso *Ajustar Aplicação*.
3. O sistema apresentará ao usuário uma tela de login.
4. O usuário preenche a tela de login com seu nome e senha.
5. O sistema irá avaliar se nome e senha do usuário são válidos.
6. Se os dados do usuário forem válidos, o sistema irá para o caso de uso *Escolhe recurso*.
7. Se os dados do usuário forem inválidos o sistema irá retornar ao caso de uso *Efetuar Login*.

Cenários Secundários

- Senha Incorreta.
- Rede não acessível.

Ajusta Aplicação

Este caso descreve o processo pelo qual o sistema permite a utilização de uma nova estação para o sistema proposto.

Atores

- `Agente Calibra`

Fluxo dos Eventos

Fluxo básico

1. O sistema apresentará uma tela, julgando que poderá se tratar de uma nova estação;
2. O sistema apresentará uma tela, na qual o usuário deverá informar o endereço da nova estação;
3. O sistema apresentará uma tela, na qual o usuário deverá informar a descrição da nova estação;
4. O sistema apresentará uma tela, na qual o usuário deverá informar o endereço de alguma outra estação que faça parte de seu ambiente.

Cenários Secundários

- Endereço local incorreto
- Endereço de estação remota incorreto

Selecionar Ferramenta

Este caso descreve o processo pelo qual o usuário obtém acesso às ferramentas sistema (representadas em menu).

Atores

- Usuário

Fluxo dos Eventos

Fluxo básico

1. O caso inicia-se após o usuário ter efetuado Login.
2. O sistema disponibilizará ao usuário uma tela contendo um menu com as ferramentas disponíveis no sistema.
3. O usuário seleciona no menu a ferramenta que deseja executar.
4. O sistema progride, chamando o caso de uso *Procurar Recurso*.

Cenários Secundários

Procurar Recurso

Este caso descreve o processo pelo qual o sistema efetua a busca pelos recursos da ferramenta que o usuário selecionou.

Atores

- `Agente Procura`

Fluxo dos Eventos

Fluxo básico

1. O caso inicia-se quando o usuário selecionar uma ferramenta disponibilizada pelo sistema;
2. O sistema acionará o `Agente Procura`, que irá percorrer as estações que compõem o ambiente do usuário;
3. A cada estação visitada, irá procurar os recursos de propriedade do usuário que foram gerados com a ferramenta selecionada;
4. O `Agente Procura` retorna o resultado da busca, o sistema progride, acionando o caso de uso *Apresentar Recurso*.

Cenário Secundário

- .O recurso solicitado pode não ser encontrado, pois a estação que efetuou a seleção da ferramenta pode estar sem conexão a rede.

Apresentar Recurso

Este caso descreve a apresentação dos recursos pertencentes ao usuário e que foram gerados com a ferramenta selecionada.

Atores

Fluxo dos Eventos

Fluxo básico

1. Após executar o caso *Procurar Recurso,* com o resultado da procura, o sistema irá montar uma tela contendo uma lista de recursos encontrados;
2. Fim de caso.

Cenário Secundário

Selecionar Recurso

Este caso descreve o processo pelo qual o usuário deverá selecionar um dos recursos disponibilizados pelo sistema.

Atores

Fluxo dos Eventos

Fluxo básico

1. O usuário seleciona determinado recurso;
2. Duas situações poderão ocorrer nesse caso:
    a. O recurso poderá estar localizado na máquina que está sendo executada (neste caso, o sistema não necessita transportá-lo);
    b. O recurso poderá estar localizado em outra estação: o sistema irá acionar o caso de uso *Transportar Recurso;*
3. O sistema irá executar a ferramenta juntamente com o recurso selecionado;
4. Fim de caso.

Cenários Secundários

Transportar Recurso

Este caso descreve o processo pelo qual o sistema transporta o recurso solicitado para a estação que o solicitou.

Atores

- `Agente Transporte`

Fluxo dos Eventos

Fluxo básico

1. O sistema dispara o `Agente Transporte` mediante parâmetro como nome do recurso e estação de localização;
2. O `Agente Transporte` transporta o recurso solicitado da estação onde o mesmo se encontra até a estação solicitante;
3. O `Agente Transporte` elimina o recurso transportado da estação de origem;
4. Fim de caso.

Cenário Secundários

3.4 Arquitetura do sistema

3.4.1 Diagrama geral

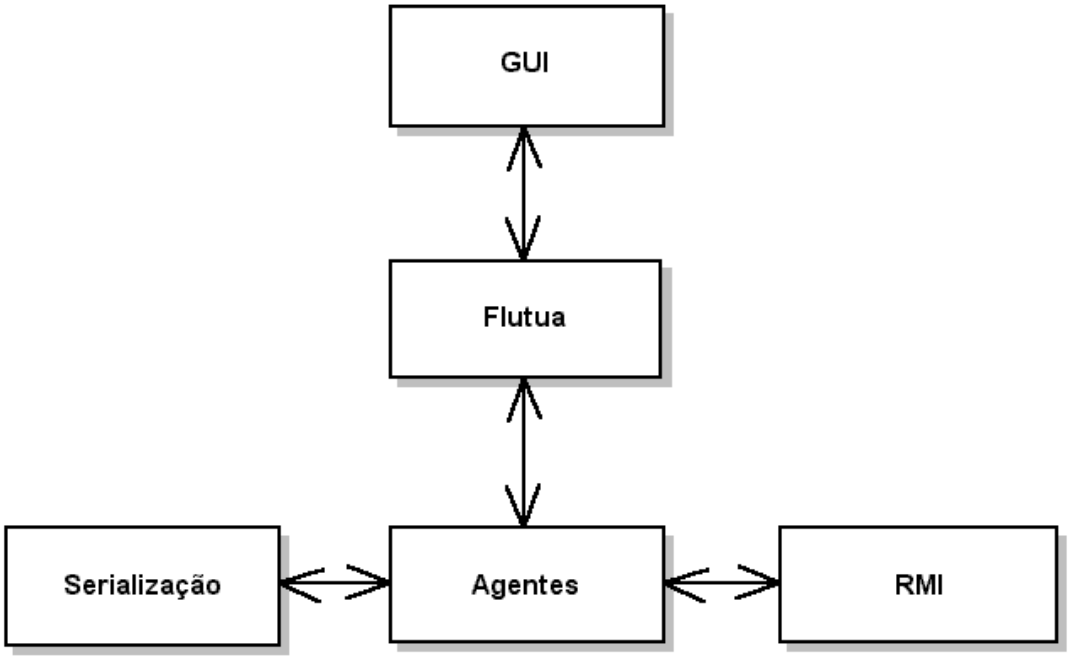


Figura 3.2 - Diagrama geral

GUI - *Grafic User Interface* (Interfase gráfica com usuário)
Este sub-sistema é responsável pelo controle da interação do usuário com o sistema.

Flutua - Este é o ambiente móvel proposto para a solução do problema.

Agentes - responsáveis pelos acessos móveis. Foi dada ênfase às funcionalidades proporcionadas por esses objetos.

- **Agente Calibra** – Para cada nova estação que o usuário venha a utilizar, será instanciado um novo agente. Este agente é responsável pela divulgação da nova estação nas estações que compõem o sistema. Para tanto, ele visita cada estação, adiciona o nome e o endereço da nova estação ao objeto denominado `host`.

- **Agente Procura –** Quando um usuário selecionar do menu uma ferramenta, será instanciado este agente. Sua função é percorrer todas as estações que compõem o sistema e disponibilizar uma lista de todos os recursos disponíveis e pertencentes ao usuário e à ferramenta selecionada. O protocolo utilizado para formar a lista encontra-se descrito na figura 3.4.
- **Agente Transporte** – Quando o usuário selecionar um recurso da lista de recursos disponíveis, este agente é instanciado para efetuar o transporte do recurso da estação de origem para a estação destino. Quando terminar o transporte, o agente retorna à máquina de origem e elimina o recurso, evitando redundância de dados.

Seriação - Faz a persistência dos dados (armazena).

RMI - Subsistema de comunicação responsável pelo suporte à mobilidade dos agentes.

3.4.2 Composição do sistema

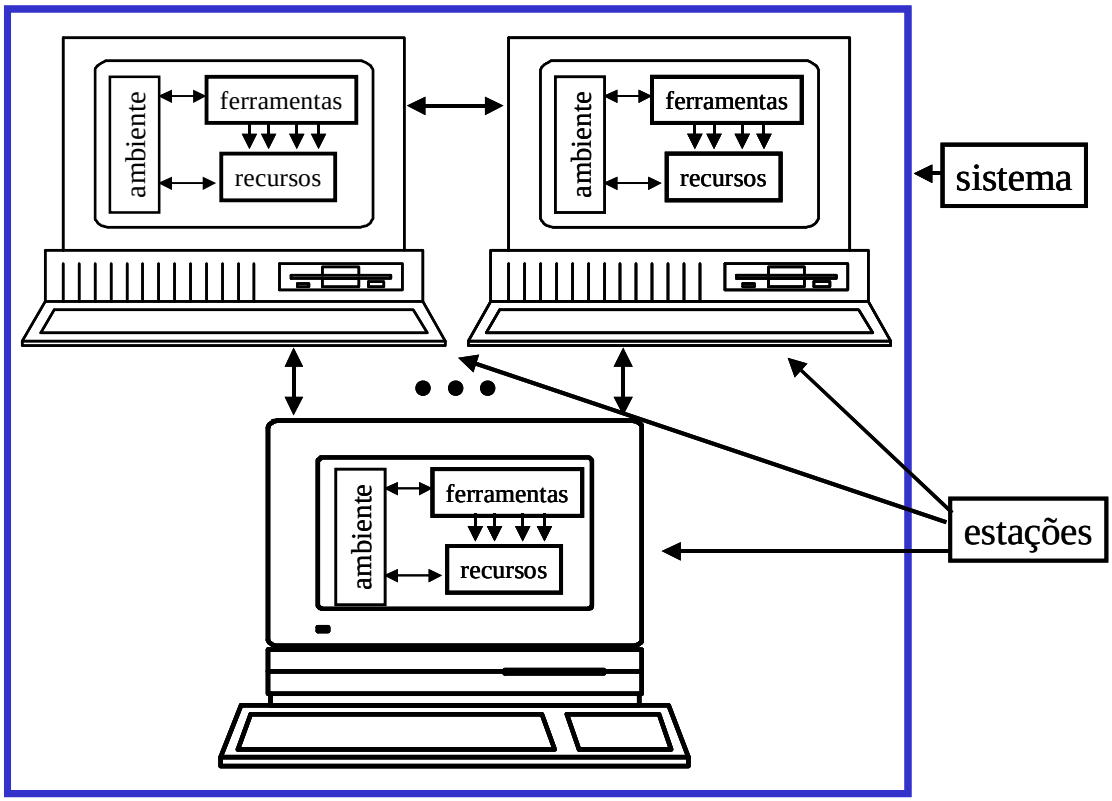
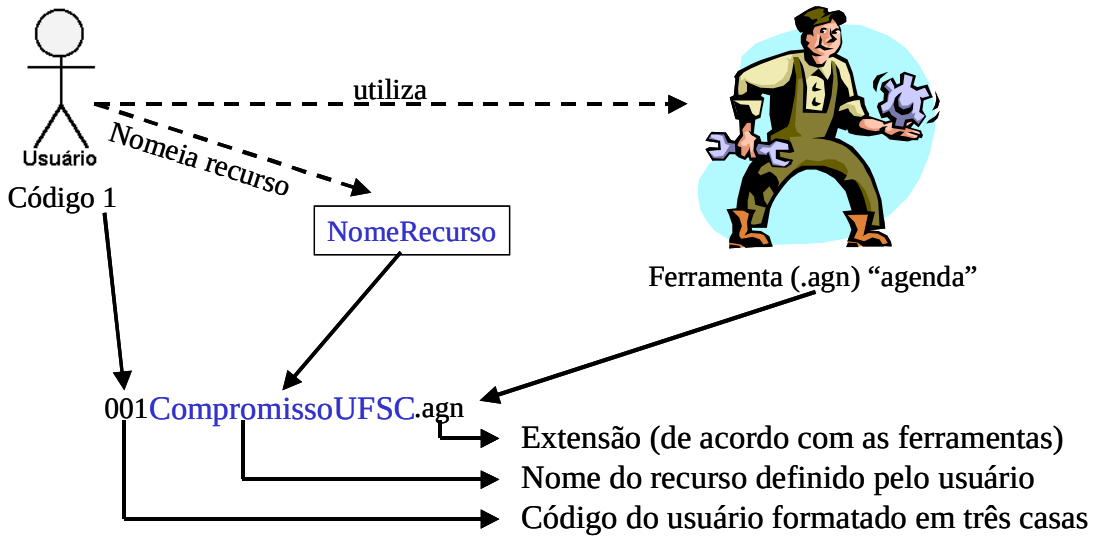


Figura 3.3 - Composição do sistema

Para se ter uma idéia da composição do sistema criou-se à figura. 3.3. Cada nova estação utilizada por um usuário será composta pelo ambiente, ferramentas e recursos, sendo que os recursos serão gerados pelo próprio usuário. O sistema irá crescer a cada nova estação utilizada.

3.4.3 Protocolo



Nome físico : 001CompromissoUFSC.agn

Nome lógico: CompromissoUFSC

Figura 3.4 - Protocolo de tratamento de recursos

Para conseguirmos identificar os recursos por usuário e por ferramenta, criou-se o protocolo representado na figura 3.4. Na identificação do autor de um recurso, utilizou-se o seu código formatado em três casas. Para identificar a ferramenta utilizada para gerar um recurso, utilizou-se uma extensão de arquivo. Cabe ao usuário definir um nome para o novo recurso. O sistema se encarrega de compor o nome físico. As listas de recursos serão geradas de acordo com a ferramenta selecionada pelo usuário bem como seu código. Nas listas de recurso selecionadas para o usuário, constarão o nome do recurso[2] e a descrição da estação onde o recurso está localizado[3]. O nome do recurso será apresentado sem a formatação física, para o usuário será apresentado somente o nome que ele mesmo definiu para o recurso, facilitando o entendimento.

---

[2] Suprime-se a formatação do código do usuário, bem como a extensão do arquivo.

[3] Somente para ilustração, isto também poderia ser transparente ao usuário, dando a ele a impressão de que todos os seus recursos estariam disponíveis na estação onde está operando o sistema.

3.4.4 Diagrama de atividade geral do sistema

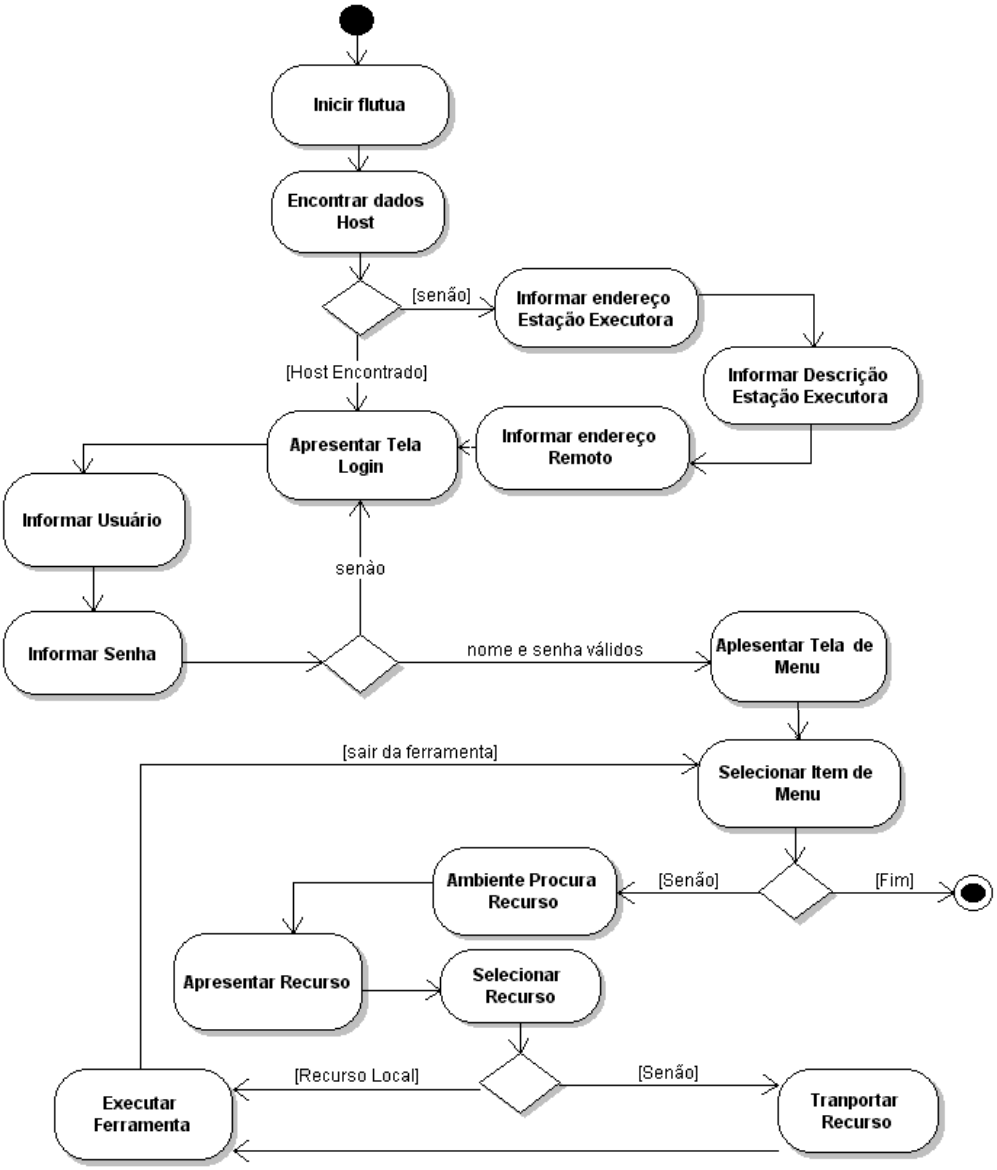


Figura 3.5 - Diagrama de atividades

O diagrama representado na figura 3.5 descreve a seqüência das atividades do sistema proposto desde a execução até o comportamento das rotinas transparentes para o usuário. Para esclarecer melhor o mecanismo que faz com que os recursos do usuário sejam encontrados e transportados, criou-se as figuras 3.6 e 3.7 .

3.4.5 Diagrama de atividade salvar recurso

Figura 3.6 - Diagrama de atividade Salvar Recurso

Na figura 3.6 encontra-se descrito o processo para armazenar recursos criados por usuários. Para composição do nome físico será utilizado protocolo descrito na figura 3.5. Após efetuar a composição do nome, o recurso poderá ser armazenado.

3.4.6 Diagrama de atividades compor lista de recursos

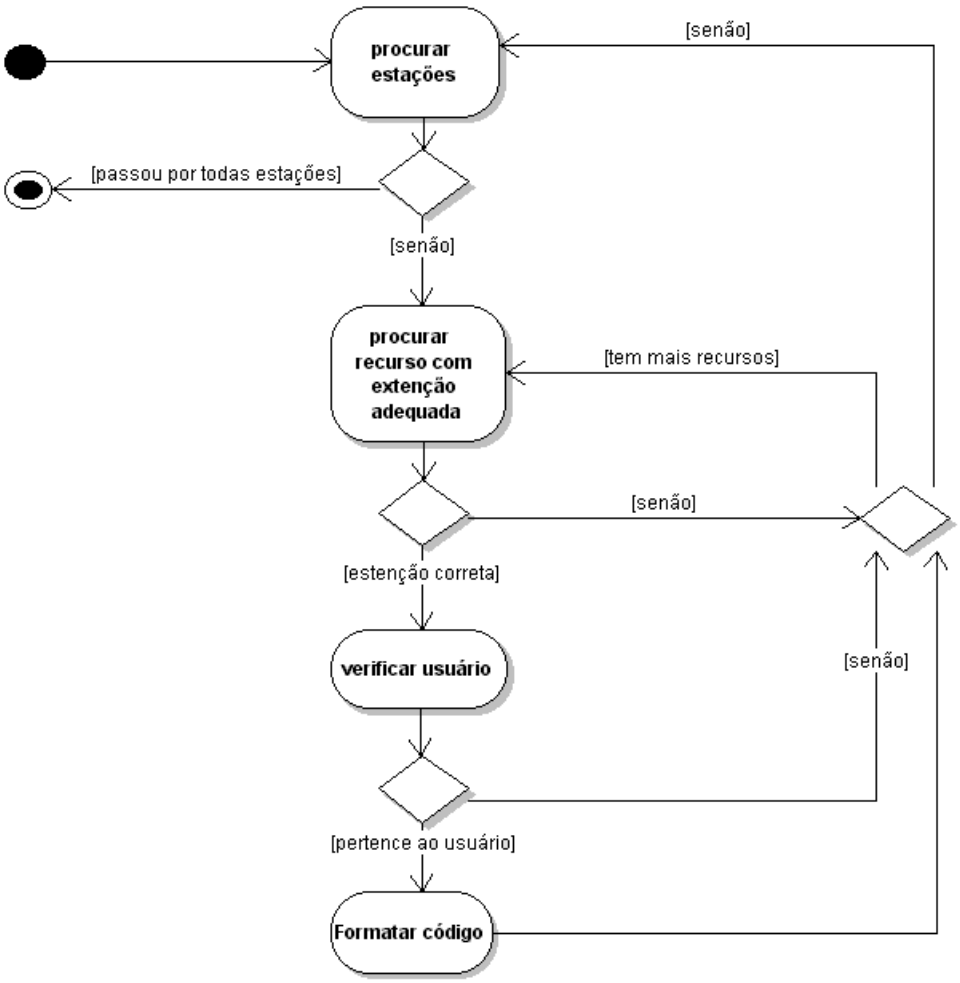


Figura 3.7 - Diagrama de atividade compor lista de recursos

Para editar um recurso, o usuário obrigatoriamente deverá selecionar uma ferramenta. De acordo com a ferramenta selecionada, o sistema fará uma investigação em todo o sistema, procurando encontrar recursos disponíveis do usuário e da ferramenta selecionada. Em cada estação será repetido o processo de procura. Se um arquivo atender os requisitos, seu nome será adicionado a uma lista. O usuário poderá efetuar a seleção de um recurso disponível. Após selecionar um recurso, o sistema executará a ferramenta com o recurso para editá-lo. Este processo está representado na figura 3.7.

3.4.7 Diagramas de estado

Na figura 3.8, encontramos representados os estados do objeto “host”. Este objeto armazena os dados do hospedeiro que compõe o sistema. Para cada nova estação que for registrada, o agente calibra se encarrega de registrar este objeto, bem como o registrará no objeto de “hosts”, que é, na verdade, uma lista de todos as estações que compõem o sistema. Este objeto é de extrema importância para o funcionamento do sistema, pois é através dele que o sistema consegue atingir todas as estações que compõem o sistema.

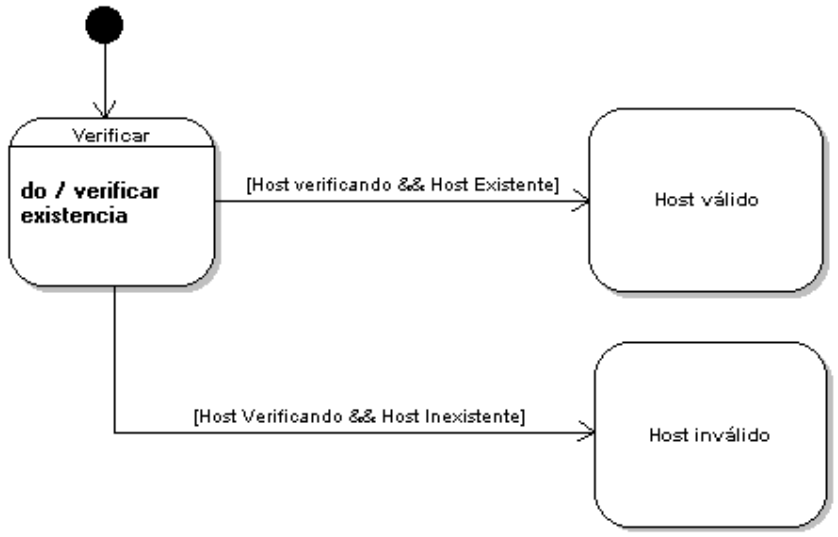


Figura 3.8 - Diagrama de estado do objeto “host”

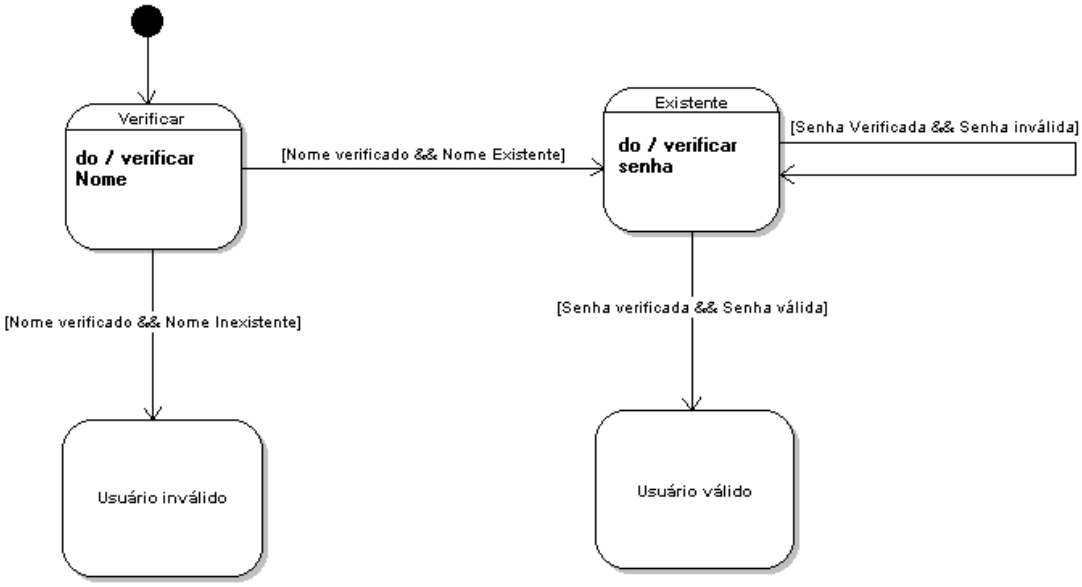


Figura 3.9 - Diagrama de estado do objeto ”pessoas”.

O objeto “pessoas” armazena dados como código, nome e senha dos usuários e está representado na figura 3.9. Cada estação irá possuir este objeto. Para cada novo usuário do sistema, o agente calibra irá registrar seus dados em todas as estações. Os dados armazenados neste objeto irão garantir que um usuário não tenha acesso aos recursos de outro usuário. Como visto na figura 3.4, o código do usuário irá fazer parte da composição do nome físico de um recurso.

Capítulo IV

Projeto e Realização do Sistema Móvel

4.1 Introdução

Neste capítulo estão descritos os resultados alcançados decorrentes da realização do sistema citado no capítulo anterior. Serão apresentadas as estruturas utilizadas para desenvolver este sistema. É válido ressaltar que não há servidores neste tipo de aplicação, sendo que qualquer estação poderá subsidiar outra estação.

Todo o protótipo foi construído sendo utilizado para tal a plataforma JAVA da *Sun MicroSystems*, o que proporciona transparência para as plataformas de *hardware* e *software*.

4.2 Armazenamento de dados

Para armazenar as informações sobre as configurações do ambiente, criou-se uma estrutura em *HashMap* disponibilizada pelo Java. Nestes arquivos, podemos armazenar, excluir, alterar e consultar informações, desde que conheçamos a estrutura dos mesmos.

4.3 Infraestrutura de comunicação

Para infraestrutura de comunicação, utilizou-se RMI descrito no capitulo I. Os três agentes criados para dar suporte à mobilidade foram realizados utilizando RMI.

4.4 Protótipo

Este protótipo foi construído de forma que em sua concepção não existam explicitamente as figuras cliente e servidor. Todas as estações que irão compor este

sistema, de alguma forma ou de outra, representarão os papéis de cliente e servidor. Á medida em que seus usuários forem utilizando este sistema em diferentes localizações, o sistema irá disponibilizar, de forma transparente, todas as ferramentas e recursos. Para que isto seja possível, o usuário, ao iniciar a aplicação em uma nova estação, deverá informar ao sistema o endereço de uma estação em que já utilizou o sistema e também o endereço da estação onde está executando. Com estas informações, o sistema encarrega-se de apresentar ao usuário todos os seus recursos disponíveis. Cabe salientar que estes recursos devem ser padronizados e serão transportados para as estações à medida em que o usuário os selecionar.

4.5 Interface gráfica

A interface gráfica com o usuário facilita a compreensão das necessidades do sistema para com o usuário e vice-versa. A interface construída para este sistema tem por objetivo fazer com que o usuário inicie o processo de utilização do mesmo, tornando transparente em qualquer tipo de plataforma de *hardware* e *software*. A ordenação das interfaces que serão apresentadas a seguir segue a ordem, na qual o sistema evolui.

Para iniciar a utilização do sistema, o usuário deverá iniciar na raíz de seu diretório um arquivo denominado `rmi.bat.` Este arquivo irá efetuar o *rmiregistry* responsável por dar suporte à mobilidade aos agentes. Após iniciar o `rmi.bat,` o usuário poderá iniciar o sistema propriamente dito, bastando, para tal, iniciar de seu diretório o arquivo `flutua.bat.` Este arquivo inicia as variáveis de ambiente e a própria aplicação.

4.5.1 Interface de endereço da estação

Ao iniciar o sistema pela primeira vez em uma determinada estação, será apresentada ao usuário a interface representada pela figura 4.1. Nesta interface, o usuário deverá informar ao sistema o endereço da estação na rede na qual a aplicação está sendo executada. Esta informação será armazenada em um arquivo denominado *host.obj*[4].

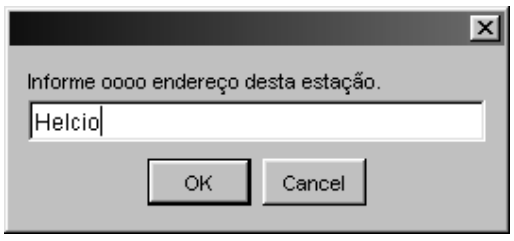


Figura 4.1 - Tela de Endereço da máquina executora

[4] Encontra-se especificado no diagrama de classes;

4.5.2 Interface de descrição da Estação

Após informar os dados da estação executora, o sistema apresentará a interface representada na figura 4.2. Nesta interface, o usuário deverá efetuar uma breve descrição da estação à qual está operando. Esta descrição servirá para facilitar a futura identificação das estações. Esta descrição será também armazenada no objeto *host.obj*.

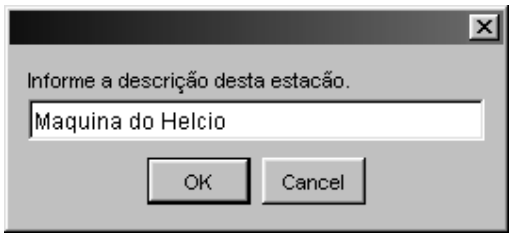


Figura 4.2 - Tela de descrição da máquina executora

4.5.3 Interface de estação de conexão à rede

Dando seqüência, a aplicação apresentará uma interface ao usuário, na qual o mesmo deverá definir uma estação já utilizada e que esteja compondo e sistema. A figura 4.3 representa esta interface. Após informar uma estação da rede, a aplicação se comunicará com a estação indicada, para verificar a conexão com a mesma. Caso não haja comunicação com a rede, a tela representada na figura 4.4 será apresentada para o usuário (caso o mesmo esteja utilizando *Windows*). Este processo será repetido até o momento do usuário entrar com endereço válido, ou então, cancelar a operação, o que impede o usuário de definir um endereço inválido.

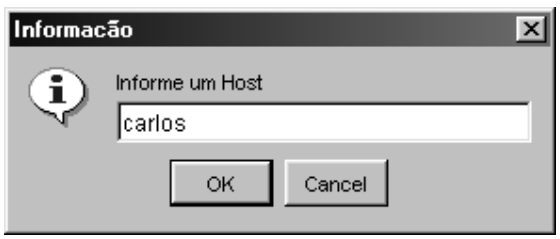


Figura 4.3 - Tela de endereço de estação da rede

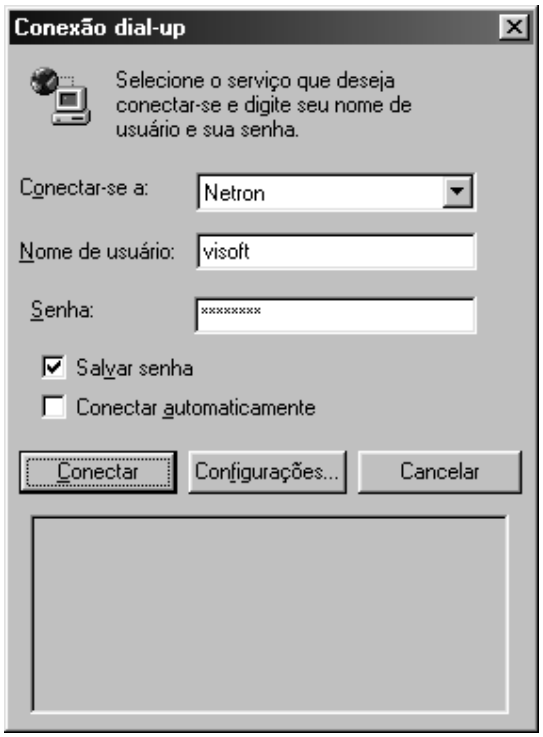


Figura 4.4 - Tela de conexão à rede (Internet)

4.5.4 Interface login do usuário

Após efetuar os processos acima [5], será apresentada ao usuário a interface de *login*, representada pela figura 4.5. Nessa interface, o usuário deverá informar seu nome e senha. Estes dados são vitais para o perfeito funcionamento do sistema, pois para cada usuário será gerado um código, o qual será incorporado aos nomes dos recursos do usuário.

[5] necessários somente na primeira execução em cada estação.

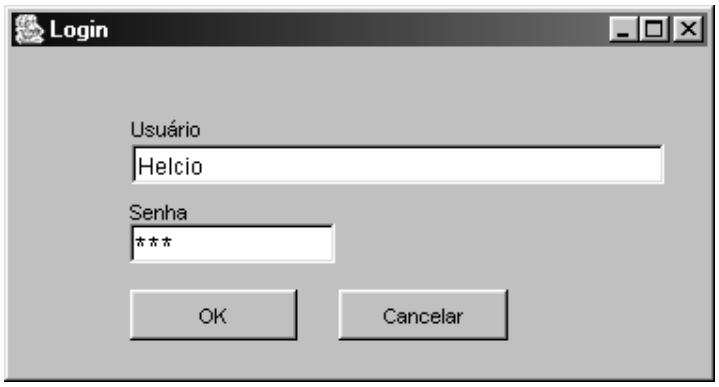


Figura 4.5 - Tela de login de usuário

Para inserir um novo usuário no sistema, basta entrar com nome e senha novos na tela de *login* representada pela figura 4.5. Quando o sistema tentar validar o usuário, e este for um novo, será apresentada ao usuário a interface representada pela figura 4.6, onde o usuário poderá optar por criar novo usuário.

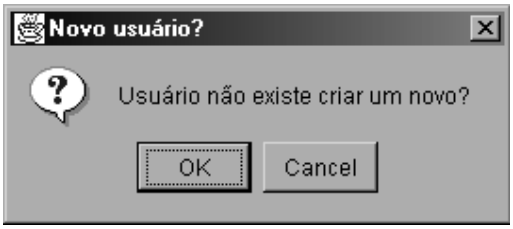


Figura 4.6 - Tela de confirmação de novo usuário

4.5.5 Interface Menu do sistema

Se o usuário for identificado com sucesso, ou um novo usuário for criado, a aplicação apresentará ao usuário uma interface com as ferramentas para ele disponíveis e representados em forma de menu. A figura 4.7 representa esse menu. A partir deste momento, o usuário poderá selecionar a ferramenta que desejar utilizar. Para cada ferramenta selecionada e para cada usuário, será apresentada, antes da ferramenta, uma lista de recursos disponíveis, isto será apresentado com riqueza de detalhes a seguir.

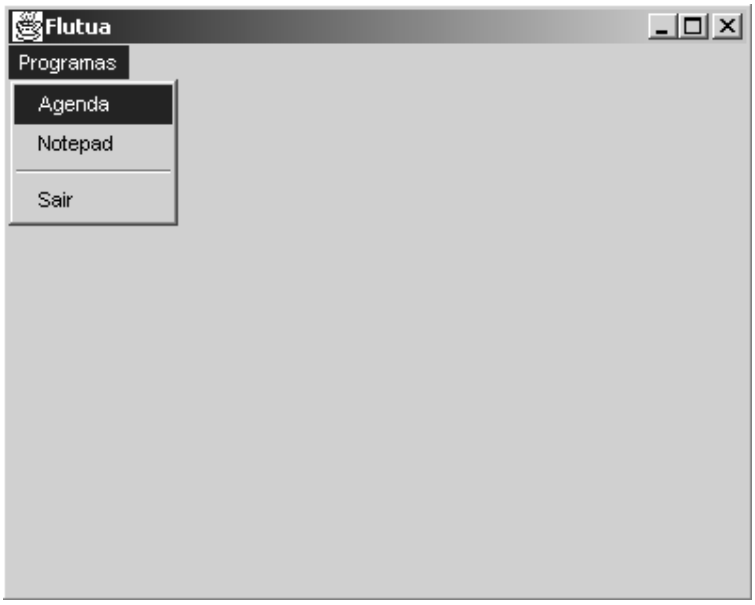


Figura 4.7 - Menu do sistema

4.5.6 Interface da lista de recursos encontrados

Vale a pena ressaltar que as informações acima citadas serão armazenadas e distribuídas entre as estações que compõem este sistema. Para que tal processo aconteça, a aplicação armazenará em cada estação as informações sobre as estações que compõem o ambiente. Ao selecionar uma ferramenta, a aplicação irá disparar um processo de busca para encontrar os recursos disponíveis para o usuário, de acordo com a ferramenta selecionada. O resultado da busca será apresentado conforme figura 4.8. Para editá-lo, o usuário deverá apontá-lo e pressionar o botão <<selecionar>>. O sistema irá transportar o recurso do local onde o mesmo foi encontrado até o local onde foi solicitado. Após os processos de localização e transporte, a aplicação executará a ferramenta selecionada, juntamente com o recurso também selecionado pelo usuário.

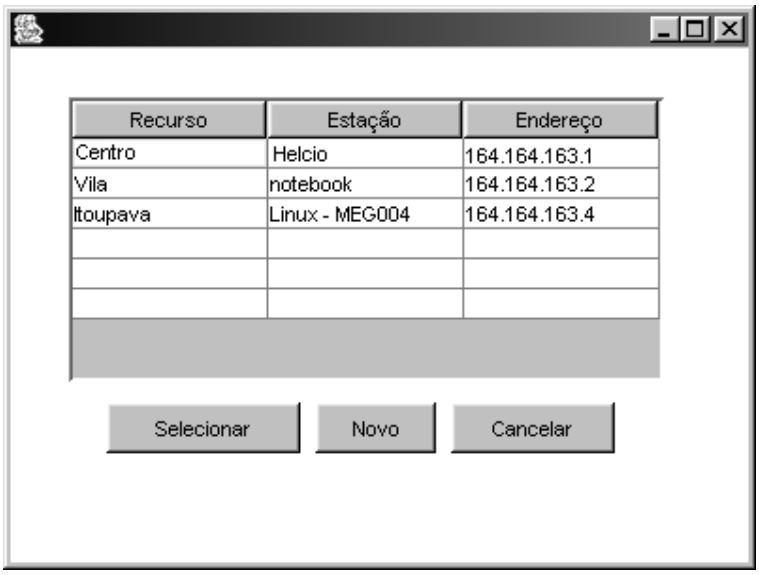


Figura 4.8 - Lista de recursos disponíveis

Para criar um novo recurso, o usuário deve pressionar o botão <<Novo>>. Será apresentada a interface representada na figura 4.9. O usuário deverá definir um nome (descrição) para o novo recurso.

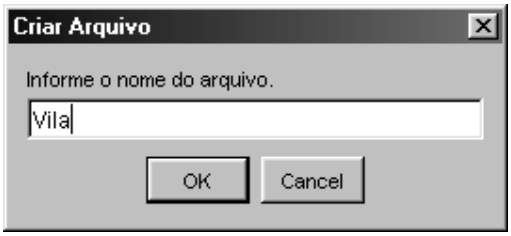


Figura 4.9 - Nomear novo recurso

4.5.7 Agenda

Na figura 4.10, encontramos a interface de uma agenda, na qual o usuário poderá agendar compromissos. Esta aplicação é de extrema simplicidade, pois o intuito deste trabalho é demonstrar o ambiente como um todo e não a implementação de uma agenda. Esta é apenas utilizada como uma ferramenta disponível ao usuário, na qual poderão ser gerados recursos.

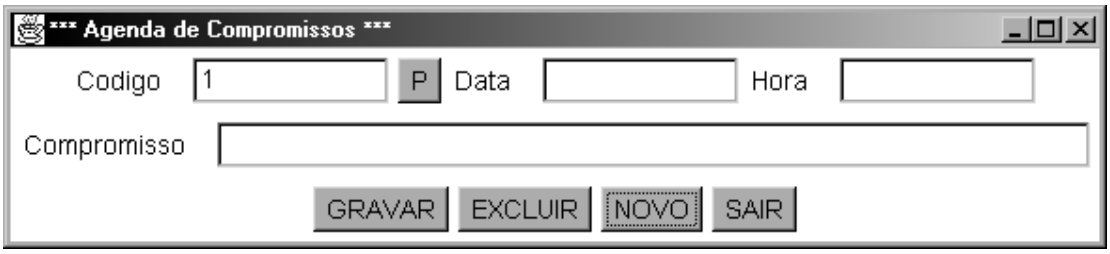


Figura 4.10 - Interface da agenda

4.5.8 Editor

Representada na figura 4.11, temos a interface de um editor de textos, da mesma forma como a agenda. O editor é também apenas uma ferramenta para o sistema.

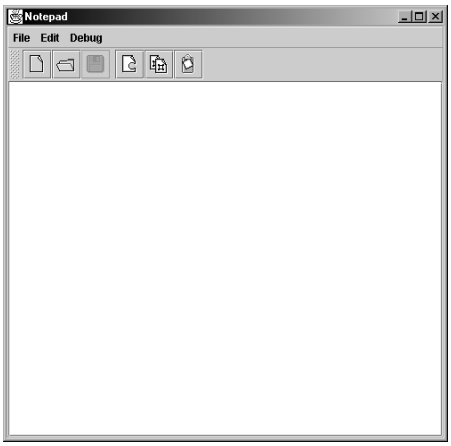


Figura 4.11 - Interface editor de textos

4.6 Diagrama de classes

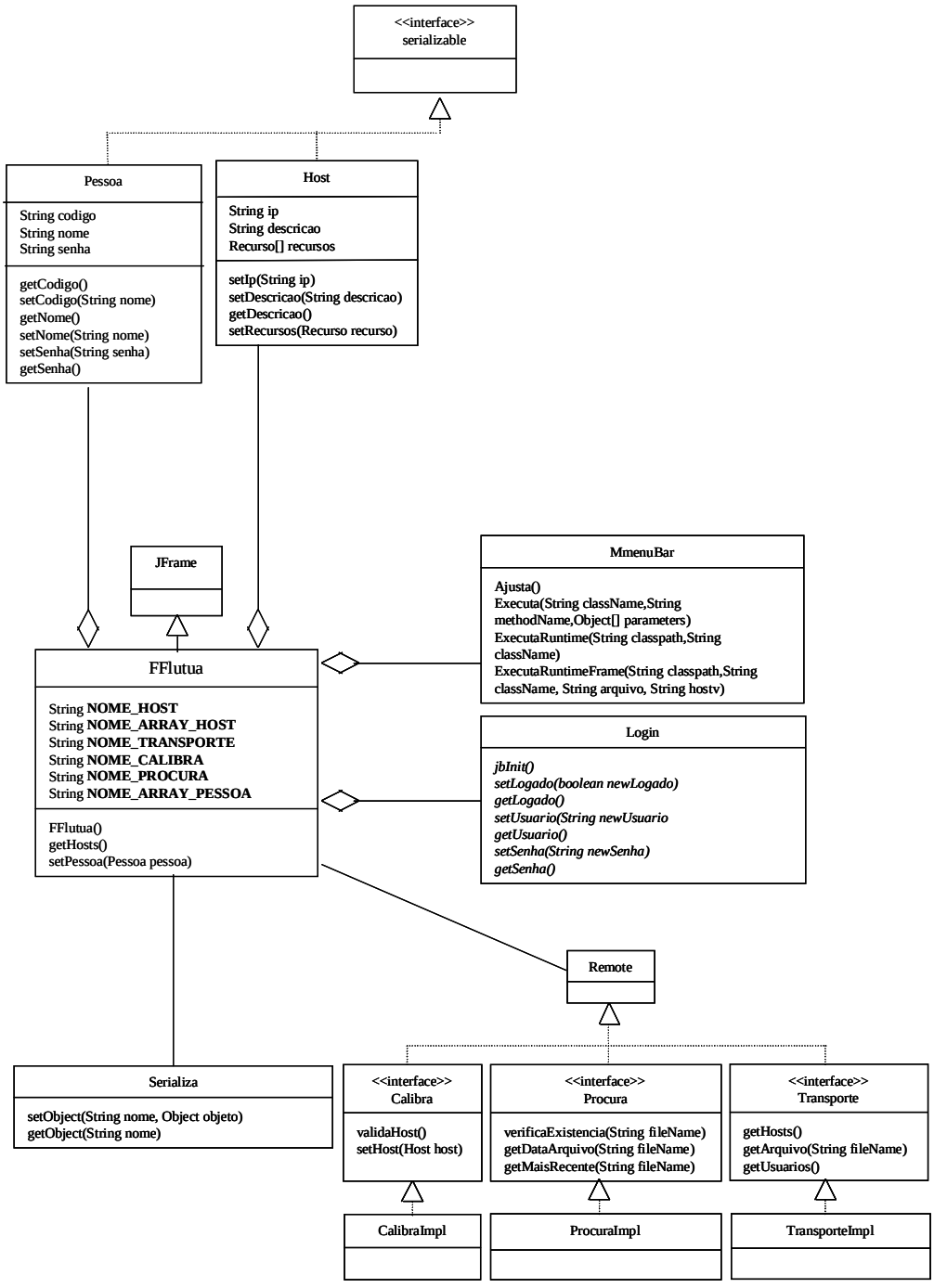


Figura 4.12 - Diagrama de classes (implementação)

Representado na figura 4.12, temos o diagrama de classes que compõe o sistema proposto. Este sistema foi construído em três camadas (*Model-View-Controler Pattern* "MVC") [JAV01].

Os detalhamentos das classes do diagrama serão especificados com os diagramas de seqüência seguintes. À descrição completa das classes, métodos e atributos poderão ser encontrados no anexo 01.

4.6.1 Descrição das Classes

Host -	Serve para armazenar as informações sobre os Hospedeiros que compõem o ambiente.
Pessoa -	Armazenará as informações referentes a usuários.

FLogin -	Este Frame é uma tela de login padrão.
Fflutua -	Esta classe é um frame, no qual será montado toda a base da aplicação.
MMenuBar -	Menu que será utilizado no Frame Fflutua.

Calibra -	Interface que pré-define os métodos que serão implementados pelas classes. Estende a classe Remote, pois assim terá suporte a RMI.
CalibraImp -	Classe que implementa a interface Calibra e estende a classe UnicastRemoteObject para ter suporte a RMI
Procura -	Interface que pré-define os métodos que serão implementados posteriormente pela classe *procuraImp*. Estende a interface Remote para suportar RMI.
ProcuraImp -	Esta classe implementa a interface *procura* e estende a classe UnicastRemoteObject para ter o suporta a RMI.
Transporte -	Contém os métodos que deverão ser implementados para utilizar o transporte de arquivos, estende a Interface Remote para suportar RMI.
TransporteImp -	Esta classe implementa a interface *Transporte* e estende a classe UnicastRemoteObject para ter o suporta a RMI.
Serializa -	Esta é uma classe genérica que "Grava" e "Recupera"

4.7 Diagramas de seqüência

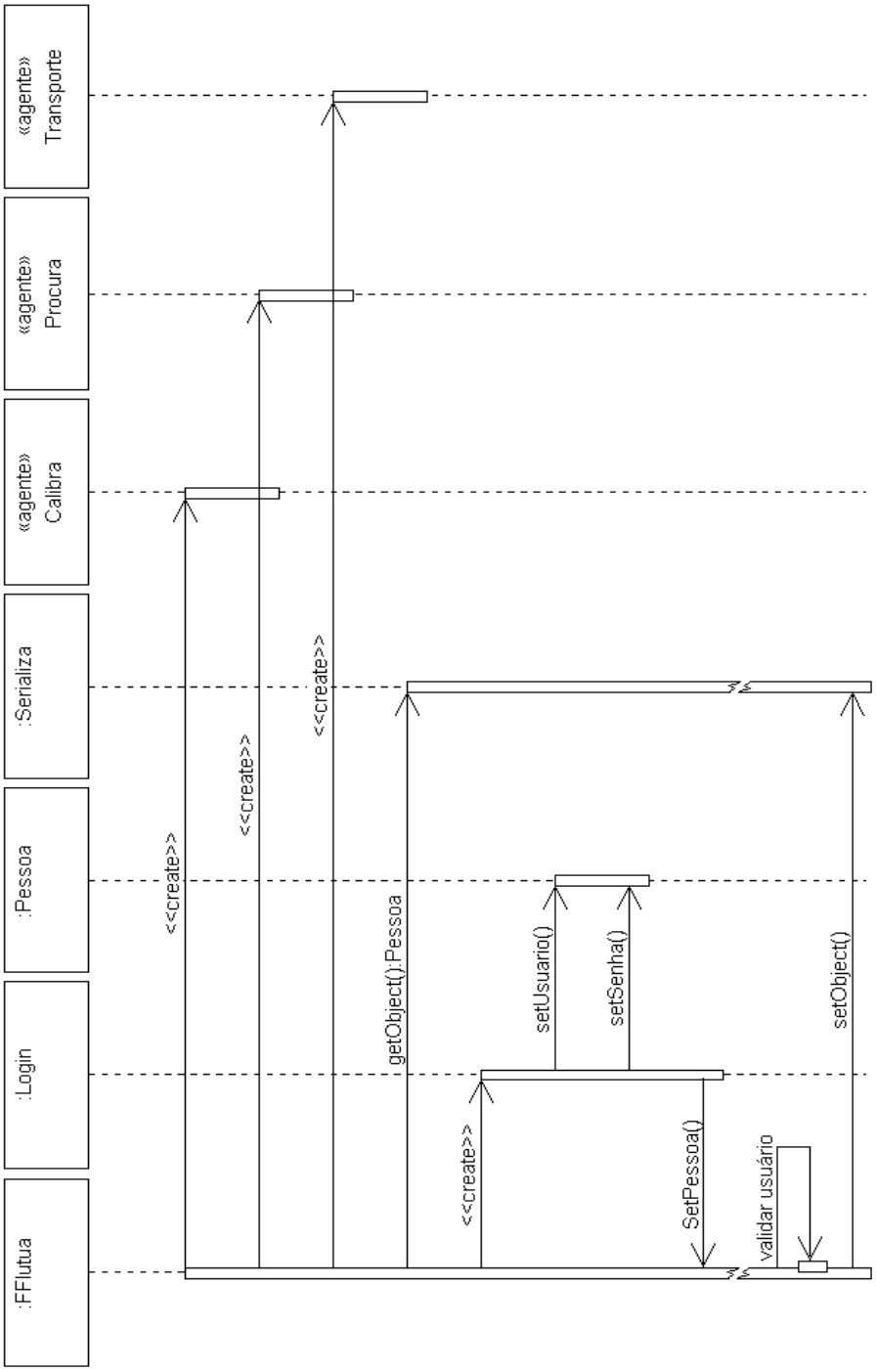


Figura 4.13 - Diagrama de seqüência efetuar *login*

O processo do *login* encontra-se representado na figura 4.13. Este processo é muito importante para este sistema, pois é através dele que conseguimos configurar e incorporar cada estação no sistema proposto. Ao iniciar o sistema, os três agentes utilizados serão criados. O método getObject() retornará uma lista de usuários existentes no objeto “Pessoa” com o intuito de comparar o conteúdo deste objeto com os dados digitados pelo usuário no processo validação de usuário. O sistema cria um objeto denominado “Login”. O *login* é interface, na qual o usuário entrará com seu nome e senha. Através do método setUsuário() setSenha() estas informações serão armazenadas no objeto “Pessoa”. O método setPessoa() irá verificar se este usuário já utilizou esta estação. Caso se tratar de um novo usuário, o sistema acionará o agente calibra descrito abaixo. Ao contrário, se for um usuário existente, o sistema irá validar sua senha, se esta não conferir, o usuário terá que entrar novamente com a senha. O método setObject() salva (serializa) um objeto. Os métodos getObject() e setObject serão utilizados para salvar e recuperar objetos no decorrer da aplicação. A interrupção nas linhas de vida das classes Fflutua e Serializa indica que outros processos ocorrerão.

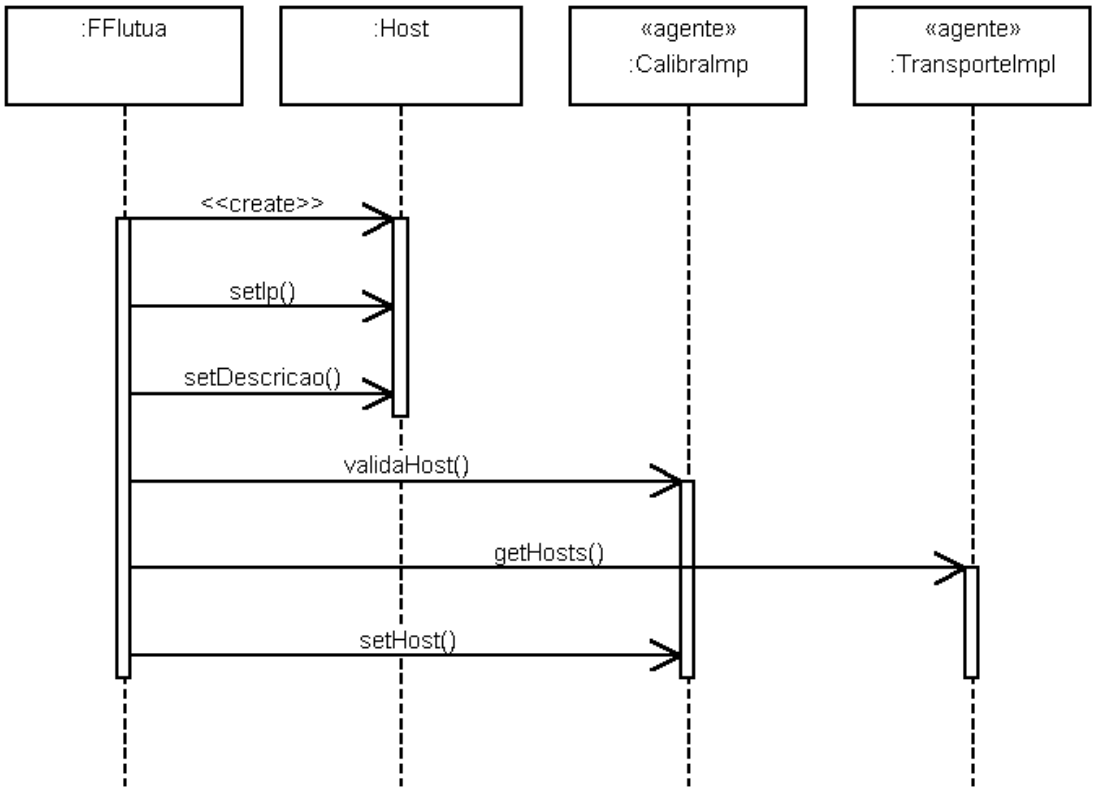


Figura 4.14 - Diagrama de seqüência ajustar aplicação

Na figura 4.14, temos representado o processo de inclusão de uma nova estação no sistema. Ao constatar a ausência do objeto '*Host'*, trata-se de uma nova estação que ainda não fazia parte do sistema e que necessita ser criada. Será solicitado ao usuário o endereço da estação, na qual o usuário está executando. Esse endereço será armazenado através do método setIp(). O mesmo acontece com a descrição, o usuário informa a descrição e o método setDescricao() o armazena. Será também solicitado ao usuário e endereço de uma estação já utilizada pelo sistema. As informações serão armazenadas no objeto "Host". Com os endereços coletados, será acionado o método validaHost(), este verifica se o endereço informado pelo usuário é valido. Se o Endereço for inválido, o sistema volta a solicitá-lo para o usuário. Se o endereço for válido, o sistema através do método getHost(), recupera a lista de hospedeiros que compõem o sistema. Através do método setHost(), o sistema inclui em todos os hospedeiros o novo endereço.

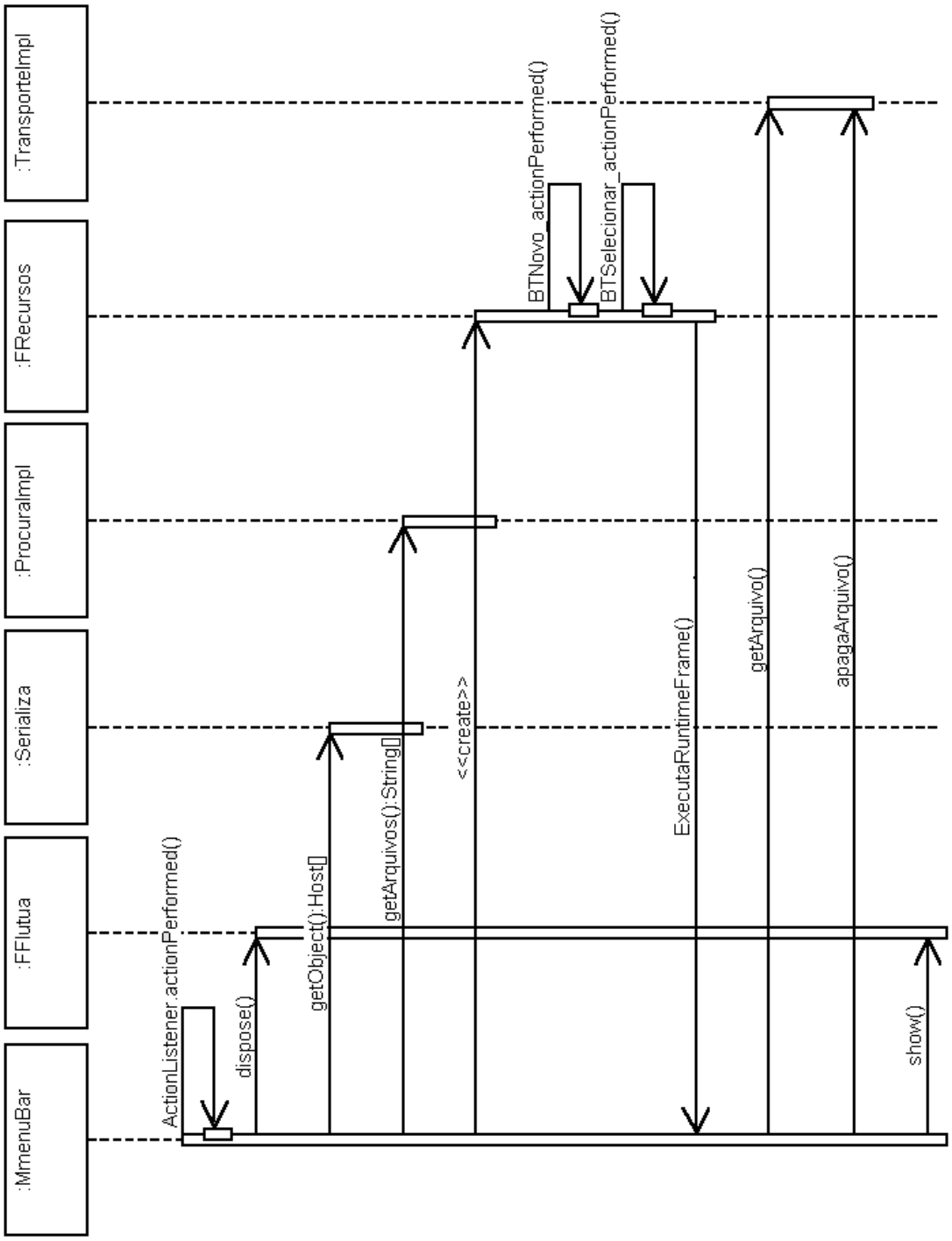


Figura 4.15 - Diagrama de seqüência executar ferramenta

Encontra-se representado na figura 4.15 o mecanismo de execução das ferramentas disponíveis para o usuário. A classe MmenuBar, através do método executa(), irá executar a ferramenta selecionada pelo usuário. Sempre após o usuário fechar uma ferramenta aberta, o sistema voltará para o menu principal do sistema, de onde poderá selecionar ferramentas disponíveis ou sair do sistema.

A captura do evento Clique de mouse é efetuada através do evento ActionListener.actionPerformed(), para disparar as atividades relacionadas a este item de menu. Quando for selecionado um item de menu, a tela de menu será ocultada, isto é, realizado pelo método Dispose().

Para gerar a lista de recursos disponíveis para o usuário, necessita-se conhecer todas as estações que compõem o sistema, isto é, responsabilidade do método getObject():Host[], que os recupera em forma de array (lista). Encontram-se, nesta lista, os endereços de todas as estações que compõem o sistema. Conhecendo todas as estações, podemos recuperar os recursos nelas armazenados. O método getArquivos():String[] nos promove tal atividade dentro de uma estação. Vale a pena lembrar que poderá existir um número indeterminado de estações e esta operação será efetuada em cada uma.

O sistema criará um objeto do tipo interface gráfica <<create>>, na qual será apresentada a lista de recursos do usuário. Nesta lista, os nomes dos recursos já estarão formatados, omitindo o código do usuário e a extensão da ferramenta que o gerou, respeitando as regras de composição de nome de recurso.

Para que o usuário tenha a possibilidade de criar um novo recurso, o BTNovo_actionPerformed() representa a possibilidade do usuário criar um novo documento. Bastando pata tanto nomear o recurso.

Com a lista gerada, o usuário poderá selecionar um recurso disponível e, então, executar a ferramenta, A ação BTSelecionar_actionPerformed() invoca o método ExecutaRuntimeFrame do MmenuBar para executar a ferramenta com seu recurso selecionado.

Se acaso o endereço do arquivo for diferente do endereço da estação na qual esta sendo executado, o método getArquivo() será executado. Este irá transportar o arquivo da origem para o destino. Quando o agente terminar de copiar o arquivo, o mesmo

através do método apagaArquivo(), irá apagá-lo na origem, isto serve para evitar redundância de dados.

Quando o usuário sair da ferramenta, o método show() será invocado. Este método voltará a apresentar a tela de menu do sistema.

4.8 Aplicação prática do sistema

Esta seção tem por objetivo demonstrar, na prática, o ambiente de *hardware* e *software* utilizados para realização deste trabalho, com o intuito de validar as funcionalidades propostas e construir o sistema ora proposto. Vale a pena ressaltar que este ambiente foi testado em dois laboratórios, entretanto, para facilitar a compreensão dos leitores, montou-se à figura 4.16, que é uma representação de todas as estações e suas diferentes plataformas de *hardware* e *software* utilizados.

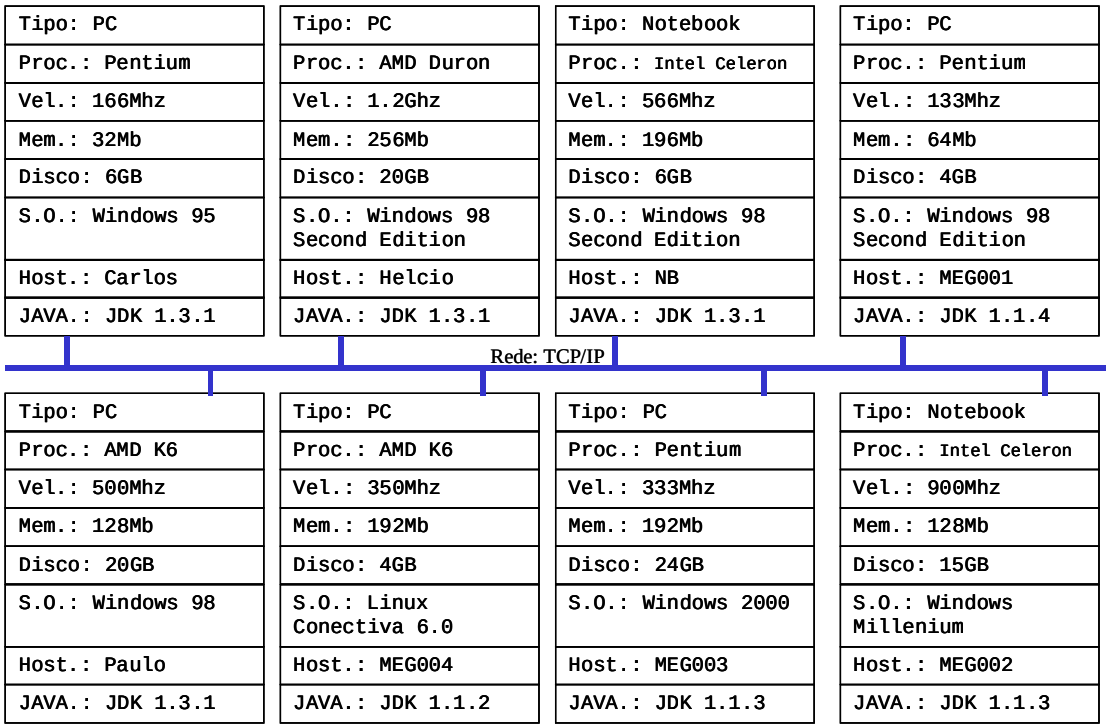


Figura 4.16 - Laboratório de testes

Utilizou-se o protocolo de rede TCP/IP, que atualmente é o protocolo de rede mais utilizado no mercado e também o protocolo utilizado na Internet. Outros protocolos como NetBEUI, IPX/SPX poderiam ser utilizados, o problema é que não é

possível configurar o ambiente operacional Linux com NetBEUI, pois este é um protocolo proprietário[6] *Microsoft*.

4.8.1 Problemas enfrentados com diferentes ambientes operacionais

Para construir este sistema, começamos utilizando *Microsoft Windows* 95 como plataforma operacional, fomos evoluindo nas versões da *Microsoft Windows*, sem, aparentemente, nenhum problema. Nesta evolução chegamos ao ambiente operacional Linux, onde encontramos uma dificuldade: os separadores do *classpaht* são diferentes entre os ambientes operacionais. Para configurar o *classpath* do windows (todas as versão utilizadas) utiliza-se ";" como separadores, já no Linux utiliza-se ":", o que foi mera falta de intimidade com o ambiente Linux. A mesma falta de conhecimento da ferramenta tornou necessária a utilização de FTP para instalar o protótipo e atualizar as novas versões do mesmo. Após conseguirmos executar o sistema no ambiente Linux, constatamos que a interface gráfica tem algumas diferenças, provavelmente a diferença está na resolução do monitor de vídeo.

No estabelecimento da porta para os servidores, optou-se pela porta padrão do RMI "1099". Não há necessidade de trabalharmos com diferentes portas, por tratar-se de um sistema homogêneo. Podemos preservar o usuário para que não haja necessidade desta informação nas instalações.

[6] Executa somente em ambiente para o qual foi desenvolvido.

Capítulo V

Conclusão e Trabalhos Futuros

O estudo desenvolvido nessa dissertação abordou a mobilidade de usuário e a de sistemas de computação. Mobilidade torna-se um problema cada vez maior quando temos a necessidade de localizar algo que deixamos em algum lugar. Muitas alternativas foram criadas. Os endereços, no caso dos correios, e-mail (endereço eletrônico), telefone celular.

Neste trabalho foram abordadas mais especificamente soluções de mobilidade para aplicações de computadores. Problemas como migrar o ambiente de trabalho para onde seu usuário se encontra foram resolvidos, desde que exista uma infra-estrutura mínima para tal. Além da infraestrutura vale a pena ressaltar que para um perfeito funcionamento do sistema todos os equipamentos devem estar sempre ligados e conectados à rede. Pois para montar a lista de recursos disponíveis para o usuário, o sistema dispara um agente que visita todas as estações que compõe seu ambiente.

Através da elaboração de um cenário, foi possível testar a aplicação em diferentes plataformas de *hardware* e *software*. Através deste processo, evidenciou-se a integração do sistema com os ambientes de trabalho. O comportamento do sistema mostrou-se eficaz nos ambientes em que foi testado, Windows e Linux, sendo que, neste último, a interface gráfica não se adaptou corretamente.

Para futuros trabalhos sugere-se os seguintes pontos que foram verificados ao longo de nossa pesquisa:

- Criptografia

Ao transportar qualquer tipo de recurso, deverão ser levados em consideração os aspectos de segurança. Como neste protótipo utilizou-se *hashmap* (para armazenar os recursos) poderia criar-se algum mecanismo de criptografia ao transportar o recurso;

- Compactação

À medida que os recursos aumentam de tamanho, provavelmente deverão apresentar problemas de desempenho, o que faria com que o usuário ficasse esperando pelo transporte de seu recurso.

- Equipamentos desligados ou não conectados

Quando o usuário invocar seu sistema e uma das estações estiver desligada, seu conteúdo não poderá ser encontrado. Poderão ser criados mecanismos de redundâncias.

- Redes sem fio

Utilização do ambiente em redes sem fio e um estado de latência e largura de banda visando uma melhor eficiência para usuário móvel.

- Outros sistemas de agentes móveis

Para a elaboração deste trabalho utilizou-se RMI para dar suporte a mobilidade dos agentes. Encontram-se citados neste trabalho diversos sistemas de agentes que poderiam ser utilizados.

Referências

[ANA00] ANASTASI, Giusepe, Aurelio La Corte, Antonio Puliafito e Orazio Tomarchio. **An agent-based approach for QoS provisioning to mobile users in the internet.** Università di Pisa. Pisa/Università diCatania.Catania/Università diMessina. Messina. Italia

[BHA96] BHAGWAT, Pravin, Charles Perkins e Satish Tripathi. **Network Layer Mobility: Na Architecture and Survey.** IEEE Personal Communications, 1996

[CAS00] Castiglia, Marcelo Pereira e Geyer, Cláudio. **Mini curso para a disciplina de Programação Distribuída e Paralela.** Instituto de Informática UFRGS 200

http://www.inf.ufrgs.br/procpar/disc/inf01008/trabalhos/sem00-1/PazziniT2ApostilaRMI.zip/indice.htm

Ultimo acesso: 09/08/2002

[CHE02] CHENG, Shang-Wen, David Garlan, Bradley Schmerl, João Pedro Souza, Bridget Spitznagel, Peter Steenkiste e Ningning Hu. **Software Architecture-based Adaptation for Pervasive Systems.** Carnegie Mellon University, Pitsburgh – USA

[COR99] CORTE, Aurélio La, Antonio Puliafito e Orazio Tomarchio. **An Agent based framework for móbile users.** Università di Catania. Catania. Italia

[DEI01] DEITEL, H. M. , DEITEL P.J. **JAVA Como Programar.** Porto Alegre: Bookman, 2001.

[DUA00] Duarte, Otto Carlos Muniz Bandeira Duarte e Vicente Barreto Domingues ***RMI (Remote Method Invocation).*** Universidade Federal do Rio de Janeiro, 2000

http://www.gta.ufrj.br/grad/00_1/vicente/

Ultimo acesso: 09/08/2002

[FOW02] FOWLER, Martin, SCOTT, Kendall. **UML Essencial.** Porto Alegre: Bookman, 2002.

[GRA00] GRAY, Robert, David Kotz, George Cylenko e Daniela Rus. **Mobili Agents**: **Motivations and state-of-the-art systems.** Dartmouth College, Hanover 2000

[HER99] HERLIHY, Maurici e Michael P. Warres. **A Table of Directories: Implementing Distributed Shared Object in Java.** Brown University/Sun Microsystems San Francisco – California 1999

[HÜB99] Hübner , Jomi Fred. **Comunicação entre Objetos Distribuídos com RMI**.

http://www.inf.furb.br/~jomi/java/apresentacoes/rmi/

Ultimo acesso: 16/08/2002

[JAV99] **Java™ Remote Method Invocation (RMI) Tutorial RMI da Sun Microsystems.** 1999

http://java.sun.com/j2se/1.3/docs/guide/rmi/index.html Sun Microsystems, Inc.

Ultimo acesso: 13/06/2002

[JAV01] **Object-Oriented Application Analisys Designfor Java™ Tecnologi (UML).** Sun Microsystems,Inc. Colorado. 2001.

[KON01] Kon, **Sistemas Operacionais Distribuídos** Universidade de São Paulo, 2001

http://www.ime.usp.br/~kon/MAC5755/aulas/Aula8.html

Ultimo acesso: 15/07/2002

[NAS00] NASSU, Eugênio Akihito. **O Significado de "Aqui" em Sistemas Transacionais Móveis.** IME-USP. São Paulo, Brasil

[NAU96] NAUGHTON, Patric, Dominando JAVA. São Paulo Makron Books 1996.

[RHO00] RHODES, Bradley J. , Nelson Minar e Josh Weaver. **Wearable Computing Meets Ubiquitous Computing: Reaping the best of both words**. MIT Media Lab. 20 Ames St. Cambridge MA

[ROC02] ROCHA, Helder **Fundamentos de Objetos Remotos.**

http://www.organisis.com.br

Ultimo acesso: 16/04/2002

[SCH99] SCHEIDER, Geri, Jason P. Winters. **APPLYIG USES CASES: A practical guide.** IEEE Personal Communications, 1996

[SOU97] SOUZA, Vandemberg Dantas de; **Desenvolvendo Applets com Java**. Campus. Rio de Janeiro : 1997.

[TAL95] TALUKDAR, Anup K. **MRSVP: Reservation Protocol for an Integrated Services Packet Network with Móbile Hosts.** C&C-C Research NEC USU, 1995

[WOL02] WOLLRAHTH, Ann e JimWaldo. **The Java Remote Method Invocation**

http://java.sun.com/docs/books/tutorial/rmi/

Ultimo acesso: 18/08/2002

[WOO96] WOOLDRIDGE, Michael J.; JENNINGS, Nicholas R.; MÜLLER, Jörj P. **Intelligent Agents III.** Springer. New York : 1996.

ANEXO 01

Especificação das classes com seus atributos e métodos

Classe *FLogin* - Este Frame é uma tela de login padrão.

Atributos

Métodos *jbInit()* - monta a parte gráfica bem como seus eventos.

setLogado(boolean newLogado) - indica que o usuário entrou com informações de login corretas.

getLogado() - Este método serve para verificar se o usuário já está logado.

BTCancelar_actionPerformed(ActionEvent e) - finaliza a tela de login, conseqüentemente a aplicação. (Evento do botão Cancelar).

this_windowClosing(WindowEvent e) - captura quando o "X" (Fechar) do Frame é pressionado.Implementa a finalização da aplicação.

setUsuario(String newUsuario) - seta o nome na tela.

getUsuario() - recupera o nome do usuário digitado na tela.

setSenha(String newSenha) - seta a senha na tela.

getSenha() - recupera a senha da digitada tela.

BTOK_actionPerformed(ActionEvent e) - implementação do Botão "OK", Nele é feito o teste de preenchimento dos campos Usuário e Senha.

Classe *Fflutua* - Esta classe é um frame no qual será montada toda a base da aplicação.

Atributos *Host host* - informações da estação onde está sendo utilizada.

ArrayList hosts - Este ArrayList conterá todos os hosts envolvidos na mesma rede.

ArrayList pessoas - Este ArrayList conterá as pessoas que já utilizaram a respectiva estação.

String NOME_HOST="host.obj" - nome pelo qual será armazenado o objeto host.

String NOME_ARRAY_HOST="hosts.obj" - nome pelo qual será armazenado o objeto hosts.

String NOME_TRANSPORTE="Transporte" - nome pelo qual será acessado o agente de Transporte.

String NOME_CALIBRA="Calibra" - nome pelo qual será acessado o agente Calibra.

String NOME_PROCURA="Procura" - nome pelo qual será acessado o agente de Busca.

String NOME_ARRAY_PESSOA="Pessoas.obj" - nome com o qual será armazenado o objeto pessoas.

Métodos *Fflutua()* - Este construtor é responsável pela implementação dos controles do agentes Calibra, Busca e Transporte.

getHosts() - Este método recupera todos os hosts existentes na rede, para isto utiliza-se a o agente de transporte, em seguida utiliza o agente Calibra para atualizar as demais informações.

setPessoa(Pessoa pessoa) - seta um usuário e ao mesmo tempo valida com o usuário e senha informados na tela de login.Caso seja um usuário novo este recupera todos os usuários existentes no HD e adiciona este novo na lista dos existentes.

Classe *MMenuBar* - menu que será utilizado no Frame Fflutua.

Atributos

Métodos *Ajusta()* – monta o menu com seus itens baseado nos recursos disponibilizados. Seta o claspath, monta os eventos de chamada para cada item de Menu Através do agente Procura. E com agente Transporte ele transporta os arquivos necessários para o perfeito funcionamento do Aplicativo.

Executa(String className,String methodName,Object[] parameters) - Este método utiliza de reflexão para executar um determinado método em uma determinada classe.

ExecutaRuntime(String classpath,String className) - utiliza o Runtime da máquina virtual Java para executar classes ou processos (Executáveis).

Classe *Host* - serve para armazenar as informações sobre os Hosts que compõe o ambiente.

Atributos *String ip* - Armazena o IP do Host.

String descrição - Armazena a Descrição do Host.

Recurso[] recursos - Armazena os recursos disponibilizados para o Host.

Métodos *setIp(String ip)* - armazena o IP.

getIp() - recupera o IP.

setDescricao(String descricao) - armazena a Descrição.

getDescricao() - recupera a Descrição.

setRecursos(Recurso recurso) - armazena os recursos que estarão disponíveis no determinado Host.

getRecursos() - recupera os recursos disponíveis nesta estação.

Classe *Pessoa* - Esta classe armazenará os dados referentes à pessoa..

Atributos *String nome* - Armazena o nome do usuário.

String senha – Armazena a Senha do usuário.

Métodos *setNome(String nome)* - armazena o nome.

getNome() - recupera o nome.

setSenha(String senha) - armazena a senha.

getSenha() - recupera Senha.

Classe *Recurso* - Mantém as informações referentes aos recursos disponibilizados para o usuário.

Atributos *String nomeArquivo* - Nome do Arquivo.

String nomeMenu - nome que será armazenado no menu.

String nomeClasse - nome físico da classe.

String nomeMetodo - nome do método que deverá ser executado.

String extencaoDadosDependentes - de onde deverão ser extraídos os dados.

String[] arquivosDependentes - arquivos que compõe o recurso.

TipoRecurso tipoRecurso - indida qual o tipo de recurso que este arquivo representa.

String[] parametros - parâmetros necessários.

Métodos

setNomeArquivo(String nomeArquivo) - armazena o nome do arquivo.

getNomeArquivo() - recupera o nome do arquivo.

setNomeMenu(String nomeMenu) - armazena o nome do menu.

getNomeMenu() - recupera o nome do menu.

getNomeClasse() - recupera o nome da classe.

setNomeClasse(String newNomeClasse) - armazena o nome da classe.

getNomeMetodo() - recupera o nome do método que será invocado por reflexão.

setNomeMetodo(String newNomeMetodo) - armazena o nome do método que será invocado por reflexão.

getExtencaoDadosDependentes() - recupera a descrição da extensão do arquivo onde estarão os dados complementares(por exemplo um biblioteca de classes).

setExtencaoDadosDependentes(String newExtDaDep) - armazena a descrição da extensão do arquivo onde estarão os dados complementares(por exemplo um biblioteca de classes).

getTipoRecurso() - recupera o tipo de recurso.

setTipoRecurso(TipoRecurso newTipoRecurso) - armazena o tipo de recurso.

getArquivosDependentes() - recupera o nome dos arquivos que o recurso dependente para executar.

setArquivosDependentes(String[] newArquivosDependentes) - armazena o nome dos arquivos que o recurso dependente para executar.

getParametros() - recupera os parâmetros.

setParametros(String[] newParametros) - armazena os parâmetros.

Classe *TipoRecurso* - Mantém as informações referentes aos tipos de recursos utilizados pelo sistema.
Ex:
1 – Complementar.
2 – Pacote para executar.
3 – Pacote Complementar.
4 – diversos.

Atributos *int codigo* - Armazena o código do tipo de recurso.
String descricao - Armazena o Descrição do tipo de recurso.

Métodos TipoRecurso() - Construtor default implementa a inicialização da classe com o tipo de recurso.
TipoRecurso(int codigo, String descricao) - Construtor que inicia a classe iniciando as propriedades código e descrição como valor passado por parâmetro.
getCodigo() - recupera o código.
setCodigo(int newCodigo) - armazena o código.
getDescricao() - recupera a descrição.
setDescricao(String newDescricao) - armazena a descrição.
getTipoRecurso() - retorna um array com os tipos de recursos existentes.

Classe *Calibra* - Interface que pré-define os métodos que serão implementados pelas classes Estende a classe Remote pois assim terá suporte a RMI.

Atributos

Métodos *validaHost()* - validará se um host esta ou não ativo/encontrado.
setHost(Host host) – armazena um host.

Classe *CalibraImp* - Classe que implementa a interface Calibra e estende a classe UnicastRemoteObject para ter o suporta a RMI

Atributos

Métodos

Classe *Procura* - Interface que pré-define os métodos que serão implementados posteriormente. Estende a interface Remote para suportar RMI.

Atributos

Métodos *verificaExistencia(String fileName)* - verifica se um determinado arquivo existe em um determinado host.
getDataArquivo(String fileName) - recupera a data de um determinado arquivo no host.
getMaisRecente(String fileName) - retorna o Host que contém o arquivo mais atual.

Classe *ProcuraImp* - Esta classe implementa a interface Procura e estende a classe UnicastRemoteObject para ter o suporta a RMI.

Atributos

Métodos

Classe *Transporte* - Esta interface contém os métodos que deverão ser implementados para utilizar o Transporte de arquivos. estende a Interface Remote para suportar RMI.

Atributos

Métodos *getHosts()* - retorna os hosts que fazem parte da rede.
getArquivo(String fileName) - retorna um array de bytes(arquivo) de um host Remoto.
getUsuarios() - retorna um ArrayList de usuários que utilizaram determinado host.

Classe *TransporteImp* - Esta classe implementa a interface Transporte e estende a classe UnicastRemoteObject para ter o suporta a RMI.

Atributos

Métodos

Classe	*Serializa*	Esta é uma classe genérica que "Grava" e "Recupera"
Atributos	-	
Métodos	*setObject(String nome, Object objeto)* - grava um objeto no HD, recebe como parâmetro o nome com o qual será armazenado e o objeto em si. Para o objeto ser armazenado deve implementar a interface *Serializable*. *getObject(String nome)* - recupera o objeto gravado no HD, recebe como parâmetro o nome com o qual o objeto foi armazenado e retorna um *Object* . Para poder utiliza-lo é necessário converte-lo para a classe original através do "*Casting*"	