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

Anubis Graciela de Moraes Rossetto

Uma Abordagem para Tratamento da Desconexão de Dispositivos Móveis na Utilização de Recursos de Grid Computacional

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

Orientador: Mário Antônio Ribeiro Dantas, Dr.

Florianópolis, junho de 2007.

Uma Abordagem para Tratamento da Desconexão de Dispositivos Móveis na Utilização de Recursos de Grid Computacional

Anubis Graciela de Moraes Rossetto

Esta Dissertação foi julgada adequada para a obtenção do título de Mestre em Ciência da Computação Área de Concentração de Sistemas de Computação e aprovada em sua forma final pelo Programa de Pós-Graduação em Ciência da Computação.

_____

Rogério Cid Bastos, Dr. (coordenador do curso)

Banca Examinadora:

_____

Mário Antônio Ribeiro Dantas, Dr. (orientador)

_____

Claudio Fernando Resin Geyer, Dr.

_____

Ricardo José Rabelo, Dr.

_____

Roberto Willrich, Dr.

"Só sabemos com exatidão quando sabemos pouco;
à medida que vamos adquirindo conhecimentos,
instala-se a dúvida"

(Johann Wolfgang Von Goethe)

Dedico este trabalho ao meu amado filho, Luiz Augusto,
pela compreensão e pelo amor que me doa todos os dias
e ao meu marido, Valerio,
por acreditar e me incentivar em todos os momentos.
Amo vocês.

Agradecimentos

Agradeço de todo coração às muitas pessoas queridas que fizeram parte desta trajetória...

Ao meu orientador, professor Dr. Mario Dantas, pela colaboração e pelo entusiasmo durante o período que estive na UFSC.

Aos meus companheiros de Lapesd, Vinicius, Parra e Alex, pelas importantes contribuições ao trabalho.

À amiga Mirla, pela agradável acolhida em Florianópolis e pelo companheirismo durante este período.

Aos amigos Jô e João, pela afetuosa acolhida e pelo incentivo.

Ao meu sempre mestre, Alexandre Lazaretti Zanatta, pelos conselhos e incentivo.

Aos amigos e compadres, Márcia e Roberto, pelo apoio, e ao meu querido afilhado Yuri, minhas desculpas pela ausência da dinda.

À Universidade Federal de Santa Catarina (UFSC) pela estrutura e corpo docente. Um agradecimento especial à prestativa e simpática Verinha.

À Universidade de Passo Fundo e aos colegas professores, especialmente na figura do coordenador do curso, pela flexibilidade nos horários para as viagens.

Aos colegas da Divisão de Avaliação Institucional da Universidade de Passo Fundo, pelo apoio e pela compreensão nos momentos difíceis.

À minha família por compreenderem a minha ausência e pelo incentivo em todos os momentos.

Por fim, não tenho palavras suficientes para agradecer as pessoas que acompanharam de perto todos os meus dias deste período e que em inúmeras ocasiões não pude dar a atenção merecida. Valerio e Luiz Augusto, obrigado pela paciência, pela torcida, pelo carinho e amor incondicional.

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Abreviações

API	Application Programing Interface
DAG	Directed Acyclic Graph
DNA	Deoxyribonucleic Acid
GRAM	Grid Resource Allocation Manager
GSM	Global System for Mobile Communications
GUI	Graphic User Interface
HTTP	HyperText Transfer Protocol
IEC	International Electrotechnical Commission
IEEE	Institute of Electrical and Electronics Engineers
ISO	International Organization for Standardization
J2ME	Java 2 Micro Edition
J2SE	Java 2 Standard Edition
JNI	Java Native Interface
MDS	Monitoring and Discovery Service
MTBF	Mean time between failures
MTTF	Mean time to failure

MTTR	Mean time to repair
NBR	Norma Brasileira
OGSA	Open Grid Services Architecture
OGSI	Open Grid Services Infrastructure
PDA	Personal Digital Assistants
RNA	Ribonucleic Acid
SNR	Signal to Noise Ratio
SSL	Secure Socket Layer
WiFi	Wireless Fidelity
Wimax	Worldwide Interoperability for Microwave Access
WLAN	Wireless Local Area Network
WMAN	Wireless Metropolitan Area Network
WPAN	Wireless Personal Area Network
WSRF	Web Services Resource Framework
WWAN	Wireless Wide Area Networks
XML	Extensible Markup Language

Resumo

A utilização de dispositivos móveis em ambientes de *grid* computacional tem sido explorada para aplicações técnico-científicas, que muitas vezes podem se beneficiar do poder computacional e mobilidade desses ambientes. Todavia, a integração dos dispositivos móveis requer um tratamento mais criterioso de algumas de suas limitações. Uma das vulnerabilidades existentes em uma configuração móvel é a sua imprevisibilidade quanto às suas conexões, assim, desconexões são usualmente freqüentes em ambientes móveis.

Esta dissertação apresenta uma pesquisa sobre o tratamento das desconexões de dispositivos móveis utilizados pelos usuários para submissão e monitoração de aplicações em ambientes de *grids* computacionais. O conceito de *workflow* foi empregado para a submissão e monitoração de aplicações com várias tarefas para solução de um único problema. A abordagem proposta é constituída por um mecanismo de tratamento de falhas que possui três componentes: o observador, o analisador e o adaptador. Nos casos de desconexão, o mecanismo desenvolvido adapta o fluxo de execução das aplicações considerando a natureza da aplicação e as configurações prévias do usuário.

Com a execução dos testes experimentais observou-se que o mecanismo proposto atua na verificação do status de conexão e, quando necessário, procede ao ajuste na execução da aplicação. Em adição, a abordagem possibilita o reinício de aplicações não finalizadas, sem perder o processamento já realizado. Neste contexto, o mecanismo garante a consistência das aplicações de maneira transparente para o usuário, bem como evita o desperdício dos recursos do ambiente *grid*. Desta forma, a abordagem atingiu com sucesso o objetivo de uma melhor utilização de um ambiente de *grid* computacional, através de um dispositivo móvel, considerando-se suas eventuais desconexões.

Palavras-chave: computação móvel, desconexão, *grids* computacionais, *workflow*.

Abstract

The use of mobile devices in computational grid environments been explored for technical-scientific applications, which many times can benefit of the computational power and mobility of these environments. However, the integration of the mobile devices requires a solid handling of some of its limitations. One of the existing vulnerabilities in a mobile configuration is its unpredictable connection; so, disconnections are usually frequent in mobile environments.

This work presents a research about the handling of the disconnections of mobile devices used by users for application submission and monitoring to grid computing environments. The concept of workflow was used for submission of applications and monitoring of multiple tasks to solve a single problem. The approach proposal is constituted by the mechanism, who comprises three components: the observer, the analyzer and the adapter. In disconnection cases, the developed mechanism adapts the execution flow, considering the nature of application and the previous configurations of the user.

The experimental tests indicate that the mechanism acts in the verification of the connection status and, when necessary, proceeds to the adjustment in the execution of the application. Moreover, the approach enables the restart of applications not completed, without losing the carried out processing. In this context, the mechanism guarantees the consistency of the applications in transparent way for the user, as well as prevents the wastefulness of grid resources. In such a way, the approach successfully achieved the purpose of one better use of an environment of grid computing, through a mobile device, considering its eventual disconnections.

Keywords: mobile computing, disconnection, grids computing, workflow.

Introdução

A computação móvel e os *grids* computacionais são dois paradigmas relativamente recentes na área de sistemas distribuídos que conferem um novo modo de interação e comportamento aos sistemas. Muitas vantagens advêm dessas novas tecnologias, porém, desafios também estão presentes em função das suas características intrínsecas. Quando se projeta uma integração destes dois ambientes com tantas peculiaridades, a complexidade imposta para alcançar a confiabilidade necessária é ainda maior.

De um lado estão os ambientes móveis, proporcionando, sobretudo, mobilidade e acesso a uma infra-estrutura compartilhada, oferecendo aos seus usuários possibilidades antes intangíveis, como obter informações através de seus dispositivos de qualquer lugar e a qualquer hora, e conseqüentemente, aumentando sua produtividade. Porém, juntamente com a computação móvel, novos desafios foram impostos, por exemplo, sua conexão intermitente e poder de processamento e armazenamento restritos. De outra parte, estão os *grids* computacionais, ambientes heterogêneos e distribuídos geograficamente, oferecendo o compartilhamento de amplos recursos para processamento de aplicações que necessitam de alto poder computacional.

É visível a expansão da utilização de dispositivos móveis entre os usuários, ao mesmo tempo em que crescem as necessidades de resolução de problemas cada vez mais complexos. Neste sentido, percebemos as vantagens proporcionadas na integração das tecnologias de computação móvel e *grids* computacionais. Ou seja, usuários móveis, com restrições de processamento, podem submeter suas aplicações ao ambiente de *grid*, monitorar a execução e recuperar os resultados. Diversas pesquisas apresentam soluções particulares com este enfoque (Borges; Dantas, 2006a) (Bruneo et al., 2003) (Clarke; Humphrey, 2002) (Gonzalez-Castano; Vales-Alonso; Livny, 2003) (Park; Ko; Kim, 2003) (Phan; Huang; Dulan, 2002).

Contudo, a integração dos dispositivos móveis requer um tratamento mais criterioso de algumas de suas limitações. Uma das vulnerabilidades existentes em uma configuração móvel é a sua imprevisibilidade quanto à conexão dos dispositivos, com

isso, desconexões são usualmente freqüentes em ambientes móveis. Neste sentido, um aspecto imprescindível em um projeto que envolva a integração destes ambientes é a confiabilidade do sistema, ou seja, garantir a execução e finalização correta do serviço.

1.1 Motivação e contextualização

Em ambientes de *grids* computacionais se percebe um movimento de integração com a tecnologia de dispositivos móveis, tanto como receptores quanto provedores de serviços. Em ambas as abordagens de aplicação de dispositivos móveis, diversas limitações são apontadas (Clarke; Humphrey, 2002) (Park; Ko; Kim, 2003) (Phan; Huang; Dulan, 2002). As principais características restritas dos dispositivos móveis são os seus recursos locais pobres (memória, processamento, armazenamento), reduzida autonomia de bateria, alta mobilidade e conectividade intermitente. Além disso, a dinamicidade é uma das características inerentes do *grid* computacional, considerando que nesses ambientes as falhas ocorrem com grande freqüência. Diante destes aspectos, a integração das tecnologias de *grid* e dispositivos móveis possui desafios no sentido de garantir a consistência das aplicações em casos de falhas de forma transparente para o usuário.

O cenário do problema abordado nessa dissertação engloba ambientes de *grids* computacionais com diversos recursos disponíveis e usuários de dispositivos móveis com capacidades restritas de computação e com necessidades de execução de aplicações com diversas tarefas. Entre os exemplos de aplicações pode-se citar: uma aplicação da área de biologia para sequenciamento de DNA de certa espécie; uma aplicação da física para analisar grande quantidade de dados colhidos por equipamentos; ou ainda, uma aplicação da área química para calcular os riscos ambientais no transporte de poluentes na atmosfera iniciados por um acidente. Neste sentido, os usuários móveis submetem as aplicações para o ambiente *grid*, utilizando os recursos disponíveis, e monitoram o andamento da execução, além de recuperar os resultados parciais e finais.

(Borges, 2006b) desenvolveu um trabalho com enfoque na submissão e monitoração de múltiplas tarefas para ambientes de *grid* computacional utilizando dispositivos móveis. O trabalho demonstra que a abordagem proporcionou uma maneira

mais automatizada e coordenada para executar aplicações em ambiente de *grid* a partir dos dispositivos móveis, utilizando o conceito de *workflow* (Hollingsworth, 1995). Com o emprego do conceito de *workflow*, a abordagem possibilitou que as tarefas da aplicação trabalhassem em um estilo colaborativo para resolver um único problema, combinando redução de consumo de energia com características de alto desempenho.

As tarefas de uma aplicação que estão sendo executadas em recursos do ambiente *grid* podem necessitar de interação com o usuário móvel durante seu processamento, por exemplo, a entrada de dados do usuário, ou ainda a referência a dados armazenados no dispositivo. Neste caso, é necessário manter a conexão com o dispositivo. Porém, nestes ambientes a conexão é intermitente e uma desconexão sugere alguns problemas: a) o que acontecerá se aplicação necessitar de alguma interação? b) como ficará a execução das tarefas, elas continuarão a executar? b) quando o usuário possuir alta prioridade de monitoração, ou seja, acompanhar todo o andamento do processamento, colhendo resultados parciais, continua a execução? d) quando o usuário voltar a conectar, submeterá novamente toda a aplicação, perdendo o processamento já realizado? A abordagem proposta neste trabalho tem como intuito sanar os problemas que podem ocorrer neste cenário.

Conforme (Bruneo et al., 2003) e (Park; Ko; Kim, 2003) expõem, o ambiente do *grid* computacional deve suportar falhas, adaptando-se aos requisitos da aplicação. Neste sentido, o cenário apresentado aponta para a necessidade de tratamento dos problemas de desconexão dos dispositivos, adaptando a execução às características da aplicação, considerando que a disponibilidade intermitente da computação móvel é característica inerente desta tecnologia, além do fato que tal problema não é tratado na pesquisa de (Borges, 2006b).

1.2 Objetivos

O objetivo desse trabalho de pesquisa é propor um mecanismo de tratamento da desconexão de dispositivos móveis que submetem e monitoram aplicações em ambientes de *grids* computacionais, a fim de alcançar uma execução e finalização correta da aplicação. O modelo proposto para o mecanismo de tratamento da

desconexão é uma extensão da abordagem proposta por (Borges; Dantas, 2006a) (Borges, 2006b).

A ocorrência de uma desconexão voluntária ou involuntária do dispositivo móvel é uma falha que pode provocar um erro, e este por sua vez pode gerar um defeito. Desta forma, a intenção é detectar a falha e ajustar a execução, evitando um estado de defeito. Neste contexto, é necessário adaptar o fluxo de execução das aplicações, visando garantir, de maneira transparente para o usuário, a consistência das aplicações submetidas pelos usuários móveis em casos de desconexão do dispositivo.

Os objetivos específicos deste trabalho são:

- Estudar os conceitos e características relacionados à computação móvel;
- Pesquisar trabalhos relacionados que tratam a integração entre dispositivos móveis e ambientes de *grid* computacional;
- Estudar mecanismos de tratamento de falhas;
- Propor um mecanismo de tratamento da desconexão, incluindo a detecção da desconexão, decisão de como proceder de acordo com a aplicação e realizar as adaptações necessárias de maneira transparente para o usuário;
- Implementar o mecanismo proposto para avaliação;
- Avaliar o mecanismo proposto;
- Avaliar o consumo de bateria dos dispositivos móveis com o mecanismo proposto.

1.3 Metodologia

A pesquisa pode ser definida como um conjunto de investigações, operações e trabalhos intelectuais ou práticos que tenham como objetivo a descoberta de novos conhecimentos, a invenção de novas técnicas e a exploração ou a criação de novas realidades. (Kourganoff, 1990). Assim, a pesquisa é uma busca por respostas ou solução de um dado problema empregando métodos e técnicas apropriadas (Silva, 2001). A fim de desenvolver a pesquisa e alcançar resultados ao final das atividades é necessário definir os métodos, os procedimento e materiais que o pesquisador utilizará. Quanto à

Metodologia da Pesquisa, (Silva, Menezes, 2001) define como um conjunto de procedimentos aplicados para alcançar uma investigação disciplinada das relações entre as variáveis de um problema. Além disso, cada tipo de pesquisa possui, além do núcleo comum de procedimentos, suas peculiaridades próprias.

A pesquisa pode ser classificada de diferentes formas de acordo com distintos autores (Gil, 1991) (Silva, Menezes, 2001) (Lakatos, 2000). Uma classificação amplamente utilizada é apresentada a seguir, indicando, também, qual método foi aplicado neste trabalho:

- Quanto à natureza da pesquisa: pode ser considerada básica ou aplicada. A pesquisa básica objetiva gerar conhecimentos novos úteis para o avanço da ciência sem aplicação prática prevista. Já a pesquisa aplicada objetiva gerar conhecimentos para aplicação prática dirigidos à solução de problemas específicos (Silva, 2001). Esta pesquisa é classificada como **aplicada** tendo em vista que os seus resultados são aplicados em ambientes reais.

- Quanto aos seus objetivos gerais: podem ser classificadas como exploratórias, descritivas e explicativas. A pesquisa exploratória proporciona maior familiaridade com o problema, objetivando principalmente o aprimoramento de idéias ou descoberta de intuições (Gil, 1991). Esta pesquisa caracteriza-se como **exploratória** em função do seu objetivo de tratar o problema da desconexão de dispositivos móveis que submetem e monitoram aplicações em ambientes de *grids* computacionais, a fim de alcançar uma execução e finalização correta da aplicação por meio de mecanismos de tratamento de falhas.

- Quanto à abordagem do problema: classificada em qualitativa e quantitativa. A pesquisa quantitativa considera que tudo pode ser quantificável, requerendo o uso de técnicas estatísticas para classificar e analisar os resultados. A pesquisa qualitativa considera que há uma relação dinâmica entre o mundo real e o sujeito. A interpretação dos fenômenos e a atribuição de significados são básicas no processo de pesquisa qualitativa, além da tendência dos pesquisadores em analisar dados de maneira indutiva (Silva, 2001). Este trabalho caracteriza-se como uma pesquisa **quali-quantitativa**, pois faz uma análise do processo da abordagem proposta, além de apresentar dados com relação ao consumo de bateria dos dispositivos.

Os seguintes procedimentos foram adotados para a realização desse trabalho:

- Revisão bibliográfica, por meio, principalmente, de artigos científicos publicados em eventos e periódicos relevantes da área, a fim de fundamentar os diversos assuntos abordados e construir um referencial teórico do que já existe sobre o tema do trabalho. Entre os assuntos pesquisados cita-se: computação móvel, *grids* computacionais, integração entre os ambientes *grid* e computação móvel, tratamento de falhas, tratamento de falhas em ambiente *grids*;

- Estudo detalhado do trabalho de pesquisa de (Borges, 2006b), desenvolvido como trabalho de dissertação no Laboratório de Pesquisas em Sistemas Distribuídos (LAPESD);

- Definição do mecanismo de tratamento de falhas considerando as características inerentes dos dois ambientes de integração a fim de alcançar uma execução e finalização correta das aplicações;

- Implementação do módulo de tratamento de falhas e adaptação dos módulos GUI Móvel e Gerenciador *Workflow* para interagir com o módulo proposto;

- Avaliação do mecanismo proposto a partir de testes com um *workflow* da área da biologia;

- Coleta e análise de dados do consumo de bateria dos dispositivos móveis com a abordagem proposta;

- Escrita da dissertação e descrição dos resultados.

Cabe destacar que este trabalho de pesquisa foi desenvolvido no Laboratório de Pesquisas em Sistemas Distribuídos (LAPESD) localizado no Centro Tecnológico da Universidade Federal de Santa Catarina, que possuía a infra-estrutura de hardware e software necessária para o desenvolvimento da pesquisa.

1.4 Estrutura do Documento

Este trabalho está organizado da seguinte forma: o capítulo 2 aborda os principais conceitos sobre a computação móvel; o capítulo 3 apresenta um panorama sobre *grids* computacionais, os desafios da integração com a computação móvel e os trabalhos

relacionados; os conceitos e princípios básicos da tolerância a falha, bem como sua aplicação em sistemas distribuídos, são apresentados no capítulo 4, com vistas ao entendimento da importância da adoção de mecanismos de tratamento de falhas. No capítulo 5 é apresentado o trabalho relacionado de (Borges, 2006b) e detalhado o modelo proposto. O capítulo 6 contém uma descrição do ambiente experimental, os aspectos relacionados à implementação e os resultados experimentais obtidos; por fim, o capítulo 7 apresenta as conclusões do trabalho realizado e indica os trabalhos futuros.

2 Computação Móvel

A computação móvel, uma abordagem de sistemas distribuídos, evidenciou uma significativa expansão nos últimos anos, possibilitando que os usuários de dispositivos móveis tenham acesso a um ambiente de rede e compartilhem dados, recursos e serviços. A crescente utilização deste paradigma é originada, sobretudo, dos avanços obtidos recentemente nas tecnologias de rede sem fio, que proporcionam aos usuários acesso a uma infra-estrutura compartilhada, independente de sua localização física (Forman; Zahorjan, 1994). Outro aspecto que pode ser observado é a disseminação da utilização destes equipamentos devido à redução de valores, facilidades de transporte, tamanho reduzido e leveza, além de mais funcionalidades incorporadas aos aparelhos. Um exemplo é o *smartphone* que reune as funções de dispositivo de computação e de telefone celular.

Se por um lado o advento da computação móvel propicia o acesso a informações *anywhere-anytime* (Perry et al., 1999) o que traz benefícios aos usuários principalmente no que diz respeito à possibilidade de mobilidade (mudança de local), de outro, diversos desafios surgiram juntamente com esta tecnologia. Alguns autores (Borges, 2006b), (Bruneo et al., 2003), (Clarke; Humphrey, 2002), (Litke; Skoutas; Varvarigou, 2004), (Kurkovsky; Bhagyavati; Yang, 2004), (Park; Ko; Kim, 2003) e (Phan; Huang; Dulan, 2002), citam diversos problemas e desafios que são impostos pelo paradigma da computação móvel e que estão em fase de pesquisas ou ainda em aberto. Do mesmo modo, os avanços na computação móvel tornam possíveis mais usuários acessarem serviços através de uma infra-estrutura de rede independente de localização, todavia, as suas características inerentes revelam grandes desafios. Algumas dessas características que suscitam problemas alvos de pesquisas, a fim de alcançar soluções apropriadas, são:

- pequena capacidade de processamento;
- reduzida capacidade de memória;
- duração reduzida de bateria em relação às necessidades dos usuários;
- heterogeneidade de dispositivos (diferentes tipos de interfaces com maneiras de interação diferentes e limitada ou ainda baixa resolução e pequenas dimensões do

monitor);

- instabilidade e qualidade da rede;
- alta mobilidade.

O trabalho de (Forman; Zahorjan, 1994) aponta que os desafios da computação móvel não são triviais, ou seja, fogem do paradigma conhecido de sistemas distribuídos. Conforme (Chen; Kotz, 2001), (Imielinski; Viswanathan; Badrinath, 1994) e (Satyanarayanan, 2001), o desenvolvimento de sistemas para estes ambientes, que envolvem dispositivos móveis e redes *wireless*, necessitam de um tratamento diferenciado do projeto de uma aplicação convencional de sistemas distribuídos. Os elementos que devem ser considerados neste processo envolvem, sobretudo, as características inerentes destes ambientes, o padrão de comunicação e a rede.

Este capítulo apresenta as principais propriedades da computação móvel, os padrões de rede *wireless*, aspectos relacionados com a desconexão, além de mostrar as possibilidades de integração com *grids* computacionais.

2.1 Propriedades da Computação Móvel

Nesta seção são apresentadas as principais propriedades da computação móvel: a) comunicação sem fio; b) mobilidade; e c) portabilidade. Além disso, evidenciam-se os desafios oriundos de tais propriedades da computação móvel que estão intimamente relacionadas. De fato, os elementos descritos devem ser considerados de forma conjunta como parte fundamental dos projetos de sistemas para estes ambientes, diferentemente das abordagens tradicionais.

2.1.1 Comunicação sem fio

A comunicação sem fio, ou comunicação *wireless*, estabelece a troca de mensagens entre dois pontos ou dispositivos por meio de uma rede que utiliza um mecanismo de ondas eletromagnéticas para transmitir os dados. Segundo (Pitoura; Samaras, 1998), em

função do acesso sem fio, a comunicação na computação móvel defronta-se com muitos obstáculos devido ao fato que o ambiente interage com o sinal. Além disso, em contraste com computadores estacionários, os quais ficam conectados a uma única rede, computadores móveis encontram mais conexões de rede heterogêneas em diversos caminhos (Forman; Zahorjan, 1994). Alguns dos problemas decorrentes da comunicação sem fio são: largura de banda limitada ou/e altamente variável, alta taxa de erros, aumento na latência de comunicação, retransmissões e *timeout delays*.

Nesse contexto, nota-se que as redes sem fio *(wireless)* possuem um recurso de largura de banda escasso em relação à rede cabeada, mesmo considerando as melhorias alcançadas nos últimos anos. Ademais, outros fatores que tornam mais complexa a comunicação são o fato da largura de banda ser dividida entre os usuários que estão em uma célula e ainda e o movimento dos usuários com seus dispositivos. Neste sentido, (Mateus; Loureiro, 1998) afirma que devido às altas taxas de erro na comunicação *wireless*, a eficiência do canal é comprometida. Em decorrência disso, as desconexões na comunicação *wireless*, tornam-se freqüentes.

(Forman; Zahorjan, 1994) observam ainda que, ao considerar que a conexão a um link sem fio é fácil, a segurança da comunicação pode ser comprometida mais facilmente que na comunicação por meio de cabos *(wired)*, especialmente se a transmissão estende-se por uma grande área. Isto requer que os projetos de software incluam medidas de segurança. Um mecanismo de comunicação segura que pode ser empregado em canais não seguros é a criptografia (codificação).

2.1.2 Mobilidade

Mobilidade é a habilidade de mudar de localização enquanto conectado a uma rede (Forman; Zahorjan, 1994). No entanto, a partir desta habilidade, característica inerente dos sistemas de computação móvel, surgem outros fatores relevantes, tais como, informações que em ambientes fixos são considerados estáticos, em ambientes móveis tornam-se dinâmicos. Por exemplo, em um computador fixo, configura-se um servidor a ser usado, já no dispositivo móvel necessita de um mecanismo para determinar qual servidor usar dependendo do local. Assim, podemos observar que em um ambiente

móvel, a localização do usuário pode ser estimada com um item de dado no qual valores mudam a cada movimento (Imielinski; Viswanathan; Badrinath, 1994).

Para (Jing; Abdelsalam; Elmagarmid, 1999) a computação móvel se distingue da clássica conexão fixa devido a: 1) a mobilidade de usuários nômades e seus computadores e 2) restrições dos recursos móveis, tais como largura de banda *wireless* escassa e tempo de vida da bateria limitado. A mobilidade de usuários nômades implica que os usuários podem conectar-se de diferentes *access points* através de *links wireless* e podem querer ficar conectados enquanto se movem, apesar da possibilidade de desconexão.

A mobilidade apresenta benefícios para os usuários móveis na medida em que estes não precisam conhecer a localização do servidor, dos dados ou dos recursos que desejam acessar. Neste sentido, os sistemas devem proporcionar mecanismos que dêem suporte à mobilidade de maneira transparente para o usuário final, como apresentado em (Ferreira et al., 2007).

Segundo (Imielinski; Viswanathan; Badrinath, 1994), mobilidade é um comportamento com implicações para ambas as redes, fixas e sem fio. Na rede fixa, os usuários móveis podem estabelecer uma conexão de diferentes tipos de portas em locais diferentes. Já a conexão sem fio permite mobilidade e conectividade irrestrita de qualquer local dentro da área de cobertura.

De acordo com (Satyanarayanan, 1996) a mobilidade possui restrições intrínsecas:

- *elementos móveis são recursos pobres em relação a elementos estáticos:* considerando diversos aspectos tais como tamanho, energia, ergonomia, recursos computacionais (processador, memória e capacidade de disco).

- *restrição de mobilidade:* refere-se ao aspecto de vulnerabilidade quanto à perda e roubo, ou seja, além dos benefícios da mobilidade, herdam-se as fragilidades.

- *conectividade é altamente variável e impacta no desempenho e confiabilidade:* algumas construções podem oferecer confiabilidade e alta largura de banda, porém outras podem somente oferecer baixa largura de banda. Em ambientes abertos este problema pode ser ainda maior, ocasionando lacunas na cobertura da rede.

- *elementos móveis dependem de uma fonte de energia finita:* enquanto não estiverem disponíveis melhorias quanto às fontes de energia para dispositivos móveis, será necessário considerar sensivelmente este aspecto nos projetos de software e

hardware.

2.1.3 Portabilidade

A portabilidade em computação móvel refere-se à capacidade de portar um dispositivo com capacidade de computação, de pequenas dimensões, leve e com certo tempo de autonomia de energia (Pitoura; Samaras, 1998). Neste sentido, a portabilidade traz diversos benefícios a usuários que necessitam deslocar-se com freqüência e ao mesmo tempo carecem de informações ou acesso de computação para manter sua produtividade. No entanto, juntamente com estas características e benefícios, menores recursos estão disponíveis quando comparados com computadores convencionais. Cabe também destacar que existem várias categorias de dispositivos de computação móvel, desde laptops com maiores recursos, PDA, *smartphones*, até os aparelhos celulares, com recursos mais limitados. Outro aspecto que merece atenção é o fato destes dispositivos não possuírem muito espaço para baterias, ocasionando um tempo de autonomia de energia limitado. (Forman; Zahorjan, 1994) aponta nestes termos que as aplicações podem conservar energia pela redução da sua necessidade por computação, comunicação e memória.

2.2 Rede *Wireless*

O objetivo das redes sem fio é fornecer conectividade em uma infra-estrutura sem fios e cabos. Estas redes proporcionam aos usuários móveis meios de comunicação com outras entidades (usuários, recursos, redes). (Coulouris; Dollimore; Kindberg, 2007) apresenta, conforme tabela 1, quatro tipos de redes sem fio: WPAN, WLAN, WMAN e WWAN. Na tabela é possível observar o alcance de cada tipo de rede sem fio, bem como a largura de banda e a latência.

WPAN - (Wireless Personal Area Network) - rede pessoal sem fio. Fornece conectividade com uma pequena abrangência para dispositivos móveis, ou seja, que estejam fisicamente próximos. É interessante para interligar teclados, impressoras, telefones móveis, agendas eletrônicas, computadores de mão, câmeras fotográficas

digitais, mouses e outros. Os padrões mais utilizados para estabelecer a comunicação são ondas de raio infravermelho.

WLAN - (Wireless Local Area Networks) - redes locais sem fio - rede para fornecer conectividade para dispositivos móveis dentro de uma área (casa, prédio) e prover acesso à Internet ou outra rede fixa (LAN, WAN). Possui variantes de padrões, oferecendo largura de banca de 10 e 100 Mpbs com abrangência de até 1,5 quilômetros.

WMAN - (Wireless Metropolitam Area Network)- rede metropolitana sem fio - possuem, geralmente, maior abrangência que as WLAN, por exemplo, uma cidade ou região metropolitana.

WWAN - (wireless wide area networks) - rede de longa distância sem fio - é por meio deste tipo de rede que os usuários de telefonia móvel têm acesso com ampla cobertura, seja por antenas ou sistemas de satélite. É importante salientar que estas redes oferecem taxas de transmissão de dados relativamente baixas quando comparadas com as demais redes.

Tabela 1. Tipos de rede sem fio (Coulouris; Dollimore; Kindberg, 2007)

Rede	**Exemplo**	**Alcance**	**Largura de banda (Mbps**	**Latência (ms)**
WPAN	Bluetooth (IEEE 802.15.1)	10-30m	0,5-2	5-20
WLAN	Wifi (IEEE 802.11)	0,15-1,5 Km	2-54	5-20
WMAN	WiMax (IEEE 802.16)	5-50 Km	1,5-20	5-20
WWAN	Redes telefônicas GSM, 3G	Mundial	0,010-2	100-500

Conforme (Pitoura; Bhargava, 1994), uma estrutura de sistemas distribuídos que suporta mobilidade possui uma arquitetura que consiste em dois tipos distintos de *hosts*: *hosts* móveis e fixos. Os *hosts* móveis são considerados computadores/dispositivos portáteis interligados em rede por meio de uma conexão sem fio, possibilitando a mobilidade. Já os *hosts* fixos (computadores) estão em uma rede cuja localização e conectividade são constantes. Alguns dos *hosts* fixos, chamados de estação base ou estação de suporte móvel, são acrescidos com uma interface *wireless* para comunicar

com *hosts* móveis. A figura 1 apresenta a arquitetura de rede que suporta conexão sem fio e mostra a interação entre os diferentes elementos que constituem este cenário. A área geográfica abrangível por uma estação base é chamada de célula. Cada *host* móvel pode diretamente comunicar com uma estação base, enquanto se movimenta na área geográfica de abrangência. O processo durante o qual um *host* móvel entra em uma nova célula chama-se *hand-off*. Geralmente, algumas áreas de abrangência de uma célula sobrepõem a área das outras.

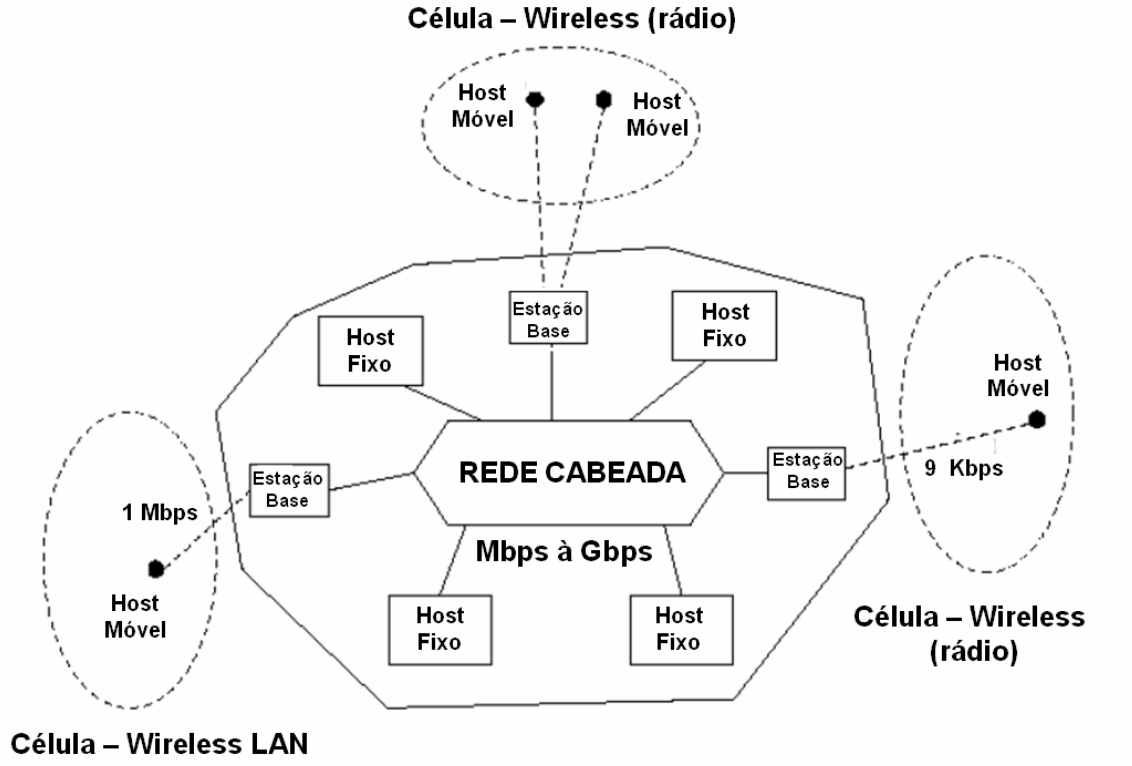

Figura 1. Arquitetura de rede com suporte a conexão *wireless*
(Pitoura; Bhargava, 1994)

Enquanto em sistemas distribuídos não móveis, um *host* opera de dois modos diferentes, conectado ou desconectado. No ambiente móvel, o *host* pode operar de diversas formas, sendo um dos fatores determinantes a disponibilidade de largura de banda. O autor cita que as desconexões freqüentes, sejam elas totais ou parciais, não devem ser tratadas como falhas, ou seja, um *host* móvel deve ser capaz de operar em

casos de impossibilidade de conexão. Os autores (Pitoura; Bhargava, 1994) ainda comentam que outro fator que pode determinar o modo de operação do dispositivo é o tempo de vida de bateria. Neste caso, o dispositivo poderia entrar em um estado de economia de energia.

Outro aspecto importante diz respeito à área de abrangência da célula que pode variar amplamente, desde uma pequena sala (infravermelho), por alguns quilômetros (rádio) ou ainda muitos quilômetros (satélite).

(Pitoura; Bhargava, 1994) destacam que a mobilidade também pode causar a perda ou degradação da conexão *wireless*, visto que os usuários podem mover-se entre as células. Neste contexto, o número de dispositivos em uma célula pode variar dinamicamente, por exemplo, um evento que reúna muitos usuários móveis, pode sobrecarregar a capacidade da rede *wireless*.

2.3 Desconexão

No cenário do ambiente móvel pode-se entender que um *host* móvel está conectado quando está em completo modo de operação. Por outro lado, considera-se desconectado o *host* móvel que se encontra inacessível ao restante da rede (Imielinski; Badrinath, 1993). As desconexões em uma ambiente com características de mobilidade podem ocorrer por diversas razões, por exemplo (Conceição, Kon, 2006):

- alta variabilidade da qualidade da conexão;
- necessidade de economia de recursos (bateria) ou término da energia;
- *hand-offs* (pode ocorrer um salto na força do sinal quando se desconecta do ponto de acesso de sinal mais fraco e conecta-se a outro de sinal mais forte);
- interferências (interferência eletromagnética que pode ocorrer com ondas de microondas, sinais de motores ou outros equipamentos elétricos);
- sombreamento (variação de sinal causada pela obstrução de objetos tais como montanhas, prédios, outdoors, mobílias ou paredes; outra forma ocorre quando entre duas estações existe uma terceira que impede a comunicação entre as duas primeiras).

Corroborando com as propriedades e problemas vistos anteriormente, é possível afirmar que o ambiente móvel é suscetível a desconexões, o que se pode considerar

como uma característica inerente da computação móvel. Para (Mateus; Loureiro, 1998) e (Pitoura; Samaras, 1998) as desconexões de rede podem ser classificadas como: a) voluntárias ou forçadas; b) previsíveis ou repentinas; c) curtas ou longas.

Voluntárias: quando a desconexão é intencional, por exemplo, o usuário pode evitar o acesso à rede para diminuir custo da tarefa de comunicação, o consumo de energia ou o uso de largura de banda;

Forçadas: quando a região onde o usuário está não tem cobertura para acesso à rede;

Previsíveis: as desconexões previsíveis podem ser de natureza voluntária, em função da variação na taxa sinal ruído *(SNR - Signal-to-Noise Ratio),* ou ainda quanto a energia disponível na bateria (em situações quando atinge certo limiar pode fazer com que todos os recursos do ambiente passem a trabalhar com outra qualidade de serviço);

Súbitas: desconexões que ocorrem repentinamente, de forma abrupta;

Curtas ou longas: as desconexões podem durar um pequeno período de tempo, por exemplo, quando há algum obstáculo no sinal, ou ainda ter duração longa em função, por exemplo, de falta de energia da bateria.

Segundo (Imielinski; Badrinath, 1993), quando é possível detectar mudanças no sinal, na previsão de tempo de vida da bateria, ou pelo conhecimento da distribuição da largura de banda, pode-se chamar de desconexões previsíveis.

Os autores ainda enfatizam que os projetos de computação móvel devem considerar a possibilidade do sistema operar de modo desconectado, pelo fato das desconexões serem comuns nestes ambientes.

Conforme citado em (Forman; Zahorjan, 1994), as falhas da rede são uma grande preocupação em computação móvel. Nos projetos de aplicações para computação móvel, deve-se decidir como serão feitos os investimentos buscando amenizar as limitações do ambiente: disponibilizar mais recursos na rede, tentar prevenir desconexões, ou possibilitar aos sistemas lidar com as desconexões de maneira refinada quando possível. Porém, nem tudo pode ser mascarado para o usuário; nestes casos, boas interfaces de usuário podem ajudar fornecendo *feedback* sobre quais operações estão indisponíveis em função da desconexão.

Em um sistema distribuído convencional, um *host* opera em um ou dois modos:

conectado ou desconectado. Já em um ambiente móvel, há mais modos de operação de um *host* móvel (Pitoura; Bhargava, 1994). Pode-se observar no diagrama de estados da figura 2 que os estados possíveis são: totalmente conectado, parcialmente conectado, desconectado e modo "cochilo" (*doze*).

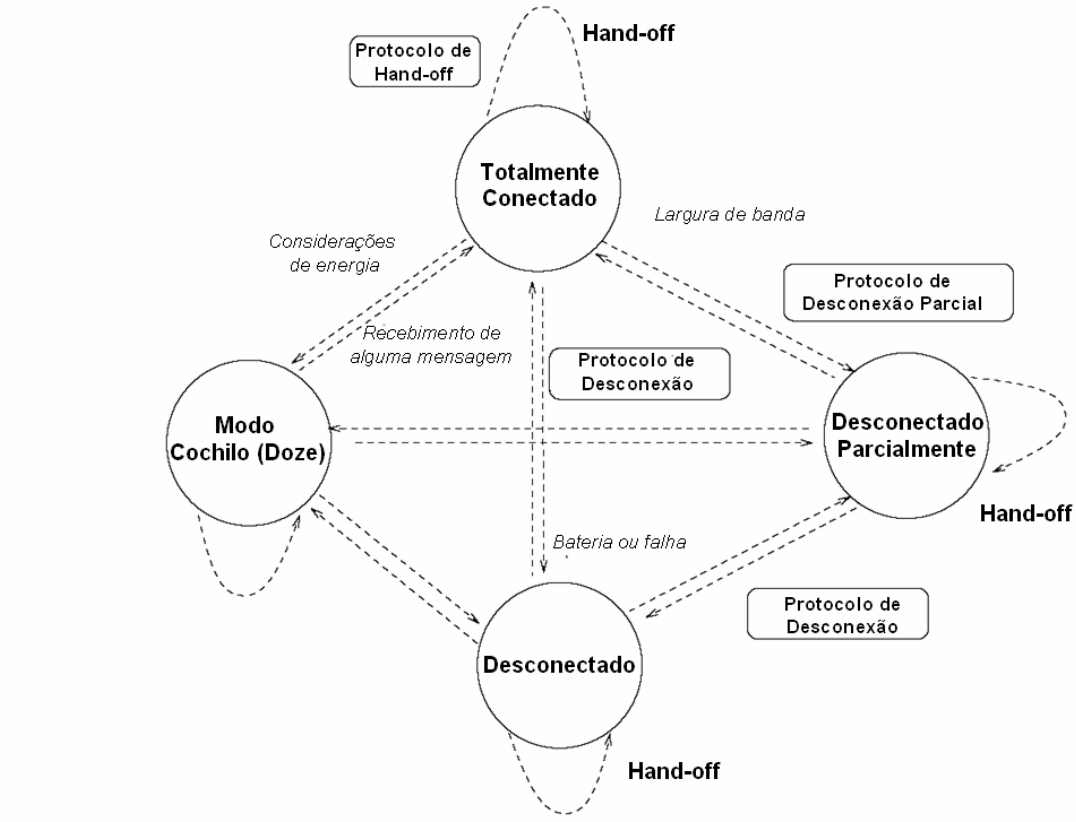

Figura 2. Estados de operação de um elemento móvel (Pitoura; Bhargava, 1994)

Segundo (Pitoura; Bhargava, 1994) o grau de conexão é relativo à disponibilidade da largura de banda. Além disso, como desconexões parciais ou totais são muito freqüentes, elas não deveriam ser tratadas como falhas. Pelo contrário, um *host* móvel deveria ser capaz de operar sob baixa ou sem conexão com a rede estática. Um *host* móvel opera em um modo de desconexão parcial quando a comunicação pela rede está fraca.

Em adição, um *host* móvel pode alternar para modo cochilo com o objetivo de economizar energia. Em decorrência disso, a velocidade de *clock* da UCP é reduzida e nenhuma operação do usuário é executada até receber alguma mensagem externa para

retornar a operação normal.

2.4 Adaptação em sistemas móveis

(Satyanarayanan, 1996) menciona que a mobilidade intensifica a tensão entre autonomia e interdependência, que é característica de qualquer sistema distribuído, em função da relativa escassez de recursos de elementos móveis bem como sua baixa confiança. Assim, segundo o autor, a adaptação é a chave da mobilidade. Neste sentido é imprescindível que sistemas projetados para computação móvel observem eventos/recursos do sistema e a partir destes elementos façam adaptações em tempo de execução, primando pela entrega correta do serviço e alcançando assim um estado de confiabilidade.

3 *Grids* Computacionais

O conceito de *grids* computacionais surgiu em meados da década de 1990, usando este termo como uma referência às malhas do sistema de energia elétrica. (Foster; Kesselman; Tuecke, 2001) definiram um ambiente de *grid* como uma infra-estrutura de software e hardware que possibilita o compartilhamento coordenado e seguro entre coleções dinâmicas de indivíduos, instituições e recursos. Tal ambiente possibilita, por meio da infra-estrutura da Internet, o acesso a recursos como armazenamento, processamento, equipamentos, entre outros, distribuídos geograficamente e subordinados a controles não centralizados (Baker; Buyya; Laforenza, 2000) (Foster; Kesselman, 1999) (Foster; Kesselman; Tuecke, 2001) (Foster; Kesselman, 2003). Cabe destacar que pesquisadores brasileiros utilizam o termo "grade" no lugar de "*grid*", como pode ser visto em (Yamin, 2004) (Goldchleger, et al., 2002).

As entidades que disponibilizam seus recursos no ambiente de *grid* são chamadas de Organizações Virtuais, por exemplo, empresas, centro de pesquisas e universidades. As organizações virtuais compartilham os seus recursos sob uma determinada política ou regra em uma configuração de *grid* (Dantas, 2006).

Em (Roure; Baker; Jennings, 2003) são apresentadas a heterogeneidade, a escalabilidade e a dinamicidade como as principais características do *grid*. A heterogeneidade é a mais evidente, pois os numerosos domínios participantes do *grid* podem possuir recursos distintos (software, conexão, políticas, protocolos, interfaces) e estar espalhados por grandes distâncias geográficas. No *grid* o número de recursos é variável, levando ao aumento/redução do desempenho do *grid* a qualquer momento, ou seja, é um sistema escalável. A dinamicidade do *grid*, ou adaptabilidade, refere-se ao fato de que nesse ambiente falhas são freqüentes, justamente pelas características citadas anteriormente.

Neste contexto, é possível observar que os *grids* computacionais proporcionam aos usuários inúmeros recursos/serviços abstraindo do usuário o aspecto da localização do recurso. A fim de alcançar esta transparência e tratar as suas características, os sistemas de gerenciamento desta plataforma devem ser capazes de adaptar-se dinamicamente e

suportar possíveis falhas.

O paradigma de *grid* é alvo de pesquisas no meio científico, que busca soluções para os desafios ligados à complexidade deste ambiente. Com a evolução das pesquisas e o baixo custo desta plataforma, aplicações comerciais utilizando a abordagem de *grid* têm tido uma maior abrangência. Estes ambientes têm apresentado uma grande expansão nos últimos anos, com diversas aplicações em diferentes áreas de pesquisa, tais como biologia, medicina, física, astronomia e meteorologia. Desta forma, encaminha-se para a integração de *grid* com outras áreas da computação, buscando usufruir do poder computacional disponível nesse ambiente.

Inicialmente, os esforços nas pesquisas relacionadas a *grids* computacionais envolveram soluções proprietárias para compartilhar recursos de computação de alto desempenho. Posteriormente foi introduzido o *middleware*, para tratar a escalabilidade e a heterogeneidade, focando no alto poder computacional e em grandes volumes de dados. Segundo (Dantas, 2006), um *Middleware Grid* é uma camada que fornece protocolos para permitir que múltiplos elementos (servidores, ambientes de armazenamento, redes) participem de um ambiente *grid* unificado.

Assim, tendo em vista a necessidade de ambientes que ocultem do usuário a complexidade, novas ferramentas foram inseridas neste cenário, por exemplo, os *middlewares Legion* e *Globu*s , considerados *core middleware*.

O *Legion*, um sistema baseado em objetos, foi projetado para suportar inúmeros computadores e objetos agrupados por enlaces com altas taxas de transmissão, transparecendo um único sistema computacional. Além disso, possuem mecanismos de abstração que ocultam dos usuários as políticas de escalonamento, gerenciamento de dados, tolerância a falhas, autonomia de sites e opções de segurança (Foster et al., 2002).

O *Globus*, que teve grande impacto na comunidade científica e é um dos *middlewares* mais utilizados em ambientes *grids.* A seguir são apresentados mais detalhes do Projeto *Globus.*

3.1 Projeto *Globus*

O projeto *Globus* busca prover uma infra-estrutura básica que pode ser utilizada para construir implementações de alto desempenho e portáveis. Para alcançar este objetivo o projeto *Globus* possui mecanismos de baixo nível que podem ser utilizados para desenvolver serviços de alto nível (Foster; Kesselman, 1997). O *Globus* possui um conjunto de ferramentas abertas denominada *Globus Toolkit* (Globus, 2006) com bibliotecas para monitoramento, descoberta e gerenciamento de recursos, juntamente com mecanismos de segurança.

Atualmente, as pesquisas em ambientes para *grid* estão abordando um modelo orientado a serviço, com a preocupação de fornecer informações possíveis de serem interpretadas e processadas por máquina (Roure; Baker; Jennings, 2003). Diante de vários projetos em termos de ferramentas e ambientes focados para *grid*, uma proposta de padronização surgiu baseada no conceito de serviços *grid – Open Grid Services Architecture (OGSA)* (Foster et al., 2002). Para melhor entender estes padrões a seguir são apresentados os conceitos e características de *web services* e OGSA.

3.1.1 *Web Services*

Um *web service* é um sistemas de software projetado para suportar interoperabilidade na interação máquina a máquina sobre uma rede (W3C, 2007). Assim, o paradigma *Web Service* determina um conjunto de padrões e tecnologias que facilitam a integração de sistemas heterogêneos, tanto dentro de organizações, como com outros parceiros. O *Web service* é baseado em um conjunto de padrões (XML, SOAP, WSDL, UDDI), que tem como objetivo possibilitar integração de dados:

- XML (*eXtensible Markup Language*): documentos neste formato são usados para entrada e saída de dados;
- HTTP (*Hypertext Transfer Protocol*) ou *Message Oriented Middleware* (MOM): protocolo da aplicação;
- SOAP (*Simple Object Access Protocol*): é o padrão que especifica como os documentos XML são trocados através do HTTP ou MOM.

- WSDL (*Web Services Description Language*): é usada para fornecer uma descrição dos serviços disponíveis.
- UDDI (*Universal Description, Discovery and Integration*): é usado para registrar os serviços disponíveis.

3.1.2 OGSA

A OGSA é um projeto proposto pelo GGF (*Global Grid Forum*) em 2002, que estabelece um padrão para interoperabilidade entre sistemas *grid* com funcionalidades básicas para sua construção. Atualmente, o GGF faz parte do *Open Grid Forum* (OGF), uma comunidade de usuários, desenvolvedores e líderes de mercado que unem esforços para uma padronização dos *grids* computacionais (OGF, 2007). Na padronização proposta, os serviços *grid*, assim como os da *web*, provêem persistência, porém também suportam instâncias de serviços transientes, tais como a criação e o término dinâmico de serviços (Dantas, 2005). .

A OGSA é baseada nos padrões já definidos para os *Web Services* e considera um serviço em um *grid* como um *Web Service* com algumas particularidades definidas através da linguagem padrão chamada WSDL (*Web Services Definition Language*), com pequenas extensões (Foster, et al., 2002). Já as interfaces padrões para os serviços propostos pela OGSA estão definidas na OGSI (*Open Grid Services Infraestructure*), projetada para permitir a composição de serviços baseado em um conjunto de primitivas de computação distribuída (Tuecke, et al., 2002). Na WSDL faltam alguns elementos, então o OSGI tem algumas extensões à WSDL, porém, o padrão para comunicar entre as interfaces do OSGI é completamente baseado na WSDL. Para isso, o OSGI define as interfaces que o serviço mostra ao cliente por meio da WSDL. Assim, os serviços do *grid* podem ser implementados em qualquer sistema, sendo que os clientes não precisam ter nenhum conhecimento da implementação do serviço.

A OSGA define *PortTypes* (um termo do WSDL) para várias ações do cliente, mas a única interface obrigatória para um serviço *grid* é o *GridService PortType*. Este *PortType* é obrigatório pois é com ele que os clientes descobrem informações sobre os serviços do *grid*, pelo conceito de *introspection* e outras maneiras. Já a OSGI trata mais

os detalhes de comunicação entre clientes e serviços *grid*.

Os serviços essenciais para coordenar o trabalho de aplicações que interagem com recursos disponíveis no ambiente distribuído são especificados na OGSA em oito categorias:

- Serviços de infra-estrutura: possibilita comunicação entre recursos diferentes (computador, armazenamento, aplicações) removendo barreiras associadas ao compartilhamento;
- Serviços de gerenciamento de recursos: monitoramento, reserva, distribuição e configuração dos recursos do *grid*, baseados nos requerimentos de qualidade de serviço;
- Serviços de dados: possibilita a movimentação de dados onde ele é necessário – gerenciar replicação, execução de consultas e atualizações e transformação de dados em novos formatos;
- Serviço de contexto: descrever os recursos requeridos e políticas de uso para cada cliente que utiliza o *grid* – disponibilizar recursos otimizados baseados nos requerimentos;
- Serviço de informações: fornecer acesso às informações sobre o *grid* e seus recursos, incluindo o estado e disponibilidade de um recurso específico;
- Serviço de auto-gerenciamento: suporte a obtenção de níveis indicados de serviço de forma automatizada, para reduzir os custos e a complexidade de gerenciamento do sistema;
- Serviço de segurança: reforçar políticas de segurança, oferecendo recursos compartilhados seguros e autenticação e autorização de usuários;
- Serviço de gerenciamento de execução: possibilita que ações simples e mais complexas de *workflow* sejam executadas incluindo: localização, fornecimento e gerenciamento da tarefa.

A versão 3 do *Globus Toolkit (GT3)* incorporou conceitos da OGSI e em 2004 foi lançada a especificação da WSRF (*Web Service Resource Framework*) que propôs mudanças na OGSI, usando conceitos de *web services*. Atualmente o *Globus Toolkit* encontra-se na versão 4.0.

3.2 *Workflow* em ambientes *grid*

Outras abordagens, tais como *frameworks workflow* estão sendo adotadas com o intuito de ocultar do usuário a complexidade da infra-estrutura e os padrões envolvidos (Laszewski; Hategan, 2005) (Shahab et al., 2004). Neste sentido, um conceito que vem sendo empregado recentemente em configurações de *grid* é o *workflow* (Borges, 2006b) (Cannataro et al., 2004) (Oinn et al., 2004) (Shahab et al., 2004). Em (Hollingsworth, 1995) é apresentado o seguinte conceito para *workflow*:

> *"Workflow está relacionado à automatização de procedimentos, onde documentos, informação ou tarefas são passadas entre os participantes de acordo com um conjunto pré-definido de regras, para se alcançar ou contribuir para um objetivo global de um negócio. Apesar de um workflow poder ser manualmente organizado, na prática a maioria dos workflows são organizados dentro de um contexto de um sistema de informação para prover um apoio automatizado aos procedimentos".*

Ao introduzir o conceito de *workflow* em ambientes *grid* é possível atingir uma maior agilidade na execução de diversas tarefas para uma única aplicação. Uma aplicação organizada com um esquema de *workflow* minimiza os erros na ordenação da submissão das tarefas, além de especificar em quais recursos distribuídos as tarefas devem executar.

3.3 Integração entre Computação Móvel e *Grids* Computacionais

A computação móvel é uma tecnologia potencial na integração com *grids*, permitindo duas abordagens: como receptor de recursos ou como um próprio recurso. Esse processo de integração tem sido influenciado pelas capacidades maiores das redes *wireless* disponíveis, protocolos de comunicação mais leves e dispositivos móveis cada vez mais utilizados. Busca-se, desta forma, estender os serviços de *grid* para usuários móveis, acessando recursos distribuídos de forma efetiva, transparente e segura, garantindo a qualidade de serviço. O trabalho apresentado em (Park; Ko; Kim, 2003) menciona duas ênfases para a inserção de dispositivos em ambientes de *grid* :

- Interface móvel: nesta abordagem os dispositivos atuam apenas como uma

interface capaz de submeter e monitorar tarefas em ambientes estáticos;

- Recurso móvel: nesta opção, os dispositivos são considerados como recursos, ou seja, formam um *grid* móvel.

Até pouco tempo, considerava-se que o *grid* não comportava completamente dispositivos móveis, devido aos seus recursos limitados. Todavia, existem vários esforços neste domínio, buscando superar os problemas relacionados às tecnologias. Uma iniciativa de integração foi proposta em (Chu; Humphrey, 2004), o *Mobile OGSI.NET,* que estende a implementação de *grid* para dispositivos móveis baseado na plataforma .NET.

Devido a sua natureza móvel, estes dispositivos estão mais suscetíveis a interferências dos fatores ambientais, tais como distúrbios atmosféricos, alterações de terreno e vegetação. Mesmo com significativos avanços no poder de processamento e na taxa de transmissão de dados dos dispositivos móveis, os recursos disponibilizados para algumas aplicações não são suficientes (Sairamesh et al., 2004). Por outro lado, a tecnologia móvel proporciona facilidades de transporte do dispositivo em função do seu tamanho reduzido e leveza. Assim, os recursos disponíveis em um *grid* podem suprir carências computacionais dos dispositivos móveis. Porém, os usuários necessitam de ambientes apropriados para integrar estas duas abordagens e, desta forma, estabelecer uma maior confiabilidade ao sistema.

3.3.1 Trabalhos Relacionados

Entre os trabalhos de pesquisa que abordam a integração de dispositivos móveis com ambientes de *grids*, podemos citar os trabalho de (Borges; Dantas, 2006a), (Brooke; Parkin 2005), (Bruneo et al., 2003), (Clarke; Humphrey, 2002), (Hummel et al. 2006), (Park; Ko; Kim, 2003) e (Phan; Huang; Dulan, 2002).

Em (Park; Ko; Kim, 2003) os autores focam no problema de desconexão para tratá-lo do ponto de vista que o dispositivo móvel trabalha como recurso do *grid*, por exemplo, para executar tarefas no próprio dispositivo. Desta forma, esta abordagem propõe um novo algoritmo de escalonamento baseado na instabilidade do ambiente móvel para prever uma provável desconexão e assim migrar a aplicação para outro

dispositivo, antes de ocorrer a desconexão. Nesta abordagem, os recursos que são disponibilizados no ambiente *grid* são os próprios dispositivos móveis. Entretanto, esta forma de interação de *grid* móvel limita a resolução de problemas complexos que demandem grande capacidade de processamento e armazenamento (por exemplo, seqüenciamento de DNA em bioinformática ou análise de dados de uma galáxia em astro-física), tendo em vista as restrições existentes nos dispositivos em termos de processamento, memória e armazenamento.

O trabalho apresentado em (Bruneo et al., 2003) analisa os problemas dos usuários que buscam explorar as potencialidades da computação em *grid*. Tal trabalho apresenta uma arquitetura baseada em agentes móveis, capaz de gerenciar a transparência de acesso a serviços distribuídos e a mobilidade dos usuários.

Em (Phan; Huang; Dulan, 2002) há a proposta de uma arquitetura de cluster baseada em *proxy* para introduzir dispositivos móveis em ambiente *grid*. A proposta do trabalho descrito em (Clarke; Humphrey, 2002) é integrar dispositivos móveis a uma infra-estrutura de computação em *grid* baseada no *Legion*, não considerando o dispositivo apenas como uma interface, mas com significativas capacidades computacionais.

Na perspectiva das abordagens apresentadas em (Brooke; Parkin 2005), (Hummel et al. 2006), clientes PDAs podem submeter várias tarefas que trabalham juntas para resolver o mesmo problema, empregando o conceito de *workflow*. O principal foco de (Brooke; Parkin 2005) é a implementação de segurança no cliente PDA, visto que a submissão de tarefas no *middleware* UNICORE requer obrigatoriamente uma comunicação segura entre clientes e o *middleware grid*. Em (Hummel et al. 2006), o principal foco é a descoberta de *workflows* de *data mining* em *grids* para submissão a partir de PDAs. Por outro lado, estas abordagens mostram claramente que clientes PDAs implementam um pequeno conjunto de funcionalidades *workflow*.

O estudo dos trabalhos relacionados revela que pouca atenção tem sido dedicada a alguns aspectos importantes da integração das tecnologias, entre eles: reduzido tempo de vida da bateria, e tratamento da desconexão dos dispositivos e suas conseqüências.

Como já mencionado anteriormente, o trabalho de (Borges, 2006b) prevê a submissão e monitoração de múltiplas tarefas para ambientes de *grid* computacional utilizando dispositivos móveis. A abordagem proporcionou uma maneira mais

automatizada e coordenada para executar aplicações em ambiente de *grid* a partir dos dispositivos móveis, utilizando o conceito de *workflow*. A abordagem de (Borges, 2006b) é apresentada com mais detalhes no capítulo 5.

Tabela 2. Trabalhos Relacionados

Trabalhos	**Tratamento desconexão**	**Workflow**	**Consumo Energia**
Borges, 2006	Não	Sim	Sim
Brooke; Parkin 2005	Não	Sim	Não
Bruneo et al., 2003	Não	Não	Não
Clarke; Humphrey, 2002	Não	Não	Não
Hummel et al. 2006	Não	Sim	Não
Park; Ko; Kim, 2003	Sim	Não	Não
Phan; Huang; Dulan, 2002	Não	Não	Não

A tabela 2 apresenta, de maneira resumida, as características dos trabalhos relacionados. Conforme a tabela, os itens avaliados nos trabalhos levaram em consideração se a abordagem trata o problema da desconexão, se emprega o mecanismo de *workflow* para submissão de múltiplas tarefas e se há uma preocupação com o consumo de energia dos dispositivos móveis. Quanto ao tratamento da desconexão observamos que somente o trabalho (Phan; Huang; Dulan, 2002) emprega um mecanismo de tratamento. Todavia, o referido trabalho não aplica o conceito de *workflow* para submissão de aplicações e seu enfoque é na utilização dos dispositivos como próprios recursos do ambiente *grid*. Quanto ao aspecto consumo de energia dos dispositivos móveis, observa-se que apenas o trabalho de (Borges, 2006b) trata o problema. Outro aspecto importante deste trabalho é a submissão das aplicações com o conceito de *workflow*. Neste sentido, este trabalho de pesquisa apresenta como proposta uma extensão do trabalho de (Borges, 2006b) a fim de tratar a desconexão dos dispositivos que submetem e monitoram aplicações em ambientes *grids,* e, alcançando um estado de execução e finalização sem erros.

4 Tratamento de Falhas

Em todo sistema computacional, supõe-se que este desempenhe as funções para as quais foi projetado, seja um sistema de software ou hardware. O funcionamento não correto de algum componente do sistema computacional pode acarretar prejuízos materiais, ou até mesmo a perda de vidas humanas (Dantas, 2005). De acordo com (Weber, 2002) sistemas totalmente infalíveis são impossíveis, pois falhas são inevitáveis. Assim, a área de tolerância a falhas busca empregar mecanismos que ofereçam sistemas computacionais com um maior nível de confiança.

Um importante conceito relacionado ao assunto, é a dependabilidade, do inglês, *dependability*. A definição apresentada em (Avizienis; Laprie; Randell, 2001) refere-se à dependabilidade como a capacidade de um sistema disponibilizar um serviço que pode justificadamente ser confiável. Segundo (Weber, 2002) o objetivo da tolerância a falhas é alcançar a dependabilidade. Decidiu-se usar o termo dependabilidade neste trabalho assim como alguns autores brasileiros tem referenciado (Dantas, 2005) (Weber, 2002), embora outros autores considerem que o termo não exista na língua portuguesa.

Neste capítulo são abordados os conceitos da dependabilidade, bem como os elementos que a ameaçam, os atributos da dependabilidade, além de apresentar as técnicas de tolerância a falhas e, por fim, são apresentados alguns elementos de falhas em sistemas distribuídos.

4.1 Dependabilidade

Em sistemas de computação, um aspecto que não pode ser ignorado é a questão de garantir a confiabilidade dos resultados esperados e sua disponibilidade. Neste sentido, dependabilidade é vista como a dependência existente entre os componentes do sistema. Como observamos, um sistema deve ter a capacidade de entregar o serviço corretamente. A entrega de um serviço pelo sistema é o seu comportamento tal como ele é percebido pelo usuário, que interage através da interface do sistema. Dessa forma, a

função de um sistema é o que ele propõe fazer, e é descrita pela especificação funcional. Um serviço correto é entregue quando o serviço executa a função proposta pelo sistema. Um defeito do sistema é um evento que ocorre quando a entrega do serviço diverge do serviço correto. Assim, um defeito é uma transição do serviço correto ao serviço incorreto, ou seja, a não execução da função do sistema.

O sistema de classificação, formulado em (Avizienis; Laprie; Randell, 2001) sugere uma taxonomia para a dependabilidade que consiste de ameaças da dependabilidade, seus atributos e os meios para atingi-la. A taxonomia é ilustrada na figura 3.

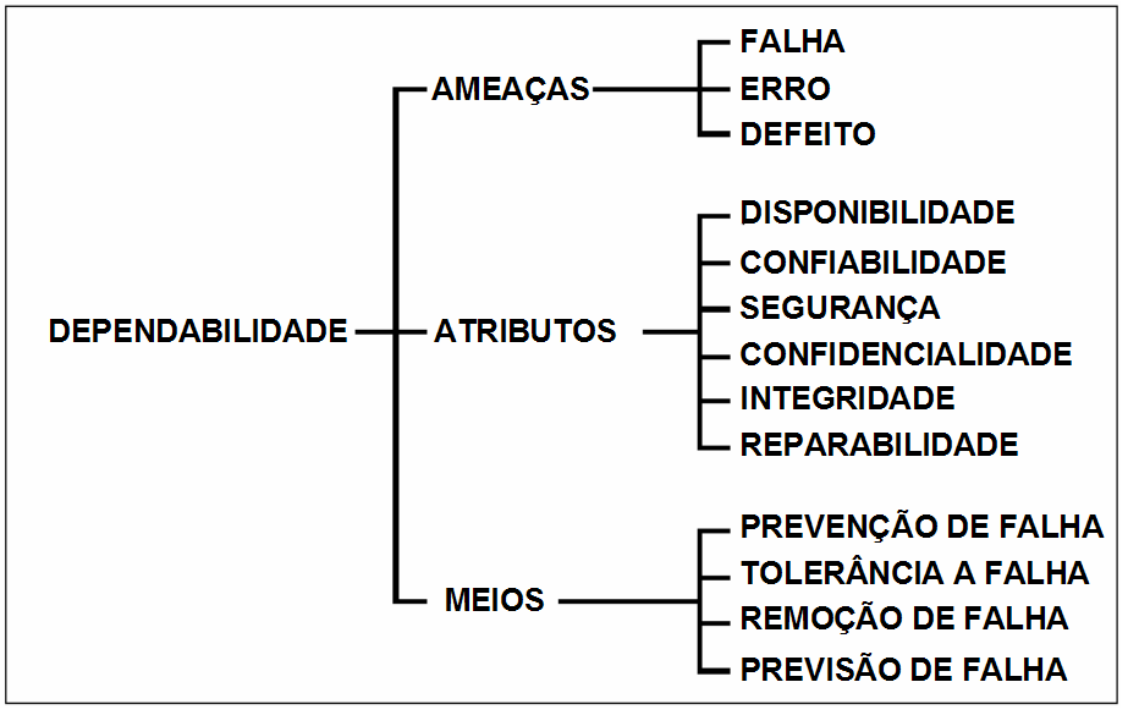

Figura 3. Taxonomia de dependabilidade (Avizienis; Laprie; Randell, 2001)

As ameaças da dependabilidade propostas por (Avizienis; Laprie; Randell, 2001) são, na verdade, uma cadeia de ameaças (falha, erro, defeito). É importante destacar que, alguns autores divergem destes termos e utilizam a expressão "falta" referenciando a "*fault*", e "falha" para "*failure*". Utilizamos neste trabalho, a primeira abordagem apresentada (falha, erro, defeito).

Um processo de serviço incorreto inicia com uma falha até alcançar uma circunstância de defeito. Um sistema onde ocorre uma falha, seja qual for sua natureza, pode gerar um erro ou não. Uma falha que gera um erro é considerada uma falha ativa, caso contrário, é uma falha latente. O erro gerado pode se tornar ativo caso o sistema atinja um estado de defeito, senão é um erro latente. Ao ocorrer um defeito, o sistema não tem a capacidade de entregar corretamente o resultado esperado, ou seja, entra em um estado de serviço incorreto. Dessa forma, um sistema pode atingir uma situação de falha e posteriormente um erro, mas não chegar ao grau de defeito, entregando mesmo assim o serviço corretamente. A figura 4 ilustra a cadeia de ameaça da dependabilidade apresentada em (Dantas, 2005).

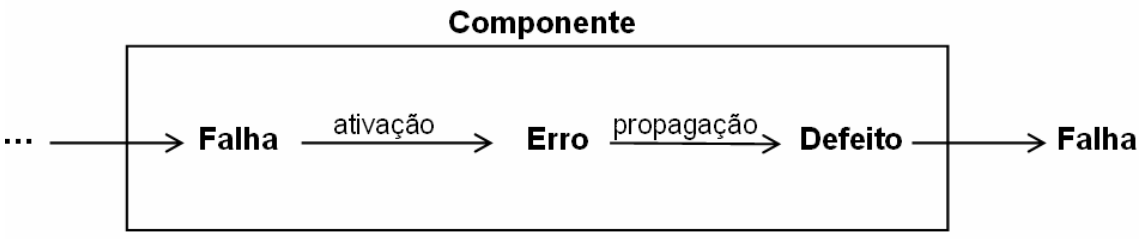

Figura 4. Cadeia de ameaça da dependabilidade

Segundo (Dantas, 2005), os tipos de falhas e suas origens são muito variados. Um exemplo citado pelo autor, é o mau funcionamento do sistema de ventilação de um computador pessoal, considerado uma falha. Como conseqüência, um possível erro no funcionamento do equipamento será seu superaquecimento que levará o sistema a apresentar um defeito. Porém, o mau funcionamento do sistema de ventilação (falha) pode não ser contínuo e não causar a elevação de temperatura (erro) do computador, não causando um mau funcionamento do sistema computacional (defeito) .

Em termos gerais, as causas de falhas podem ser as mais variadas, desde problemas de especificação, implementação, defeito de componente ou ainda fatores externos, tais como interferências eletromagnéticas. (Laprie, 1998) propõe uma classificação de falhas ilustrada na figura 5.

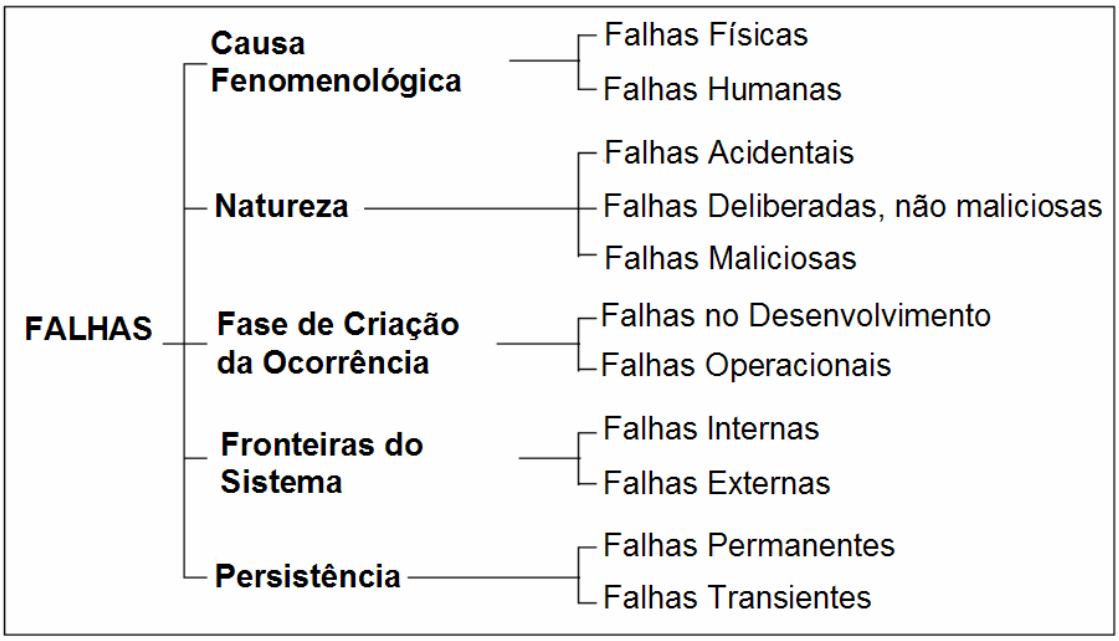

Figura 5. Classificação de falhas (Laprie, 1998)

4.1.1 Atributos

Os principais atributos utilizados para medir a dependabilidade dos sistemas são: confiabilidade, disponibilidade, segurança, confidencialidade, integridade e reparabilidade (Avizienis; Laprie; Randell, 2001). Na tabela 3 são apresentados os elementos considerados para cada atributo.

Com o objetivo de verificar a dependabilidade, algumas medidas de avaliação relacionadas com a confiabilidade são (Weber, 2002):

- *Taxa de defeitos (failure rate, hazard function, hazard rate):* número esperado de defeitos em um dado período de tempo; é assumido um valor constante durante o tempo de vida útil do componente;
- *MTTF (mean time to failure):* tempo esperado até a primeira ocorrência de defeito;
- *MTTR (mean time to repair):* tempo médio para reparo do sistema;
- *MTBF (mean time between failure):* tempo médio entre os defeitos do sistema.

O peso dos atributos da dependabilidade é determinado considerando os requisitos de cada sistema, ou seja, alguns atributos poderão ter um peso maior e outros, menores. Cada atributo é quantificado como uma probabilidade, por exemplo, uma forma de medir a confiabilidade de um sistema pode ser pela probabilidade R(t) de o sistema não apresentar defeito durante o intervalo de tempo considerado, isto é equivalente a:

$$R(t) = P\,(X > t)\, e^{-\lambda t}$$

Onde, *t* é um intervalo de tempo, dado que o sistema apresentava serviço correto em *t=0*, *X* é o tempo para o qual o sistema apresenta defeito (variável aleatória contínua, pois X não pode ser previsto a partir de um modelo determinístico) e o parâmetro l é uma taxa de defeito do sistema (valor constante).

A descrição dos objetivos requeridos dos atributos da dependabilidade em termos da freqüência e rigor aceitável das modalidades de defeitos, e a correspondente duração aceitável de interrupção (quando relevante), para um conjunto de estados de falhas em um ambiente, é o requerimento de dependabilidade do sistema (Avizienis; Laprie; Randell, 2001).

A partir dessas referências, podemos apontar que nos projetos de sistemas computacionais deve-se avaliar a sua natureza e determinar os pesos dos atributos, visando alcançar a dependabilidade do sistema. Neste sentido, o projeto deve prever também algumas técnicas de tolerância a falhas para alcançar tal objetivo. Na próxima seção abordaremos tais técnicas.

Tabela 3. Atributos da dependabilidade

Atributo	**Resumo**
Disponibilidade (*availability*)	probabilidade de o sistema estar operacional num instante de tempo determinado
Confiabilidade *(reliability)*	capacidade de atender a especificação, dentro de condições definidas, durante certo período de funcionamento e condicionado a estar operacional no início do período
Segurança *(safety*)	probabilidade do sistema ou estar operacional e executar sua função corretamente ou descontinuar suas funções de forma a não provocar dano a outros sistemas ou pessoas que dele dependam
Confidencialidade *(Confidentiality)*	proteção contra falhas maliciosas, visando privacidade e autenticidade
Integridade *(Integrity)*	Capacidade de não ocorrer alterações impróprias de estado em um sistema
Reparabilidade *(Maintainability)*	Capacidade que um sistema tem de passar por reparos e modificações

4.2 Técnicas para alcançar a dependabilidade

Na literatura encontramos quatros principais meios de atingir a dependabilidade: prevenção, tolerância, remoção e previsão de falhas (Avizienis; Laprie; Randell, 2001). Tais técnicas objetivam tratar as falhas que podem ser ativadas, causando erros capazes

de se propagarem e gerar defeitos. É importante salientar que as técnicas citadas podem ser utilizadas em conjunto em um sistema. A escolha de uma das técnicas dependerá das características e necessidades de cada sistema.

4.2.1 Prevenção de Falhas

A técnica de prevenção de falhas utiliza mecanismos de controle de qualidade durante o projeto e desenvolvimento de hardware e software. (Dantas, 2005) comenta que este tipo de técnica, por si só, é insuficiente para alguns sistemas atingirem a dependabilidade, visto que, eventualmente, defeitos irão ocorrer no sistema e reparos (manuais) serão necessários para restaurar o sistema à condição de serviço correto (Dantas, 2005). Nestes casos, uma boa alternativa é complementar o uso da prevenção de falhas com o emprego da tolerância a falhas.

4.2.2 Tolerância a Falhas

A tolerância a falhas implica em técnicas que sustentem a entrega correta do serviço, mesmo na presença de falhas ativas no sistema. Esta técnica é a mais indicada para sistemas que precisam de alta disponibilidade e confiabilidade. Porém, nestas circunstâncias são requeridos componentes adicionais a fim de alcançar a dependabilidade. Nestes casos, geralmente, emprega-se o conceito de redundância.

Conforme (Weber, 2002), as técnicas de tolerância a falhas são de duas classes disjuntas: a) mascaramento ou b) detecção, localização e reconfiguração.

O mascaramento de falhas tem como característica a ocultação das falhas. Assim, o sistema mantém-se operacional e garante a entrega do serviço sem qualquer modificação dentro do previsto. Nesta classe, comumente, se emprega mais redundância que nas demais, por isso, é muito indicada para sistemas de tempo real críticos (aviões, sondas espaciais e controles industriais).

Por outro lado, quanto à segunda classe, diferentes classificações são apresentadas por vários autores. Porém, tais classificações se fundem em alguns aspectos. Segundo (Avizienis; Laprie; Randell, 2001) há duas formas de implementar a tolerância a falhas:

detecção de erro e recuperação do sistema.

a) Detecção de erro: este mecanismo busca identificar erros no sistema e diante desta situação gerar um alerta no sistema. É importante destacar que falhas latentes não podem ser observadas diretamente. Assim, esta técnica detecta erros que são conseqüência de uma falha ativada. Um exemplo de mecanismo de detecção é o esquema de duplicação e comparação, onde duas peças idênticas de hardware realizam a mesma computação sobre os mesmos dados de entrada e comparam os resultados na saída. Caso ocorra um desacordo é assinalado erro (Weber, 2002).

b) Recuperação do sistema: a recuperação visa transformar um estado do sistema com um ou mais erros e, possivelmente, falhas, em um estado sem erros e falhas (detectadas e que possam ser ativadas novamente). A recuperação consiste em duas tarefas: tratamento de erro e tratamento de falhas.

O tratamento de erros tem como foco a restauração do sistema ao estado correto. Para isso, há duas formas que podem ser aplicadas conforme ilustrado na figura 6:

3 *Rollback:* onde o estado de transformação consiste do sistema voltar ao estado que estava antes da detecção do erro; este estado armazenado é um *checkpoint*;

4 *Rollforward:* leva os sistemas a um estado novo, no qual ainda não esteve e no qual não há erros detectáveis.

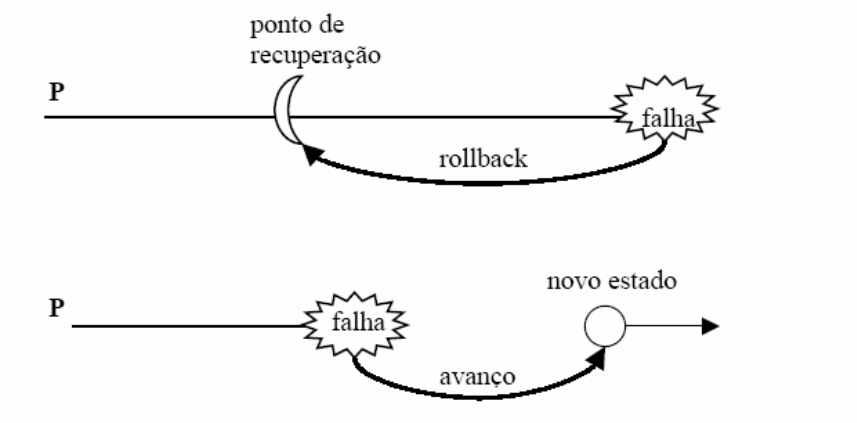

Figura 6. Recuperação por *rollback* e *rollforward*

(Weber, 2002) comenta que a recuperação é complexa para sistemas de processamento distribuído, visto que a recuperação, usualmente de retorno, pode causar

um efeito dominó. Por exemplo, ao desfazer a computação, um processo deixa algumas mensagens órfãs na rede; Processos que receberam e incorporaram essas mensagens devem, por sua vez, desfazer também a computação realizada, provocando que outros processos, que receberam suas mensagens, também tenham que desfazer suas computações.

De outra parte, o tratamento de falhas previne que falhas identificadas sejam ativadas novamente e alcancem um estado de erro. O tratamento é dividido em quatro etapas:

- *Diagnóstico da falha:* deve localizar e identificar o tipo da causa do erro;
- *Isolamento da falha:* procede a exclusão física e lógica de componentes com falhas e com participação no serviço do sistema;
- *Reconfiguração do sistema:* ativa um componente em reserva ou redistribui as tarefas entre os componentes do sistema;
- *Reinicialização do sistema:* verifica, atualiza e grava a nova configuração do sistema.

Geralmente, tratamento de falhas é seguido pela manutenção corretiva que remove falhas isoladas. É importante destacar que a manutenção requer a participação de um agente externo, o que distingue da tolerância a falhas.

4.2.3 Remoção de Falhas

A abordagem de remoção de falhas pode ser empregada tanto na etapa de desenvolvimento, como durante a vida operacional de um sistema. Na fase de desenvolvimento, a remoção de falhas consiste de três passos: verificação, diagnóstico e correção. O processo de verificação analisa se o sistema atende a determinadas propriedades, estas denominadas condições de verificação. Quando é identificado que as condições não são contempladas, é realizado um diagnóstico das origens da falha e as correções necessárias são executadas.

Ainda, segundo (Avizienis; Laprie; Randell, 2001), a técnica de verificação pode ser classificada como estática ou dinâmica:

- *estática:* verifica o sistema, por meio de modelos e teoremas, sem a necessidade de estar executando;
- *dinâmica:* o sistema é verificado durante sua execução, neste caso, deve-se fornecer as entradas necessárias para a execução, seja ela de natureza fictícia ou empírica.

Durante o processo de desenvolvimento é importante avaliar se a técnica implementada será eficiente. Uma alternativa é utilizar um mecanismo de injeção de falhas (*fault injection*). Segundo (Weber, 2002) a injeção de falhas é a introdução controlada de falhas para avaliar o comportamento da técnica empregada, sob tais condições.

Ao longo da vida operacional do sistema, a remoção de falhas é, comumente, realizada por meio de atividades de manutenção (Dantas, 2005):

- Manutenção corretiva: tem como objetivo remover falhas diagnosticadas no sistema;
- Manutenção preventiva: tem como meta descobrir e remover falhas latentes no sistema.

4.2.4 Previsão de Falhas

A técnica de previsão de falhas toma como parâmetro uma avaliação do comportamento do sistema com relação à ocorrência ou à ativação de falhas. Tal avaliação possui dois aspectos:

- qualitativos ou ordinal: busca identificar, classificar e ordenar por importância as causas de defeito nos sistemas;
- quantitativo ou probabilístico: busca medir, em termos probabilísticos, quanto os atributos da dependabilidade são atendidos.

4.3 Falhas em Sistemas Distribuídos

As características inerentes dos sistemas distribuídos requerem que mecanismos de tolerância a falhas sejam incorporados aos sistemas para que estes obtenham uma alta disponibilidade e confiabilidade, como por exemplo são apresentados nos trabalhos de pesquisa (Hosken; Dantas, 2004) e (Rista; Dantas, 2005). Para (Dantas, 2005), a tolerância a falhas deve evidentemente fazer parte do escopo do projeto de qualquer sistema distribuído considerado como arquitetura de execução de aplicações.

Garantir dependabilidade em sistemas distribuídos envolve solucionar problemas de consenso, ordenação e atomicidade na troca de mensagens entre grupos de processos, sincronizar relógios quando necessário, implementar réplicas consistentes de objetos, garantir resiliência de dados e processos em um ambiente sujeito a quedas de estações tanto clientes como servidoras, particionamento de redes, perda e atrasos de mensagens e, eventualmente comportamento arbitrário dos componentes do sistema (Weber, 2002).

Conforme (Jalote, 1994) e (Weber, 2002) os sistemas distribuídos podem ser vistos sob duas visões: modelo físico e modelo lógico. O modelo lógico engloba os componentes, tais como a rede de comunicação e os nodos (processadores, relógio, memória, armazenamento, interface de rede, software). Já o modelo lógico compreende a aplicação como distribuída, a rede completamente conectada e os canais entregam mensagens em ordem conforme envio.

Além disso, um modelo clássico de falhas em sistemas distribuídos foi concebido por Cristian *apud* (Jalote, 1994) e possui quatro categorias: *crash*, omissão, temporização, resposta e arbitrária, como pode ser visto na figura 7.

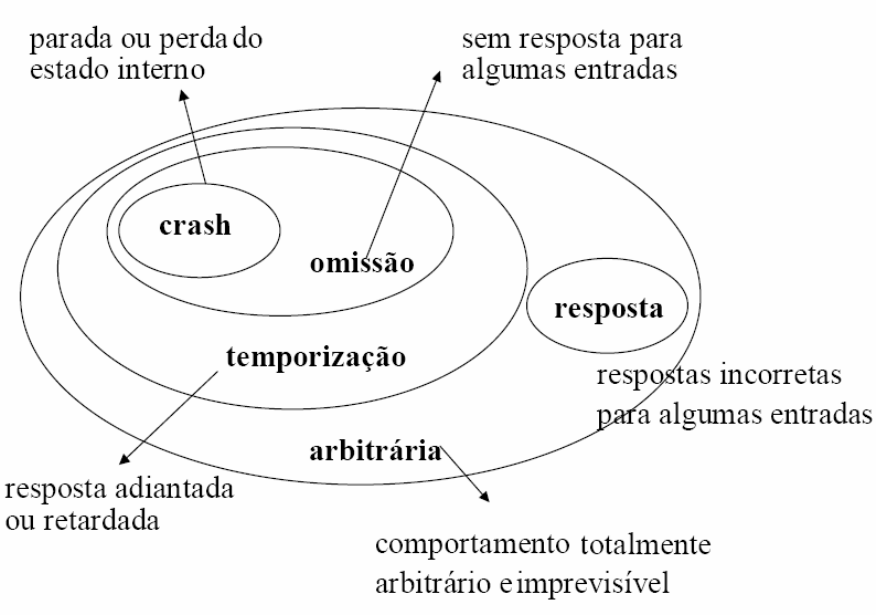

Figura 7. Classificação de falhas em sistemas distribuídos

4.3.1 Falhas na integração de ambientes móveis e *grids*

Várias pesquisas têm resultado em ambientes amigáveis para integração da tecnologia móvel com *grid*. Todavia, não existe uma preocupação forte com o tratamento do dinamismo peculiar desses ambientes que originam falhas freqüentes. Quanto às falhas nesses ambientes, podemos dividir em dois grupos: a) falhas que ocorram no ambiente *grid* (indisponibilidade de recursos, falha no recurso) ou b) problemas na comunicação entre o ambiente *grid* e o usuário.

Entre os trabalhos relacionados do capítulo 3, somente a pesquisa de (Park; Ko; Kim, 2003) prevê algum tratamento de falha na comunicação do ambiente *grid* com o usuário móvel, abordando o problema da desconexão.

O trabalho apresentado em (Yu; Buyya, 2005), que utiliza o conceito de *workflow* para submissão de múltiplas tarefas ao *grid*, apresenta uma taxonomia para tratamento de falhas no ambiente *grid*. Na proposta de (Yu; Buyya, 2005), um gerenciador *workflow* deve possuir entre suas funções a capacidade de projetar, recuperar informações, escalonar, tolerar falhas e movimentar dados. Porém, nos sistemas atuais não existe ainda uma preocupação forte no que diz respeito ao item tolerância a falhas.

Nota-se ainda que os projetos de *workflows* que fazem algum tratamento de falhas não possuem natureza adaptativa, ou seja, adotam opções estáticas de tratamento. Na figura 8 é apresentada a taxonomia para tolerância a falhas para um gerenciador *workflow* (Yu; Buyya, 2005). O tratamento para falhas que ocorram no ambiente *grid*, pode ser realizado em nível de tarefa ou *workflow*. No nível de tarefa, a técnica *Repetir (Retry)* atua tentando novamente executar a tarefa no mesmo recurso. A técnica *Alternar Recurso (Alternate Resource)* executa a mesma tarefa em um novo recurso. A técnica *Checkpoint/Reiniciar (Checkpoint/Restart)* move a tarefa para outro recurso, porém, continua a execução do ponto que ocorreu a falha. Na técnica de *Replicação (Replication)* há a execução da mesma tarefa simultaneamente em recursos diferentes. No nível de tratamento *workflow* a técnica *Alternar Tarefa (Alternate Task)* inicia a execução de outra implementação da tarefa, enquanto que a técnica de *Redundância (Redundancy)* executa múltiplas tarefas alternativas concomitantemente. Na técnica *Tratamento de exceção definido pelo usuário (User-defined Exception Handling)* um usuário define um tratamento específico para uma determinada falha de uma tarefa. No último método, ilustrado na figura 8, o sistema ignora as tarefas com falhas *(Rescue workflow)* e segue a execução do *workflow*.

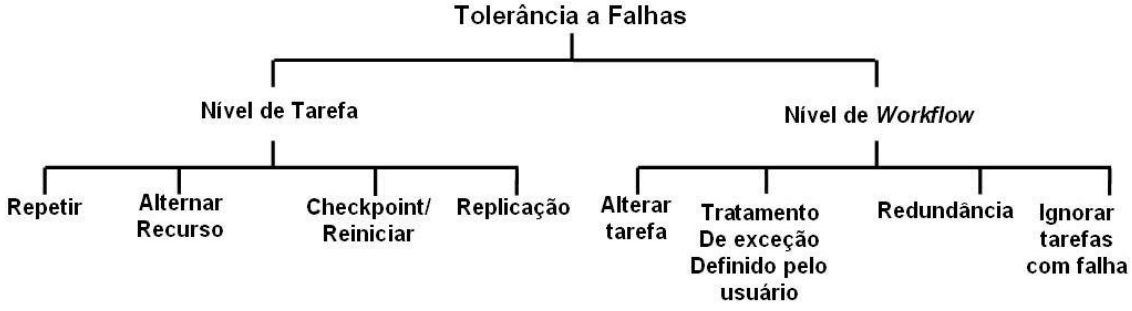

Figura 8. Taxonomia para tolerância a falhas (Yu; Buyya, 2005)

5 Abordagem Proposta

A adoção de recursos encontrados em configurações de *grid* computacional para resolução de problemas complexos, tem sido possível devido à grande evolução observada em termos de hardware e software. Tais ambientes envolvem recursos heterogêneos e de alto poder de processamento. Todavia, em relação aos dispositivos móveis, estes são limitados para resolver problemas complexos. Com este enfoque, Borges (2006b) desenvolveu uma abordagem para4 submissão e monitoração de várias tarefas em *grids* computacionais, a partir do dispositivo móvel. Este mecanismo emprega o conceito de *workflow* para prover uma maneira automatizada e coordenada de submissão e monitoração das tarefas.

A proposta deste trabalho de dissertação é estender a abordagem desenvolvida por Borges (Borges, 2006b) provendo tratamento da desconexão dos dispositivos que submetem e monitoram aplicações em ambiente de *grid.* Considerando sua natureza móvel e de recursos limitados, estes dispositivos tornam-se menos confiáveis por serem mais suscetíveis a desconexões durante a execução de uma aplicação. Por exemplo, desconexões podem ser causadas pelo tempo de vida reduzida da bateria ou interferências na rede *wireless*. A ocorrência de uma desconexão voluntária (ou involuntária) do dispositivo móvel é uma falha. Esta falha pode provocar um erro, e este erro gerar um defeito. Por exemplo, a falha é a desconexão do dispositivo, o erro é a impossibilidade de interagir com os serviços. Isto é, solicitar parâmetros de entrada em uma tarefa específica conforme os resultados obtidos em tarefas anteriores, enviar dados do status das tarefas, ou ainda referenciar dados armazenados no dispositivo. O erro pode gerar um estado de defeito, isto é, um resultado incorreto do fluxo de execução. Por esta razão, torna-se interessante o desenvolvimento de abordagens que apóiem este fluxo de execução e forneçam uma modificação adaptativa em tempo real, para que tal execução leve em conta mudanças ou problemas ocorridos no ambiente móvel enquanto a aplicação é executada, como apresentado no trabalho (Rossetto; Dantas, 2006).

A primeira seção deste capítulo aborda os principais elementos da arquitetura do trabalho de (Borges, 2006b), tendo em vista que esta proposta é uma extensão do

referido trabalho. Posteriormente é descrito o modelo conceitual da proposta com seus componentes e a interação com os módulos do trabalho de Borges (2006b)

5.1 Abordagem de Borges (2006b)

Esta seção apresenta detalhes do trabalho de Borges (2006b) tendo em vista que o trabalho desta dissertação é uma extensão do trabalho de Borges. Assim, é necessário inicialmente compreender a abordagem proposta por Borges (2006b).

Devido ao advento dos *grids* computacionais, fornecendo vários serviços, recursos e a capacidade de resolver problemas complexos, está se tornando cada vez mais comum a execução de aplicações com várias tarefas para a resolução de um único problema. Em geral, este agregado de tarefas tem interações e dependências, necessitando do uso de algumas ferramentas computacionais (por exemplo, software e banco de dados) que são compartilhadas pelas organizações virtuais do *grid* (Borges, 2006b).

As várias tarefas de uma aplicação representam um fluxo de trabalho no qual dados são enviados/recebidos entre as tarefas obedecendo a certas regras. Neste contexto, torna-se necessário o emprego de um mecanismo para controlar, organizar e automatizar este fluxo de tarefas. Para este propósito, o conceito de *workflow* foi empregado na arquitetura de Borges (2006b). A proposta apresenta uma solução genérica e de granularidade fina para definição de recursos do *grid* usados em cada estágio da execução de aplicação em um modo coordenado e automatizado.

A arquitetura de Borges (2006b) é apresentada na Figura 9. Tal arquitetura foi concebida com três componentes principais: GUI Móvel, Gerenciador *Workflow* e *Grid Middleware,* apresentados no lado esquerdo da figura 9.

GUI Móvel: apresenta interfaces de acesso único ao *grid* e adaptadas aos dispositivos móveis (PDA e celular);

Gerenciador Workflow: processa e gerencia os pedidos (por exemplo, submissão, monitoração e *download* de resultados de aplicações) vindos dos dispositivos móveis para executar em um ambiente *grid;*

Serviços do *Middleware Grid* - são fornecidos pelo uso de serviços padrão do

Globus 4.0, que trata casos de gerenciamento de dados e recursos.

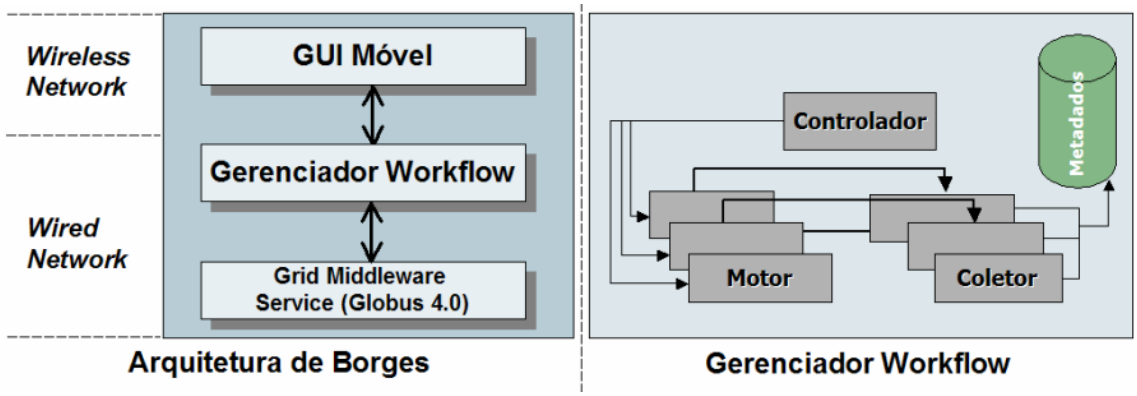

Figura 9. Arquitetura de Borges

5.1.1 Gerenciador *Workflow*

O componente *Gerenciador Workflow* (lado direito da figura 9), que se encontra no servidor, tem como atribuições processar e gerenciar os pedidos vindos dos dispositivos móveis para executar no ambiente *grid*, além de coletar informações relacionadas à execução das tarefas. Ademais, tal componente fornece automatização para usuário móvel, remetendo todas as tarefas para o escalonador de tarefas do *grid* sem a necessidade de interação do usuário. Desta forma, ele permite mais agilidade na execução de tarefas que trabalham juntas para resolver o mesmo problema. Pode-se observar também que a arquitetura provê interações do Gerenciador *Workflow* com a *GUI* móvel e com o *middleware* do ambiente de *grid*. As principais interações são: a) pedido de submissão recebido de um *workflow* qualquer, b) pedido de status do *workflow* submetido e, c) interações com os serviços do *Globus* (por exemplo, *MDS* e *GRAM*) (Foster, 2005). Uma descrição mais detalhada deste componente pode ser encontrada em (Borges, 2006).

O componente *Gerenciador Workflow* foi implementado na linguagem Java, sobre plataforma *J2SE* (Sun Microsystems, 2006c) e usando a *API Karajan Workflow Engine* oferecida pelo *Java CoG Kit* (Laszewski; Hategan, 2005). O *Karajan* é um motor de

execução e linguagem *workflow* versátil e fácil de usar. Segundo Borges (2006), a opção por esta linguagem considerou suas características, principalmente por apresentar todas as estruturas básicas de definição de *workflow* (seqüencial, paralelo, iteração ou *loop* e roteamento), além de estruturas (lista, range e índice *hash*) para tarefas repetitivas que não estão nas linguagens *workflows* disponíveis no meio científico. Em adição, esta API oferece tratamento de falhas, *checkpointing* e execução distribuída em nodos *grid*.

A arquitetura de Borges (2006) foi concebida com três módulos para o *Gerenciador Workflow*: Controlador, Motor e Coletor, como pode ser observado na figura 11. Cada aplicação submetida pelos usuários móveis tem sua própria instância Motor, instância Coletor e um identificador único.

- Controlador: este módulo é responsável por receber os pedidos que chegam dos dispositivos móveis e por ordenar estes pedidos, comandando e controlando sua ordem através de um *Identificador*. Quando este módulo recebe um pedido de submissão, ele cria uma instância do módulo Motor e direciona o pedido para esta instância, conforme ilustra o quadro direito da figura 9.

- Motor: é o principal módulo do gerenciador. Tem como função interpretar diferentes arquivos de definição *workflow* (*scripts workflows* definidos em XML) de diferentes aplicações submetidas para identificar como deve submeter e controlar a execução de todas as tarefas de cada aplicação. Os arquivos de definição ou *scripts* são descritos na linguagem *workflow Karajan* do pacote *Java CoG Kit* e estão armazenados no repositório de metadados (no servidor) disponíveis para acesso de diferentes usuários. Como passo seguinte é realizada a submissão por meio do serviço de *grid* chamado *GRAM (Grid Resource Allocation and Management)* (Foster, 2005). O escalonador *Condor* (Thain; Tannenbaum; Livny, 2005) é responsável por escalonar as tarefas nos recursos do *grid*. À medida que as tarefas são submetidas, seus *status* são informados através de eventos gerados durante a execução. Desta forma, o Motor cria uma instância chamada Coletor.

- Coletor: captura as informações geradas pelos eventos de execução e atualiza-as nos arquivos *checkpoint* (arquivo *XML*) que armazenam o atual status de cada tarefa. Este repositório com os arquivos de *checkpoints* é chamado de *matadados.*

O apêndice B apresenta os diagramas de classe da arquitetura de Borges (2006b) para os componentes GUI Móvel e Gerenciador *Workflow*.

5.1.2 GUI Móvel

A *GUI* Móvel desenvolvida para os dispositivos móveis é responsável por fornecer uma interface para submissão, monitoração de aplicações ao ambiente *grid* e proporcionar a visualização dos resultados. A GUI foi implementada usando o *J2ME (Java 2 Micro Edition) Wireless Toolkit* (Sun Microsystems, 2006b). Algumas das funcionalidades que podem ser encontradas na GUI móvel desenvolvida por Borges (2006b) são listadas a seguir:

- A interface permite consultar as descrições dos *workflows* e de suas tarefas. Esta funcionalidade auxilia os usuários na seleção do *workflow* adequado para resolver seus problemas. Os arquivos *workflow* são estruturados no padrão XML;

- É possível monitorar o andamento da execução do workflow, acompanhando o progresso da execução por meio do status de cada tarefa, por exemplo, executando, falhou ou terminou;

- Visualização de resultados finais e parciais de uma aplicação (somente partes dos arquivos de resultado que são considerados relevantes para o usuário são carregadas na interface), inclusive com o tempo de execução de cada tarefa;

Para implementar uma interface que representasse graficamente o *workflow,* Borges analisou três formalismos geralmente empregados para representar *workflows*: Redes de *Petri*, DAG e Não-DAG. Por meio da representação gráfica é possível acompanhar simultaneamente a execução das várias tarefas da aplicação.

- Redes de *Petri*: pertencem a uma classe especial de grafos orientados. Este formalismo possui os seguintes elementos gráficos: lugar, representado por círculos; transições, representadas por retângulos ou caixas, e arcos de lugares para transições e vice-versa, objetos perceptíveis e individuais que fluem através da rede como *tokens*, uma marcação inicial que define os *tokens* que cada lugar contém no início da execução e uma expressão para cada arco que representa um objeto individual. O formalismo

também descreve execuções de tarefas seqüenciais, paralelas, *join*, *split* e *loops,* além de execução condicional de tarefas, não permitindo somente modelagem de *workflow*, mas também o controle da sua execução.

- DAG (*Directed Acyclic Graph*)**:** é um grafo orientado com nenhum ciclo direcional, isto é, para qualquer vértice *v*, não existe um caminho que sai de *v* e chegue em *v* direcionalmente (AALST et al., 2000). Este formalismo permite representar estruturas como seqüência, paralelismos, *join* e *split*. Muito embora, o formalismo DAG seja amplamente utilizado, em função da sua estrutura simples, ele não permite definir ciclos ou *loops* entre as tarefas e descrever o estado das tarefas *workflows*, somente o comportamento.

- **Não-DAG:** além de todos os patrões contidos no formalismo DAG, o Não-DAG (YU; BUYYA, 2005) inclui a estrutura de ciclos ou *loops*, na qual seções de tarefas *workflows* em bloco de iteração são permitidas. Esta estrutura é frequentemente usada em aplicações científicas. Este formalismo está posicionado entre DAG e Redes de *Petri*.

O formalismo Não-DAG mostrou-se a opção mais indicada para implementação em dispositivos móveis por não apresentar a quantidade de elementos gráficos e a complexidade das Redes de *Petri* e permitir representar *loops* como no formalismo DAG. Com a representação gráfica é possível acompanhar simultaneamente a execução das várias tarefas da aplicação.

A interface de monitoração do dispositivo tem um padrão de cores em cada círculo que indica o *status* atualizado de cada tarefa. Desta forma, quando a interface de monitoração recebe as informações de *status*, ela traduz esta informação em cores que são preenchidas nos círculos correspondentes de cada tarefa. A figura 10 ilustra um gráfico Não-DAG colorido. Assim, quando o círculo está apresentado com a cor cinza significa que a está executando atualmente, a cor azul significa que foi executada com sucesso e a cor vermelha significa que a tarefa falhou por conseqüência de falta de hardware ou software.

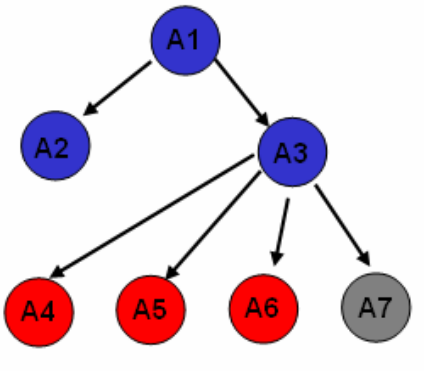

Figura 10. Gráfico Não-DAG colorido

5.2 Modelo Conceitual

Como visto nos capítulos anteriores, a integração da computação móvel com ambientes de *grids* computacionais requer o tratamento de diversas peculiaridades desses ambientes. O capítulo 4 apresentou diversas técnicas para tratamento de falhas; todavia, o cenário do problema apresentado neste trabalho instiga tratamentos específicos de algumas características, tais como a desconexão de dispositivos que submetem aplicações com diversas tarefas a uma configuração de *grid* e monitora a execução. Neste sentido, é necessário o tratamento das falhas que possam ocorrer no sentido de garantir a finalização correta das aplicações.

O cenário do problema tem um enfoque na integração de dispositivos móveis com *grids* computacionais. Neste cenário, usuários de dispositivos móveis, com limitações intrínsecas destes ambientes, submetem tarefas a recursos *grid* empregando o mecanismo *workflow*. Além disso, os usuários ficam monitorando o estado de execução de cada tarefa que compõe a aplicação e recupera resultados parciais e finais. Nesta perspectiva, aplicações podem necessitar interação com o usuário móvel durante a execução, por exemplo, a entrada de dados do usuário móvel ou referenciar dados armazenados no dispositivo. Quando ocorre uma desconexão do dispositivo móvel sob estas circunstâncias, falhas ocorrerão no processamento das tarefas da aplicação. Além disso, outras situações precisam ser tratadas:

a) decidir se a aplicação continuará sendo processada nos recursos *grid*;

b) verificar se o usuário tem alta prioridade de monitoração e precisa acompanhar todo o andamento do processamento, colhendo resultados parciais; nestes casos a aplicação não deve continuar sua execução;

c) quando o usuário reconectar, possibilitar o reinício das aplicações, retomando a execução do ponto que havia parado, sem perder o processamento já realizado.

A partir deste cenário é possível identificar a necessidade de uma camada entre o dispositivo móvel e o ambiente *grid* que proceda ao tratamento de possíveis falhas. A figura 11 apresenta um modelo conceitual da camada intermediária que implica na: detecção da falha, adaptação da execução diante da falha e, a possibilidade de reiniciar a aplicação posteriormente.

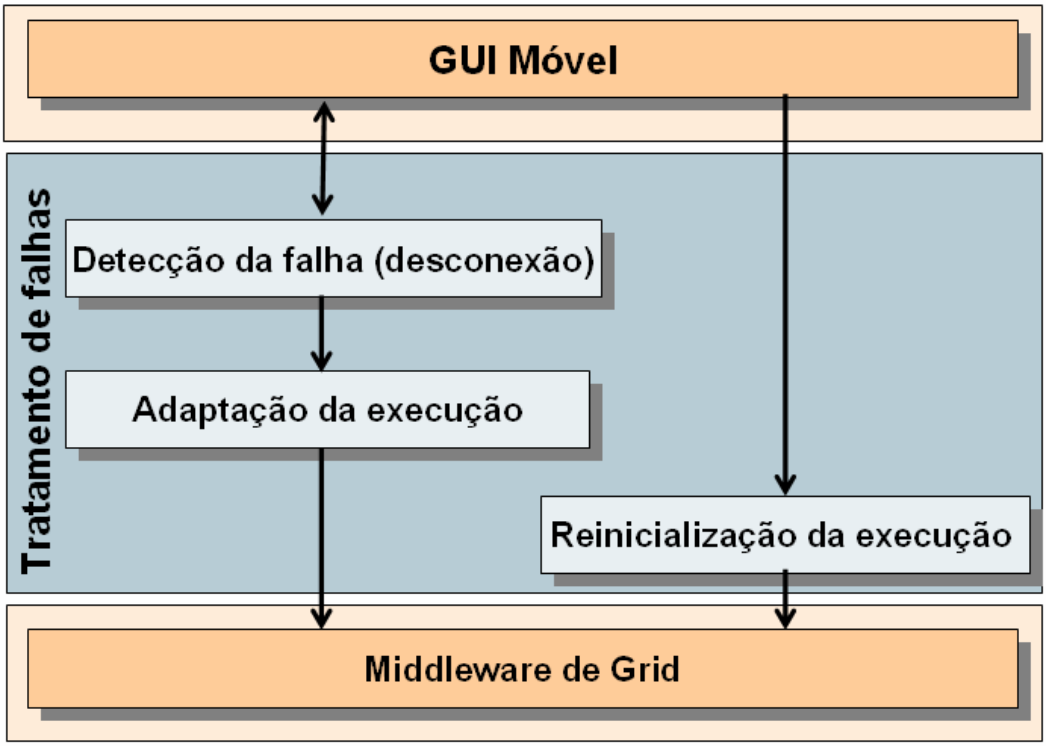

Figura 11. Modelo Conceitual

Com base no modelo conceitual apresentado, este trabalho propõe um mecanismo de tratamento de falhas, chamado de **Módulo TF** com três componentes: Observador,

Analisador e Adaptador. A figura 12 apresenta o Módulo TF, seus componentes e a interação com os demais módulos.

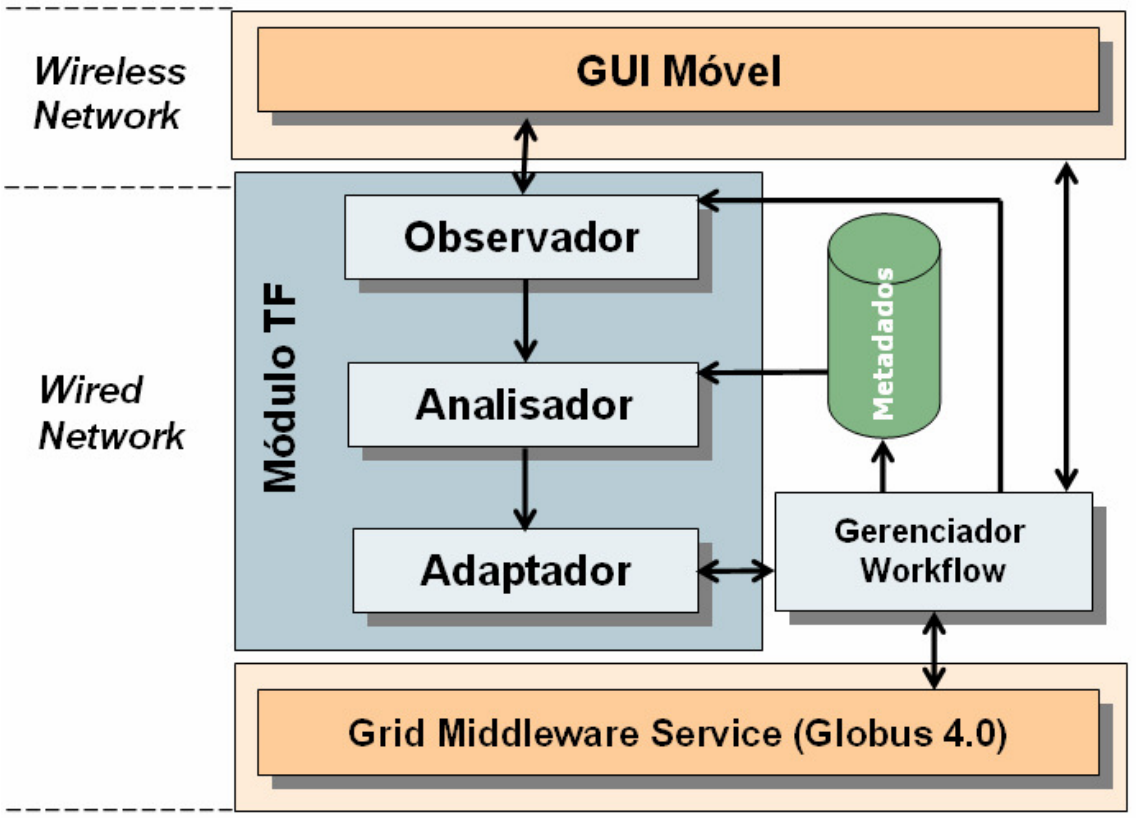

Figura 12. Modelo Proposto

O Módulo TF é incorporado à arquitetura do trabalho de Borges (2006b) conforme ilustra a figura 12. É importante destacar que o Gerenciador *Workflow* do trabalho de Borges (2006b) apenas comunicava falhas que ocorressem nas tarefas, não fazendo tratamento algum. Assim, em situações de falha todo o *workflow* deveria ser submetido novamente. O módulo de tratamento de falhas tem a função de observar o ambiente e especificar as adaptações que devem ser realizadas na execução das aplicações submetidas pelo dispositivo. Na figura 12 é possível observar que o Módulo TF utiliza o Gerenciador *Workflow* para interagir com o *Grid Middleware.*

A ocorrência de uma desconexão do dispositivo móvel é uma falha que pode

provocar um erro, e este por sua vez gerar um defeito. Desta forma, a intenção com o Módulo TF é detectar a falha e ajustar a execução, evitando um estado de defeito. O primeiro passo para alcançar este objetivo é identificar a desconexão do dispositivo que tenha submetido uma aplicação, e então, diante desta situação, determinar como ficará a execução da aplicação. Tarefas submetidas por um usuário móvel podem possuir variados fluxos e necessitar interação ou não. Há, também, possibilidades de dependência da aplicação submetida em relação ao dispositivo, por exemplo, a necessidade de entrada de dados do usuário, ou ainda a referência a dados armazenados no dispositivo. Caso o dispositivo desconecte e o gerenciador *workflow* não trate este evento, a aplicação alcançará um estado de defeito. Além disto, em muitos casos é importante para o usuário acompanhar o fluxo de execução da tarefa submetida, mesmo que tal tarefa não possua dependências. Assim, torna-se imprescindível tratar a desconexão e, se necessário, adaptar a execução da aplicação considerando a natureza da aplicação. Em outras situações, o usuário que desconectou durante uma execução e não conseguiu monitorar toda execução, pode, ao reconectar, submeter novamente a aplicação. Nestes casos, podemos observar um maior consumo de recursos do *grid*, visto que certo tempo de processamento da aplicação foi desperdiçado.

O mecanismo proposto busca adaptar o fluxo de execução para garantir a consistência das aplicações de maneira transparente para o usuário nos casos de desconexão. O módulo TF, apresentado em (Rossetto et al., 2007a) (Rossetto et al., 2007b), possui três componentes: Observador, Analisador e Adaptador. A seguir serão descritos os três componentes.

5.2.1 Observador

O **Observador** tem a função de detectar a desconexão e notificar o Gerenciador *Workflow*. O observador é instanciado pelo Gerenciador *Workflow*, quando este recebe um pedido de submissão. A partir disso, o observador envia uma mensagem inicial para o dispositivo solicitando que este envie mensagens de notificação dentro de um determinado intervalo de tempo a fim de notificar que está conectado. A partir deste momento o observador fica monitorando a chegada das mensagens no intervalo de

tempo. O intervalo de tempo para verificação da conexão é dinâmico, ou seja, definido para cada aplicação considerando três aspectos: tipo da aplicação, prioridade do usuário e tempo de vida da bateria. Este tempo é definido na GUI Móvel e enviado na primeira mensagem para o observador. Quando o observador não receber retorno dentro do tempo estipulado, considera-se que o dispositivo está desconectado e então é ativado o **Analisador.**

Quando uma tarefa da aplicação necessita da entrada de dados do usuário móvel, o Gerenciador *Workflow* pausa a execução e aguarda os dados no intervalo de tempo definido. Se os dados não chegam durante este intervalo, uma exceção é gerada e o Analisador é ativado.

5.2.2 Analisador

O **Analisador** examina a) os requisitos da aplicação (dependências) e b) as opções pré-definida pelo usuário para decidir como proceder na situação de falha.

a) Requisitos da aplicação (Dependências): é considerada dependência qualquer interação que dependa do dispositivo móvel, por exemplo, quando um *workflow* possuir definições de entrada de dados do usuário móvel, referências a arquivos armazenados no dispositivo, referência para gravar dados no dispositivo.

b) Opções pré-definidas: o usuário do dispositivo móvel pode configurar opções para situações de falha. Estas opções são enviadas juntamente com o pedido de submissão. Na GUI Móvel estas opções estão armazenadas em arquivo *xml* e o usuário pode alterá-las pela interface da GUI Móvel. O usuário pode optar por uma das seguintes opções para casos de falha: parar a execução, continuar a execução ou parar para reinício posterior. Assim, mesmo que a aplicação possua características que permitam continuar a execução, o usuário poderá determinar que esta seja interrompida. Este mecanismo permite que os usuários tomem decisões a respeito das suas aplicações considerando suas necessidades, por exemplo, usuários que desejam acompanhar todo o processo de execução, monitorando cada tarefa.

Neste processo, o analisador utiliza as opções pré-definidas pelo usuário, além de pesquisar o arquivo *xml* do *workflow*. O analisador inicialmente consulta as opções do

usuário, verificando ainda se tais opções não comprometem a execução do *workflow*. Quando não houver configurações definidas pelo usuário, o analisador toma a decisão apenas considerando a análise do arquivo *xml*.

Na avaliação do *workflow* (arquivo *xml*) busca-se identificar possíveis interações com o dispositivo móvel. A figura 13 apresenta um exemplo de *workflow* em *xml*. É através deste arquivo que o controlador submete as tarefas ao *grid*. Com as *tags* existentes no arquivo *xml* é possível prever se a aplicação possui interações e, então, determinar se ela pode continuar executando ou não. Um exemplo é a *tag* <*form*> que pode ser usada para construir um formulário de entrada de dados.

O Analisador definirá entre uma das opções a seguir:

- parar a execução da aplicação;
- continuar a execução da aplicação;
- abortar a execução da aplicação.

```
<project>
  <include file="cogkit.xml"/>
  <include file="forms.xml"/>
  <include file="java.xml"/>

  <set name="providers">
    <java:invokeMethod classname="org.globus.cog.abstraction.impl.comm
  </set>

  <while>
    <set name="formData">
      <form:form title="test" id="IDForm" waitOn="IDSubmit, IDQuit">
        <form:vbox>
          <form:hbox>
            <form:vbox homogenous="true">
              <form:label text="Host: " halign="1"/>
              <form:label text="Executable: " halign="1"/>
              <form:label text="Arguments: " halign="1"/>
              <form:label text="Stdout: " halign="1"/>
              <form:label text="Stderr: " halign="1"/>
            </form:vbox>
            <form:vbox homogenous="true">
              <form:textField id="IDHost" columns="10" halign="0"/>
              <form:textField id="IDExec" columns="10" halign="0"/>
              <form:textField id="IDArgs" columns="20" halign="0"/>
              <form:textField id="IDSTDOUT" columns="10" halign="0"/>
              <form:textField id="IDSTDERR" columns="10" halign="0"/>
```

Figura 13. Arquivo *xml* de um *workflow*

Quando é detectada uma desconexão e não há opção de configuração do usuário, por *default* (padrão) foi estabelecido que a execução da aplicação fosse parada na tarefa

que está executando naquele momento. Assim, a aplicação parada poderá ser reiniciada posteriormente. A Tabela 4 apresenta as possibilidades de decisão do Analisador de acordo com as opções do usuário e o resultado da análise realizado no arquivo *workflow*.

Tabela 4. Tabela de decisão do Analisador

Opções usuário/Tipo Aplicação	**Aplicação Dependente** (resultado da análise do *xml*)	**Aplicação não dependente** (resultado da análise do *xml*)
Abortar execução	**Abortar**	**Abortar**
Continuar	**Parar**	**Continuar**
Parar para reinicio posterior	**Parar**	**Parar**
Sem opção	**Parar**	**Continuar**

5.2.3 Adaptador

O **Adaptador** é responsável pelas modificações, quando necessárias. Há duas possíveis situações para o Adaptador intervir no fluxo de execução: a) abortar a execução, e b) parar a execução para reinício posterior. Nestes casos, o Adaptador comunica com o Motor para proceder as adequações. Vale ressaltar que o Gerenciador *Workflow* armazena os estados da execução de cada tarefa em um arquivo de *checkpoint*. Os estados são armazenados a cada alteração gerada pelos eventos do *Karajan*.

Processo de adaptação ocorre com a seguinte seqüência para cada opção:

- Abortar a execução:
  - O adaptador comunica com o motor e solicita a parada da execução;
  - É ativada a atualização do banco de dados (o status do pedido de submissão é alterado para A *(abort)*).
- Parar para reinício posterior:
  - O adaptador comunica com o motor e solicita a parada da execução;
  - É ativada a atualização do banco de dados (o status do pedido de submissão é alterado para S *(stop)*).
- Continuar a execução:

- O adaptador comunica com o motor e altera a propriedade *isConnected* para *false* (para não enviar status das tarefas para o dispositivo);
- Armazena resultados para futura entrega;
- Ao final é atualizado o banco de dados (o status do pedido de submissão é alterado para C *(completed)*).

Quando o Adaptador precisar parar a execução para possibilitar a reinicialização posterior, os estados de execução já estão armazenados no arquivo de *checkpoint*.

5.2.4 Reinício de aplicações

Quando um usuário móvel submeter um *workflow* o Gerenciador faz uma verificação na base de dados, buscando submissões do mesmo *workflow* e mesmo usuário que não tenham finalizado. Assim, quando um dispositivo voltar a conectar ao Gerenciador, será verificada se alguma tarefa da aplicação submetida não foi completada. Neste caso, o usuário será notificado da existência de uma aplicação submetida que não completou a execução. Caso o usuário confirme a reinicialização da aplicação, serão recuperados os estados das tarefas e reiniciada a execução no ponto que havia parado. Se o analisador identificar que a tarefa pode continuar a execução, apenas são armazenados os resultados do processamento para futura entrega ao usuário.

Os estados das tarefas são armazenados no servidor em um arquivo no formato *xml* com a estrutura apresentada na figura 14. O atributo *number* da *tag project* indica o usuário e identificador da submissão e o atributo *workflow_name* contém o nome do *workflow*. A *tag UIDX* significa a identificação de cada tarefa *workflow*. O atributo *line* de cada tarefa indica a linha do *xml* do *workflow* que contém a *tag* da tarefa. Os status possíveis para uma tarefa são: *COMPLETED* (finalizada), *RUNNING* (executando), *FAULT* (Falha).

```
<?xml version="1.0" encoding="UTF-8"?>
<project number="agr/2" workflow_name="Complete-Genome">
<UID8 line="11">COMPLETED</UID8>
<UID9 line="12">COMPLETED</UID9>
<UID10 line="19">RUNNING</UID10>
<UID12 line="15">COMPLETED</UID12>
<UID13 line="16">COMPLETED</UID13>
<UID15 line="19">COMPLETED</UID15>
<UID16 line="20">COMPLETED</UID16>
<UID17 line="21">RUNNING</UID17>
<UID18 line="22">COMPLETED</UID18>
</project>
```

Figura 14. Estrutura do arquivo *xml* de *checkpoint*

As falhas nos recursos do ambiente *grid* também podem ocorrer. Neste cenário, o gerenciador *workflow* irá interromper a execução e o usuário poderá reiniciar a aplicação mais tarde sem perder o processamento já realizado. Com este mecanismo é possível perceber que haverá uma economia dos recursos do *grid,* pois não será necessário processar tarefas já executadas anteriormente. Este procedimento pode ser interessante quando o usuário está pagando para utilizar recursos do ambiente *grid* (BUYYA; ABRAMSON; VENUGOPAL, 2005).

5.2.5 Interação dos componentes

O Módulo TF projetado precisa interagir como os componentes do Gerenciador *Workflow* projetado por (Borges, 2006b) e com a GUI móvel. Na Figura 14 são mostradas as interações entre os componentes da arquitetura. O apêndice C apresenta o diagrama de classes, destacando em azul as classes incluídas para o Módulo TF: *Observador, Analisador, Temporizador, ConectaDB e AlteraWorkflow.*

O diagrama de seqüência da figura 15 apresenta as interações do Módulo TF com o Gerenciador *Workflow* e a GUI Móvel para um pedido de submissão. Inicialmente, o

usuário móvel faz um pedido de conexão ao Gerenciador *Workflow* que responde com o identificador gerado. Após o usuário pode realizar o pedido de submissão de uma aplicação ao Gerenciador que coordena as tarefas da aplicação e controla o fluxo de execução e remete as tarefas para o *Condor*, usado como escalonador do *middleware Globus*. No próximo passo, o Gerenciador cria uma instância do Observador, este envia uma mensagem à GUI Móvel para que esta comece a enviar mensagens dentro de um intervalo de tempo, indicando que está conectado.

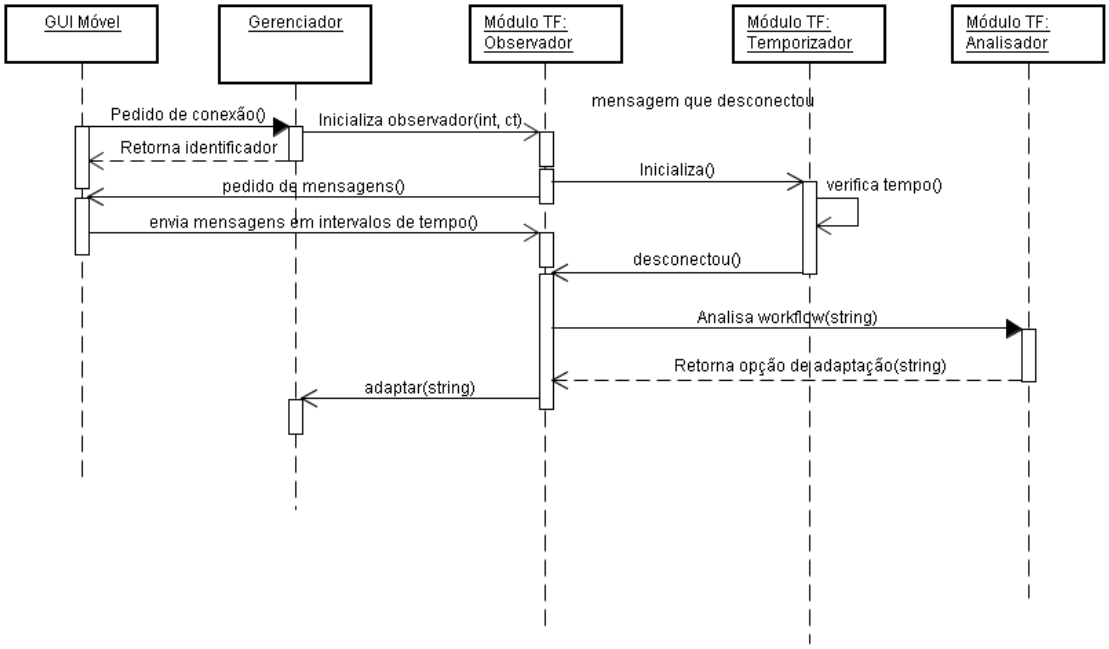

Figura 15. Diagrama de seqüência do pedido de submissão

Para controlar o tempo de recebimento das mensagens o Observador instancia um Temporizador que tem um contador de tempo. Quando o Observador recebe uma mensagem, o contador é reiniciado. O observador continua verificando a conexão até a finalização da execução da aplicação. Se o Observador não receber uma resposta do dispositivo no intervalo de tempo estipulado, é caracterizada uma desconexão. Neste caso, o Observador faz um pedido de análise do *workflow* para o Analisador que retorna a opção de adaptação. Por fim, o observador envia uma mensagem com a opção ao método *adapt* (Adaptador) do Gerenciador que realiza as adequações necessárias.

O diagrama da figura 16 apresenta o diagrama de seqüência para o reinício de uma aplicação. Quando o Gerenciador *Workflow* recebe um pedido de conexão, conforme apresentado no diagrama de seqüência da figura 16, este instancia um objeto da classe *ConnectaDB* que faz a conexão com o banco de dados e verifica as permissões do usuário. Ao receber um pedido de submissão, o Gerenciador *Workflow* solicita ao *ConnectaDB* a verificação se há alguma aplicação com execução pendente para aquele *workflow* e usuário. Caso o retorno seja positivo, o Gerenciador *Workflow* instancia a classe a *AlteraWorkflow* para que faça a leitura do *checkpoint* e carregue o *workflow* não completado anteriormente.

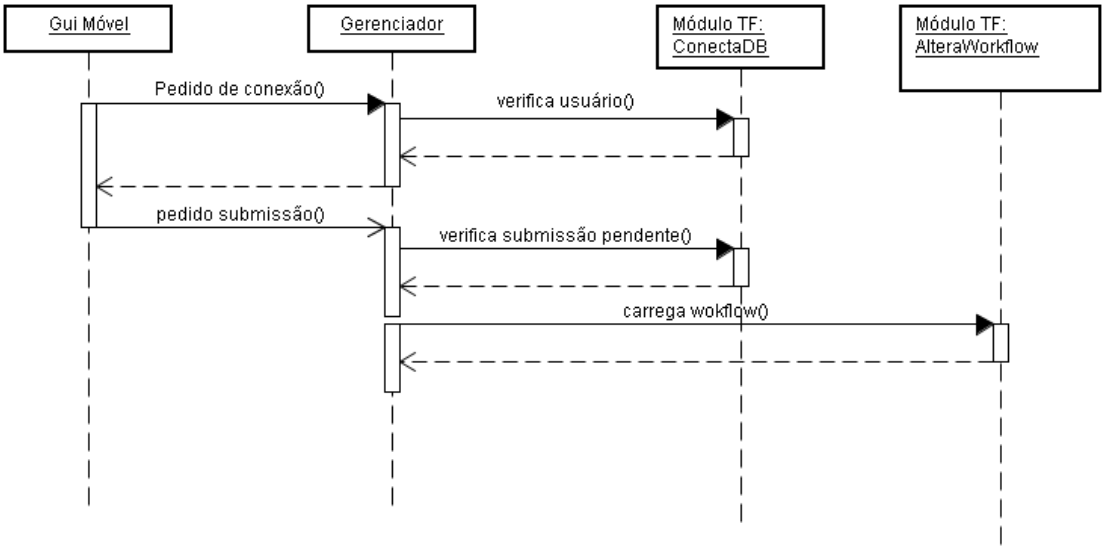

Figura 16. Diagrama de seqüência de reinício de um *workflow*

5.2.6 Intervalo de envio de mensagens de conexão

Como mencionado, para controle da conexão, o observador fica aguardando mensagens do dispositivo dentro de um determinado intervalo de tempo. Este intervalo de tempo de envio das mensagens é dinâmico, variando de acordo com alguns parâmetros:

- tipo da aplicação: se o *workflow* possui alguma dependência do usuário móvel (dados de entrada, dados armazenados no dispositivo). Pode ser dependente ou não dependente;

- prioridade de monitoração do usuário: define se o usuário possui alta, média ou baixa prioridade de monitoração da aplicação;

- tempo de vida da bateria: considera o nível de bateria existente no dispositivo:

  - **alta**: mais de 70% de tempo de vida da bateria
  - **média**: de 30% a 70% de tempo de vida da bateria
  - **baixa**: até 30% de tempo de vida da bateria

Este mecanismo foi concebido tendo em vista algumas situações, por exemplo, em determinados casos, o *workflow* submetido não tem grande prioridade de monitoração e não possui dependências. Nestes casos não há necessidade de ficar enviando um grande volume de mensagens na rede. Todavia, para aplicações com alta prioridade e dependência do usuário móvel, é imprescindível que a desconexão seja identificada o mais cedo possível. Outro aspecto importante é o gasto de bateria gerado em função do envio das mensagens da rede, ou seja, quanto maior for o acesso à rede, maior será o consumo de energia.

Para auxiliar na definição do tempo de intervalo entre as mensagens, foi uma utilizada uma rede *bayesiana.* Uma rede *bayesiana* é uma técnica da inteligência artificial que representa o conhecimento dentro de um contexto de incerteza, por meio de grafos (Russel; Norvig, 2004). Segundo (Nassar, 2003), as redes *bayesianas* são esquemas de representação de conhecimentos para desenvolver uma base de conhecimento. A base de conhecimento possui fatos e regras, associados a incertezas, que representam o conhecimento do especialista do domínio da aplicação, e estes, por sua vez, explicitam as chances de ocorrência por meio de valores de probabilidade. A partir de dados de entrada (*input*), o sistema associa as probabilidades com o conjunto de hipóteses (*output*). A hipótese com a maior probabilidade de ocorrência pode ser considerada a conclusão.

Na construção da rede *bayesiana* foram definidas três possibilidades de tempo de envio de mensagens pelo dispositivo: 8 (oito), 16 (dezesseis) e 24 (vinte e quatro) segundos. Os tempos foram definidos tomando como base os *workflows* utilizados para

avaliação. Cabe destacar que o tempo limite para o observador considerar uma desconexão é de 4 segundos a mais, ou seja, 12 (doze), 20 (vinte) e 28 (vinte e oito) segundos respectivamente. Na definição da rede foi considerado que cada tempo possui a mesma probabilidade de ocorrência. Também foram definidas as probabilidades condicionais a priori, ou seja, a probabilidade de determinado tempo diante de uma evidência. A tabela 5 apresenta as probabilidades condicionais a priori. Por exemplo, se a aplicação é dependente, é indicada uma probabilidade de 90% para o tempo de 8 segundos, haja vista que neste caso é necessário detectar a desconexão o mais breve possível. Porém, se aplicação não é dependente o tempo de monitoramento da conexão pode aumentar, neste caso a probabilidade para o tempo de 8 segundos é de apenas 10%, e 90% para 24 segundos. Estas probabilidades foram indicadas pelo autor analisando cada situação. Para cada hipótese os valores das probabilidades dos itens devem totalizar cem porcento.

Tabela 5. Probabilidades condicionais a priori

Parâmetros		**T1-8s**	**T2-16s**	**T3-24s**
Aplicação	Dependente	90	30	10
	Não dependente	10	70	90
Prioridade	Alta	85	30	5
	Média	10	30	10
	Baixa	5	40	85
Status da Bateria	Alta	85	30	5
	Média	10	30	10
	Baixa	5	40	85

A figura 17 ilustra o diagrama da rede *bayesiana* construída. A partir das probabilidades a priori, são geradas as probabilidades condicionais, ou seja, probabilidade de um evento *A* ocorrer sabendo que um evento *B* ocorreu. No último passo é aplicado o teorema de *Bayes* para gerar novos valores de probabilidades, estas são as probabilidades posteriores. Abaixo é apresentado o teorema de *Bayes*:

$$P(H_i|e) = \frac{P(e|H_i) \cdot P(H_i)}{P(e)}$$

Onde:

$P(H_i \mid e)$: probabilidade a posterior de H_i dado e (reflete a confiança da hipótese H_i depois de observar e)

$P(e \mid H_i)$: probabilidade de e dado Hi

$P(H_i)$: probabilidade a priori da hipótese H_i (representa o conhecimento de domínio)

$P(e)$: probabilidade a priori de e

Utilizou-se o software *Nética* para representar e validar a rede e assim, melhor compreender o processo. No entanto, os cálculos são realizados na própria aplicação quando o usuário faz a solicitação de submissão e o intervalo de tempo definido é enviando juntamente com o pedido de submissão para o servidor. Na figura 17 a rede *bayesiana* está em estado de latência, ou seja, nenhuma evidência da rede está selecionada, ela está aguardando a entrada de dados para calcular as probabilidades.

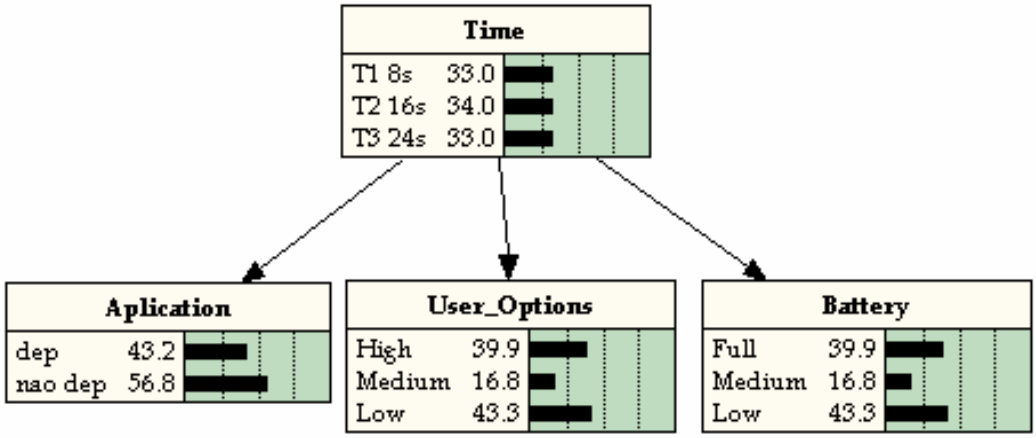

Figura 17. Rede *Bayesiana* em estado de latência

A figura 18 apresenta a rede com determinadas evidências instanciadas. Neste exemplo, a aplicação é dependente, a prioridade do usuário é alta e bateria está com um nível médio de carga. A partir dessas evidências, a rede *bayesiana* calculou as probabilidades para cada hipótese de tempo. Podemos observar que a alternativa mais

indicada de tempo para este caso seria oito(8) segundos.

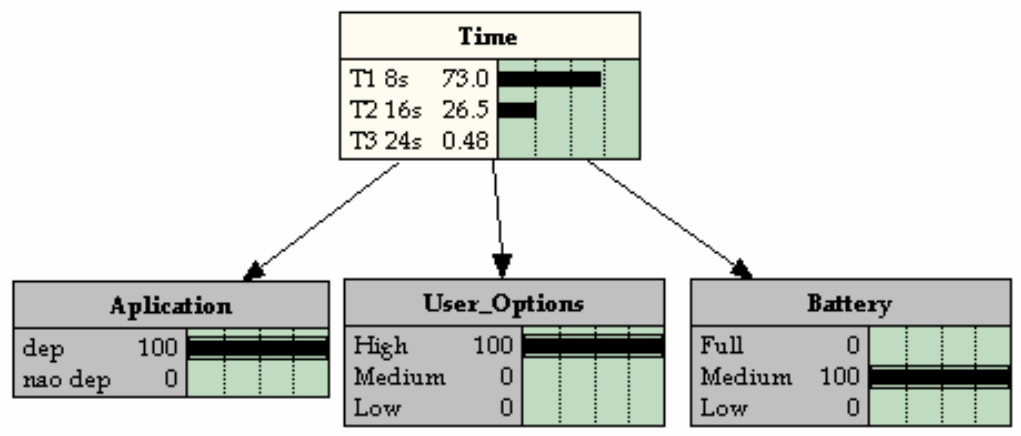

Figura 18. Rede *Bayesiana* instanciada

6 Ambiente e Resultados Experimentais

No capítulo anterior foi apresentado o mecanismo de tratamento de falhas (Módulo TF) proposto com seus componentes e a interação com o trabalho de Borges (2006b). Estes capítulo apresenta a descrição do ambiente experimental, aspectos relacionados à implementação do sistema e a avaliação do mecanismo por meio da confiabilidade do sistema e do consumo de energia dos dispositivos com o sistema desenvolvido.

6.1 Descrição do ambiente experimental

Para o desenvolvimento da abordagem utilizou-se de um ambiente real com uma rede estruturada sendo acessada por usuários de uma rede sem fio, conforme ilustrado na figura 19. No servidor, acessado pelos usuários móveis, foi instalado e configurado o *middleware Globus Toolkit* versão 4.0.0 (Globus, 2006), tendo em vista que é uma versão estável disponibilizada pela *Globus Alliance* (Alliance, 2006), possuir distribuição gratuita, documentação satisfatória e fornecer a nova especificação *WSRF (Web Services Resource Framework)* (WSRF, 2004) para definições de serviço, contemplando a tecnologia de *web services*. Cabe destacar ainda que os serviços oferecidos pelo *middleware Globus* proporcionam segurança, alocação e gerenciamento de recursos, gerenciamento de dados, comunicação entre sistemas heterogêneos, infra-estrutura de informações, detecção de falhas e portabilidade (Globus, 2006).

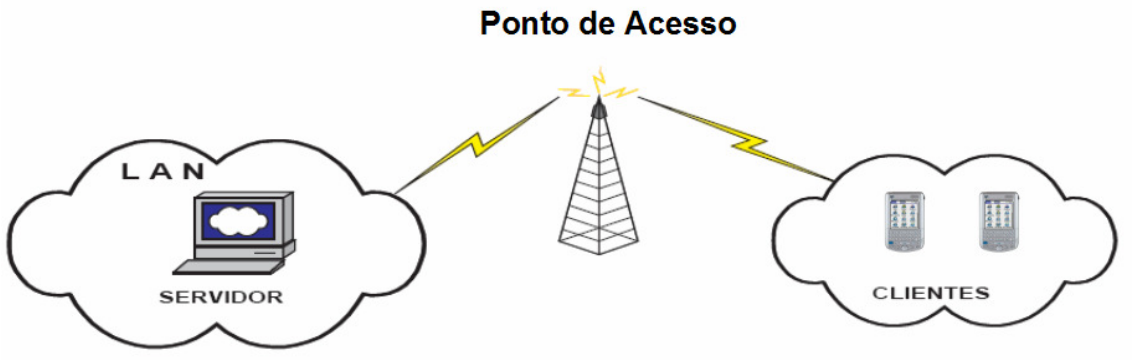

Figura 19. Arquitetura do ambiente

A interação com o *middleware* foi implementada por meio do pacote de software *Java CoG kit (Java Commodity Grid Kit)* (Laszewski et al., 2001). O pacote *Java CoG* possui um grande conjunto de APIs com diversas funcionalidades, permitindo assim, que aplicações interajam com o middleware *Globus*.

Para a comunicação do ambiente móvel com o ambiente *grid*, optou-se pela arquitetura cliente-servidor, que proporcionará aos usuários móveis usufruir os recursos disponibilizados pelas configurações *grid*. Além disso, a aplicação do dispositivo móvel foi implementada com a tecnologia *J2ME (Java 2 Micro Edition) Wireless Toolkit versão 2.2* (Sun Microsystems, 2006b), considerando os aspectos de portabilidade e documentação. O apêndice F abrange mais detalhes das configurações do ambiente de desenvolvimento.

Para estabelecer a comunicação entre os clientes móveis e o servidor, foram utilizadas funções *socket*. Os soquetes fornecem a funcionalidade para estabelecer uma conexão entre dois sistemas, sendo um método de comunicação leve e funcional de comunicação. Os *sockets* têm um baixo *overhead*, o que permite ser um protocolo veloz de comunicação.

Com este mecanismo, o servidor é iniciado e posteriormente os clientes são iniciados para conectar ao servidor. Enquanto a conexão for mantida, os clientes enviam pedidos para o servidor, que por sua vez processa o pedido e responde para o cliente. A conexão é encerrada, normalmente, quando o cliente envia uma notificação de finalização da conexão. A figura 20 ilustra o processo de conexão.

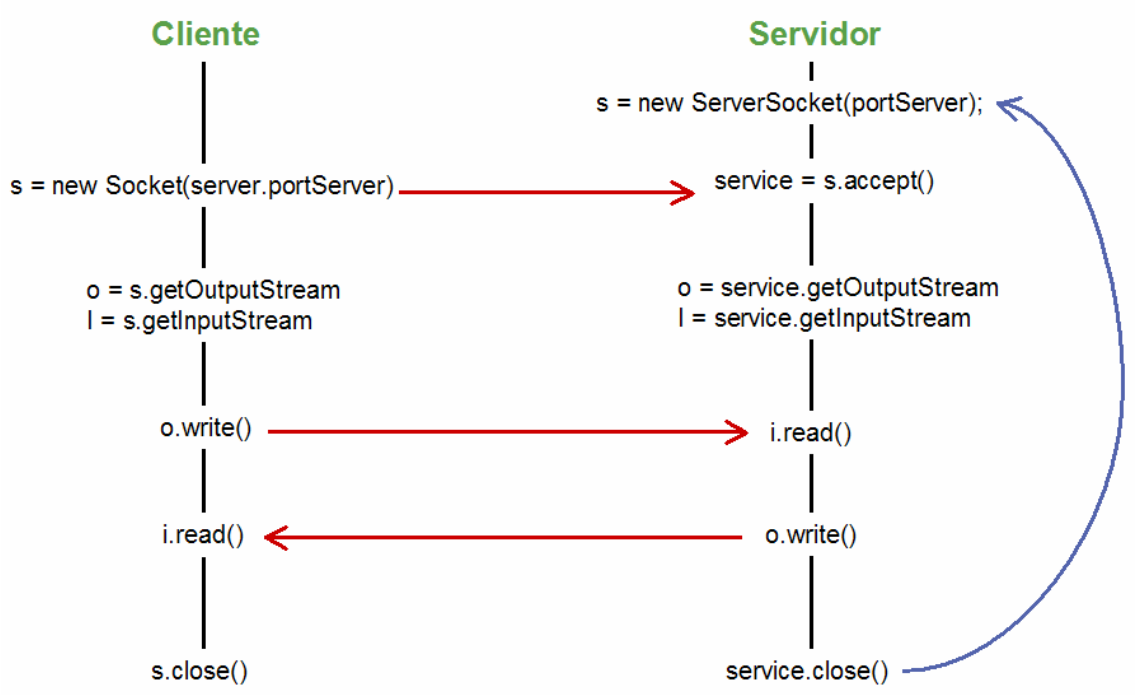

Figura 20. Comunicação *Socket*

Borges (2006b) utilizou, também, o protocolo HTTP para prover a comunicação entre os clientes e o servidor. Este protocolo foi empregado para consulta de informações já processadas pelo *grid* computacional. Porém o enfoque deste trabalho estava na conexão durante o processamento da aplicação, por isso não utilizamos o protocolo HTTP.

6.2 Aspectos da Implementação

Esta seção apresenta detalhes da implementação da abordagem proposta para avaliação do modelo. Cabe ressaltar que a implementação foi desenvolvida tomando como base o sistema do trabalho de Borges (2006b). Porém, diversas alterações foram necessárias nos módulos GUI Móvel e Gerenciador *Workflow*. Entre elas destacam-se:

- alterações no *Motor* do Gerenciador *Workflow* em virtude da troca da versão 4.1.3 do *CoG Kit Karajan* para versão 4.1.4 . Optou-se pela troca da versão do *Karajan* após alguns testes na versão anterior. O método da versão anterior

para parar a execução do *workflow* não estava efetivando a operação na versão anterior. Na nova versão as classes para execução do *workflow* tinham sido alteradas. Foram implementados no Motor os métodos *stop* e *reset;*

- alterações no *ControladorThread* para instanciar o observador;
- alterações no *ControladorThread* para interagir com a classe *ConectaDB,* responsável pela conexão com o banco e atualizações na base de dados;
- inclusão do método *adapt* no *ControladorThread.* Este método é responsável por proceder às adaptações e se necessário comunicar com o Motor;
- alterações no *ControladorThread* para consultar a base de dados e verificar se existem submissões pendentes para o usuário;
- alterações na GUI Móvel para enviar as mensagens de estado conectado para o Observador;
- alterações na GUI Móvel para calcular o tempo de envio das mensagens;
- alterações na GUI Móvel para enviar as opções do usuário para tratamento da desconexão.

Para implementação do módulo TF novas classes foram criadas. O Apêndice C apresenta o diagrama de classes destacando na cor azul as classes incluídas. A seguir são apresentadas algumas considerações sobre a implementação das classes:

- *Observador*: verifica o status da conexão com o dispositivo, por meio de uma *thread* que é criada para cada dispositivo conectado ao componente. O Observador é instanciado pelo *ControladorThread* do Gerenciador *Workflow* quando este recebe um pedido de submissão. Quando o observador é criado, este instancia a classe *Temporizador* que é responsável pelo controle do tempo de chegada das mensagens.
- *Analisador*: classe responsável pela análise do arquivo *xml* do *workflow* e opções do usuário;
- *ConnectaDB*: classe responsável pelo acesso e atualização dos dados na base de dados *MySql*;
- *AlteraWorkflow*: classe que faz a leitura do arquivo de *checkpoint* e procede as alterações no *worklow* para submeter apenas as tarefas não terminadas;
- *Adaptador*: foi implementado como um método no *ControladorThread,* tendo

em vista a necessidade de interação direta com esta classe.

6.2.1 Banco de dados

Para o usuário móvel ter acesso ao gerenciador *workflow* e submeter aplicações, foi criado um mecanismo de controle de acesso usando uma base de dados *MySql* instalado no servidor. Além disso, qualquer submissão realizada pelo usuário fica registrada na base de dados. O diagrama da figura 21 apresenta a estrutura das tabelas do banco de dados *"DBGRID"*.

A tabela *users* armazena os dados dos usuários que possuem permissão para submeter aplicações ao *gerenciador workflow*. Nesta tabela são armazenados o *login*, senha e status do usuário. O campo status pode conter duas opções distintas, *"A"* para usuário ativo ou *"I"* para usuários que não possuem mais permissão de acesso, porém deseja-se manter um histórico das suas submissões.

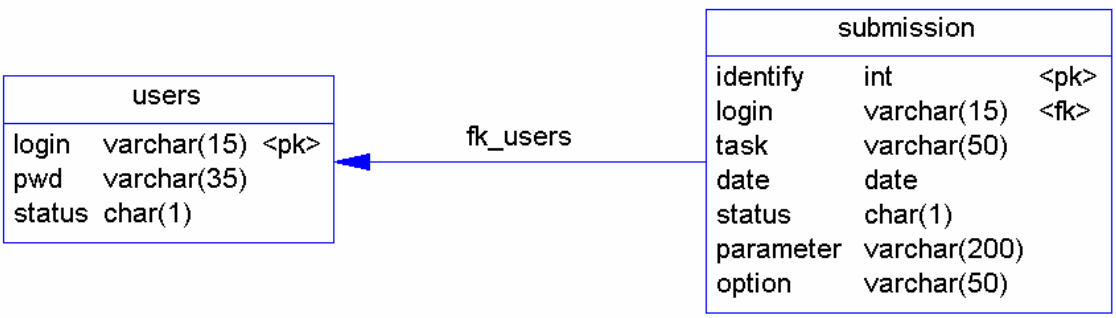

Figura 21. Estrutura das tabelas do banco de dados MySql

Com o objetivo de controlar as submissões originadas dos usuários de dispositivos móveis, definiu-se a tabela *submission*. Quando o usuário faz uma submissão é criado um registro nesta tabela. As informações armazenadas são descritas na tabela 6.

O campo status é utilizado para controlar a situação da aplicação e, sobretudo, para verificar aplicações que não foram completadas. Quando a aplicação é submetida, o valor do campo *status* é *N (new)*, até o *Karajan* retornar que a execução foi iniciada, quando o status passa para *R(running)*. Quando a aplicação é completada o status passa a ser *C(completed)*. Em situações de desconexão e que o adaptador identificou a

necessidade de parar a aplicação para reinicio posterior, o status é modificado para *S(stop)*. Ou ainda, se o adaptador abandonou a execução, o status passa para *A(abort)*. A figura 22 ilustra a tela de login apresentada para o usuário.

Tabela 6. Atributos da tabela *Submission*

Atributos	**Função**
Identify	Identificador da submissão gerado automaticamente
Login	Login do usuário que fez a submissão
Task	Descrição da tarefa submetida
Date	Data e hora da submissão
Status	Situação da aplicação (N=*new*(nova submissão);R=*running*(executando); A=*abort*(abandonada); C=*completed*(completada); S=*stop*(parada))
Parameter	Armazena os parâmetros passados junto com a submissão (são utilizadas para reinicialização da aplicação)
Option	Armazena as opções do usuário para tratamento da desconexão

Figura 22. Tela de Login

6.2.2 *Workflows* utilizados

Assim como Borges (2006b), também se utilizou dois exemplos de *workflows* da área de bioinformática no protótipo implementado. Borges (2006b) comenta que os pesquisadores desta área necessitam de ferramentas ou sistemas mais automatizados para facilitar a análise de seqüências DNA, RNA e proteínas. Além disso, estes processos necessitam de diversos programas de computador e de um grande volume de dados que em alguns casos estão armazenados em locais geograficamente distintos. Assim, os dois *workflows* utilizados são:

- *Workflow 1 (sequence-workflow)* – tem como objetivo pesquisar nucleotídeos em três diferentes bancos de dados de proteínas armazenados no servidor. As descrições dos recursos utilizados neste *workflow* estão detalhadas no apêndice E. A figura 23 ilustra o fluxo das seis tarefas do *workflow*.

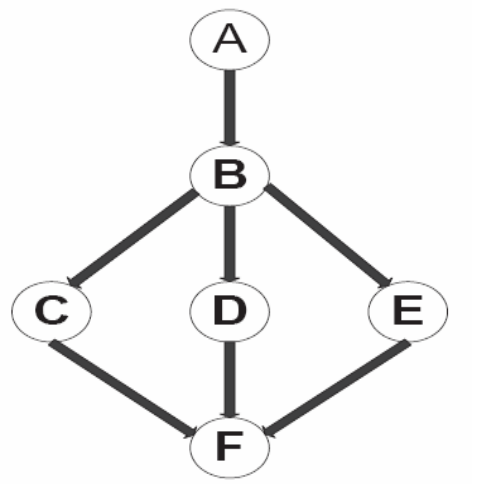

Figura 23. *Workflow 1 – Sequence Workflow*

- *Workflow 2 (Complete-Genome)*– tem como objetivo analisar a bactéria *Gluconacetobacter diazotrophicus* do projeto Genoma (Lemos, 2004). O *xml* do *workflow* definido na linguagem Karajan está no apêndice D. A descrição de todos os programas e banco de dados utilizados em cada tarefa deste *workflow* está no apêndice E. A figura 24 ilustra o fluxo das sete tarefas do *workflow* .

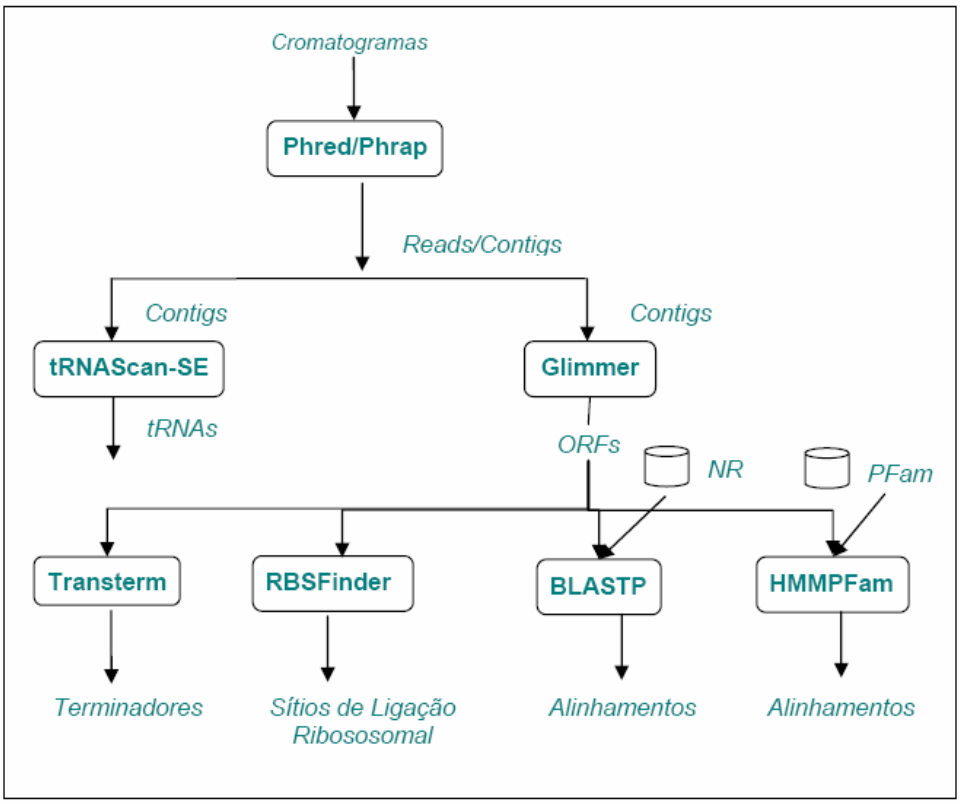

Figura 24. *Workflow 2 – Genoma Complete Workflow*

Toda vez que o usuário submete uma aplicação ao gerenciador, o Módulo TF verifica se existem aplicações pendentes. Neste caso o usuário é questionado sobre a possibilidade de reinício da execução. Se o usuário confirmar o reinício, o Módulo TF reiniciará a execução do ponto que havia parado. A figura 25 ilustra o aplicativo do dispositivo móvel monitorando os dois *workflows* implementados.

Figura 25. Monitoramento dos *workflows*

Durante o período de desenvolvimento, verificou-se que o *Cog Kit Karajan* não estava reiniciando o *workflow* usando o arquivo de *checkpoint* gerado pela própria ferramenta. Diversas tentativas foram realizadas sem sucesso. Após contato com os desenvolvedores da ferramenta foi possível constatar que o recurso realmente não estava funcionando adequadamente. Neste ponto, tomou-se a decisão de criar uma solução específica para a aplicação usada nos testes, tendo em vista o fato deste não estar no escopo do projeto e de não haver tempo hábil para analisar e implementar todo o mecanismo de geração do arquivo *checkpoint* e reinício da aplicação.

Para reiniciar a aplicação e não perder todo processamento já realizado anteriormente, tomou-se como base que as tarefas já completas não seriam executadas novamente. Assim, foi utilizado o arquivo *checkpoint* gerado pelo gerenciador *workflow* para criar um novo arquivo *xml*, considerando apenas as tarefas pendentes.

6.3 Resultados Experimentais

O cenário utilizado para testes do projeto envolve uma LAN *wireless* (WLAN), pela qual os dispositivos móveis acessam o ambiente *grid.* O dispositivo utilizado para os testes possui a seguinte configuração: Palm Tungsten C, processador 400 MHz, memória 64 MB e sistema operacional Palm OS 5.2.1. Cabe destacar que os mesmos recursos de hardware e software foram dedicados totalmente para a execução das abordagens testadas.

Os resultados obtidos com o mecanismo proposto envolveram as avaliações da confiabilidade do sistema e avaliação do consumo de energia do dispositivo móvel com a abordagem proposta.

6.3.1 Avaliação da Confiabilidade

A avaliação do sistema desenvolvido com o mecanismo de tratamento da falhas utilizou como critério a característica da confiabilidade, conforme a NBR ISO/IEC 9126-1 *apud* (Rocha, 2001). Segundo (Rocha, 2001), a confiabilidade é a capacidade do produto de software de manter um nível de desempenho especificado, quando usado em condições especificadas. Entre as subcaracterísticas da confiabilidade optou-se em utilizar na avaliação a tolerância a falhas e a recuperabilidade:

Tolerância a falhas: Capacidade do produto de software de manter um nível de desempenho especificado em casos de defeitos no software ou de violação de sua interface especificada.

Recuperabilidade: Capacidade do produto de software de restabelecer seu nível de desempenho especificado e recuperar os dados diretamente afetados no caso de uma falha.

Para estas subcaracterísticas da confiabilidade foram considerados os seguintes aspectos na avaliação do sistema:

- Ocorrendo falhas, o sistema reage de forma apropriada, conforme foi especificado?

- O sistema recupera estados de execução após ocorrer uma falha?
- O sistema alcança um estado de finalização da execução corretamente?

O primeiro teste do sistema foi realizado submetendo-se o *workflow 2 (Complete-Genome)* e utilizando-se os parâmetros da tabela 7 para definição do tempo do tempo de envio das mensagens de conexão. O *xml* do *workflow* é apresentado no apêndice D. Para analisar o comportamento do sistema diante de uma aplicação com dependência foi feita uma alteração no *workflow 2* para indicar tal situação. A alteração prevê a entrada de uma informação do usuário.

Tabela 7. Parâmetros do primeiro teste

Parâmetros	**Valor**
Configuração do usuário	Alta Prioridade
Aplicação:	Dependente
Tempo de vida da bateria	45%

O usuário utilizado para o teste foi *"agr"* que fez submissão da aplicação "*Genome Complete Workflow"*. O gerenciador *workflow* gerou um identificador para a submissão (*id=2*). Neste teste foi simulada uma desconexão do dispositivo desligando-o como se o problema fosse de término da bateria. Considerando que a aplicação leva em torno de 192 segundos (3min e 20 seg) desde a submissão até o aviso de finalização, a desconexão ocorreu em torno de 90 segundos após a submissão.

O observador identificou a desconexão e ativou o analisador. Este, por sua vez, fez a análise da propriedade *dependencia* do objeto *wf* que indicou a dependência do *workflow* em relação ao usuário móvel. Na seqüência o adaptador foi ativado e procedeu a parada da execução. O arquivo *checkpoint* neste momento possuía os dados ilustrados na figura 26. Neste caso, somente a tarefa com *uid* 17 ainda estava sendo executada, as demais tarefas já estavam finalizadas. Esta tarefa tem o maior tempo de execução, em torno de 160 segundos.

```
<?xml version="1.0" encoding="UTF-8"?>
<project number="agr/2" workflow_name="Complete-Genome">
<UID8 line="11">COMPLETED</UID8>
<UID9 line="12">COMPLETED</UID9>
<UID10 line="13">COMPLETED</UID10>
<UID12 line="15">COMPLETED</UID12>
<UID13 line="16">COMPLETED</UID13>
<UID15 line="19">COMPLETED</UID15>
<UID16 line="20">COMPLETED</UID16>
<UID17 line="21">RUNNING</UID17>
<UID18 line="22">COMPLETED</UID18>
</project>
```

Figura 26. Arquivo *xml* do *checkpoint*

No segundo teste realizado, novamente o usuário *"agr"* efetuou a submissão da mesma aplicação do primeiro teste (*Genome Complete Workflow).* Ao proceder a submissão o usuário recebeu uma mensagem de aviso conforme ilustra a figura 27. Nesta mensagem o usuário é comunicado que existe uma submissão anterior que não foi completada e pode ser reiniciada do ponto que havia parado. O usuário pode confirmar o reinício ou optar por uma nova submissão. O usuário confirmou o reinício da aplicação de identificador igual a 2.

Como mencionado anteriormente, não se obteve sucesso com o reinício de aplicações por meio do *checkpoint* gerado pelo *Karajan*. Nesta situação, o gerenciador analisou o *checkpoint* gerado no primeiro teste e fez adequações ao arquivo *xml* do *workflow* da aplicação submetida. Após este passo, procedeu-se a submissão ao *middleware Globus* para processar as tarefas não finalizadas. O processamento da aplicação foi finalizado com sucesso. É importante destacar que caso ocorresse o mesmo problema (desconexão) no segundo teste o sistema teria o mesmo comportamento, permitindo assim, vários reinícios da mesma aplicação.

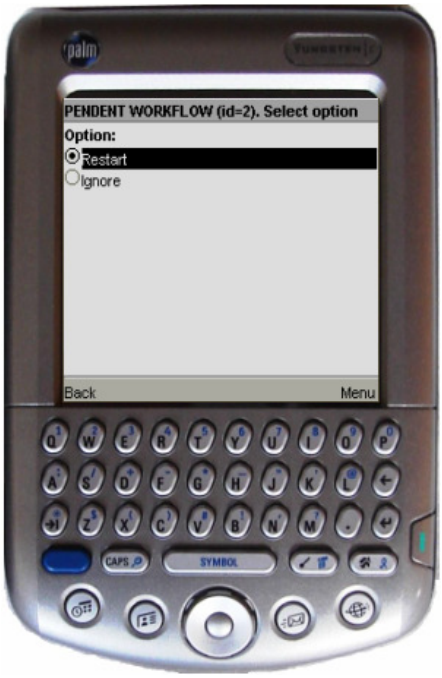

Figura 27. Mensagem de alerta sobre reinício de aplicação

Com os testes realizados foi possível avaliar como o sistema se comportou diante do problema da desconexão. Neste sentido, o sistema detectou a desconexão e tomou a decisão mais indicada para situação (parar para reinício posterior). Posteriormente, o sistema permitiu o reinício da aplicação do ponto que havia parado e finalizou a execução com sucesso.

6.3.2 Avaliação de consumo de energia

Prolongar o tempo de vida da bateria dos dispositivos móveis tem sido um dos problemas mais críticos e desafiadores na computação móvel (Rong; Pedram, 2003) (Mohapatra et al., 2005). Neste sentido, o projeto e desenvolvimento de aplicações para estes aparelhos necessitam considerar aspectos que possibilitem economizar energia. Corroborando com esse pensamento, a redução de acesso a rede sem fio proporciona uma menor dissipação de energia das baterias dos dispositivos móveis.

O trabalho de (Borges, 2006b) alcançou uma relativa economia de bateria quando comparada com a abordagem *"sem workflow"*. Todavia, neste trabalho, existia a

preocupação que a partir da implementação do mecanismo de detecção da desconexão, o consumo de energia da bateria fosse aumentado em função das mensagens enviadas para rede *wireless*. Diante desta situação, criou-se um mecanismo dinâmico para definição do tempo de envio das mensagens como mencionado anteriormente. Inicialmente, sem o desenvolvimento da rede *Bayesiana*, foram realizados testes utilizando dois intervalos de tempo, de vinte (20) e dez (10) segundos para envio das mensagens. Foi utilizado o *workflow 2 (Genoma Complete Workflow)* para os testes. Para medir o consumo da bateria, foi utilizado o software *BatteryGraph* versão 1.21 (Jeroen Witteman, 2003), compatível com a especificação *Palm Tungsten*. Para cada abordagem os testes iniciavam com a bateria do dispositivo com carga completa, a cada cinco submissões do *workflow*, o consumo de energia era verificado, até o término da bateria.

O gráfico da figura 28 apresenta os resultados obtidos nas abordagens:

- sem *workflow* : as tarefas foram submetidas uma a uma em ordem;
- com *workflow* na implementação de Borges;
- com *workflow* considerando um intervalo de 20 (vinte) segundos para envio das mensagens de conectado;
- com *workflow* considerando um intervalo de 10 (dez) segundos para envio das mensagens de conectado.

A partir dos resultados não se observou alteração no consumo de bateria do tempo de 20 (vinte) segundos em relação à abordagem de (Borges, 2006b), mantendo-se inclusive o número de execuções totais de 43 (quarenta e três). Porém quando analisado o tempo de 10 (dez) segundos, verificou-se que a tendência de um maior gasto de bateria, o que ocasionou em 3 (três) submissões a menos (40).

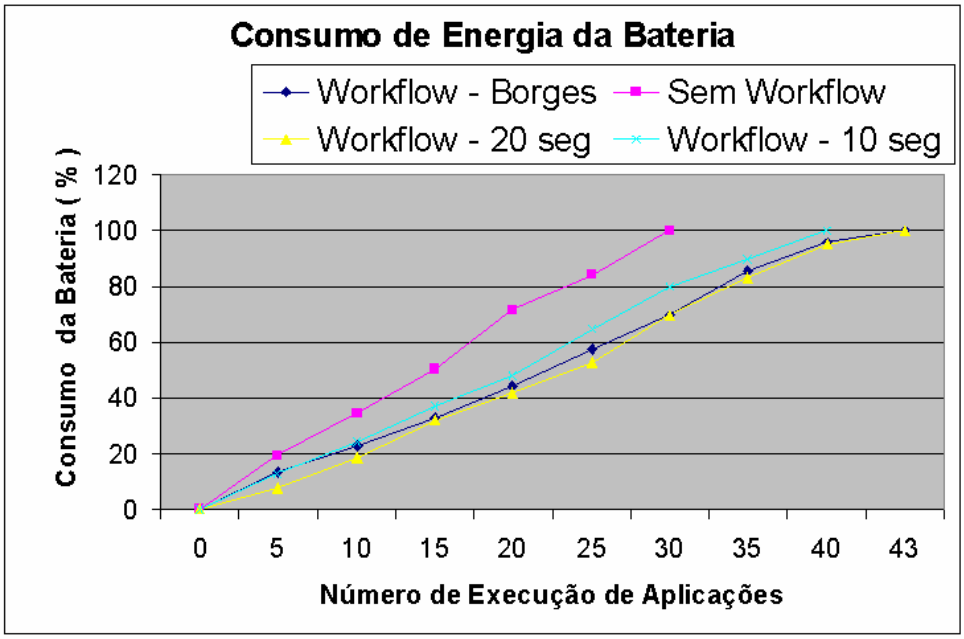

Figura 28. Gasto de energia da bateria

Na análise dos resultados obtidos e durante a implementação, observou-se que algumas alterações poderiam ser realizadas no modo de monitoração desenvolvido por (Borges, 2006b) para obter menor gasto de energia. Na abordagem de (Borges, 2006b), quando o usuário móvel solicitava o status da aplicação, o gerenciador *workflow* informava o *status* da aplicação enviando mensagens a cada 8 (oito) segundos, sendo uma mensagem para cada tarefa da aplicação. Na abordagem deste trabalho, alterou-se a forma de envio, passando a ser uma mensagem única contendo os status de todas as tarefas e a mensagem só é enviada quando ocorre alteração do status de pelo menos uma tarefa.

Após a implementação da abordagem deste trabalho, novos testes foram realizados considerando os tempos de 8 (oito) e 24 (vinte e quatro) segundos para envio das mensagens. Neste ponto não foi considerado o tempo de vida da bateria para definir o tempo de envio das mensagens. Novamente, para cada um dos tempos os testes foram iniciados com carga completa para a bateria do dispositivo, fazendo a coleta do consumo de energia a cada cinco submissões.

O gráfico da figura 29 ilustra o consumo de bateria para os testes realizados com 8 (oito) e 24 (vinte e quatro) segundos a partir da nova abordagem. Neste gráfico o

consumo de bateria é apresentado com base na voltagem (*mV=milivolt*).

Observamos que com o uso de 24 (vinte e quatro) segundos houve menor gasto de bateria, quando comparado com o tempo de 8 (oito) segundos. Embora não seja uma economia tão expressiva, quando analisando estes dois elementos apenas, a economia gerada possibilitou que mais 3 (três) submissões fossem realizadas. Contudo, uma importante evidência da abordagem testada é o número de submissões totais (45 e 48), que ficou acima da abordagem de (Borges, 2006b), mesmo considerando o tempo de 8 (oito) segundos para envio das mensagens. Muito embora, a diferença expressa pelo gráfico não seja tão expressiva, toda economia gerada nestes ambientes é importante, tendo em vista a necessidade de prolongar o tempo de vida da bateria dos dispositivos sem perder as funcionalidades do sistema. Estes dados indicam que a inclusão do Módulo TF e as alterações nos módulos da abordagem de Borges (2006b) não incidiram em aumento do consumo de energia, pelo contrário, alcançaram uma maior economia.

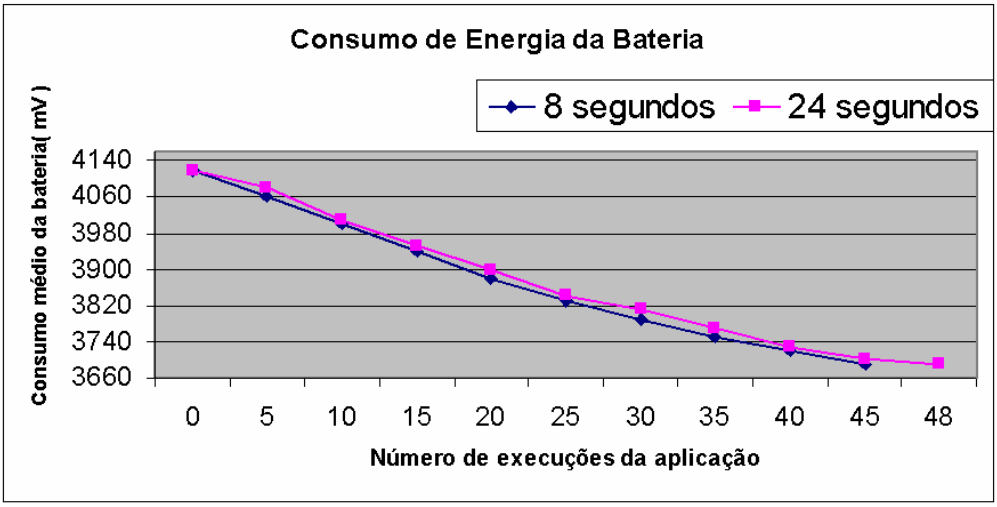

Figura 29. Gasto de energia da bateria

6.4 Considerações dos resultados experimentais

Para obtenção dos resultados do mecanismo de tratamento de falhas proposto e implementado duas avaliações foram realizadas: avaliação da confiabilidade e avaliação

do consumo de energia dos dispositivos. Os resultados demonstraram que a aplicação detectou a desconexão, notificou, adaptou a execução e possibilitou o reinício posterior sem perder o processamento já realizado. Desta forma, alcançou um estado de execução e finalização correto da aplicação. Além disso, foi avaliado o consumo de energia da bateria do dispositivo a fim de verificar se o mecanismo que estendeu a abordagem de Borges (2006b) não gerava maior consumo energia e se os tempos diferenciados de envio das mensagens geravam diferenças no consumo de energia. Os resultados apontaram que o mecanismo alcançou um menor consumo de energia da bateria dos dispositivos.

7 Considerações Finais e Trabalhos Futuros

À medida que novas tecnologias surgem na área de sistemas distribuídos, quebrando formatos tradicionais de comunicação e interação, mais desafios são impostos para implementação de sistemas computacionais que utilizem eficientemente os benefícios advindos de tais tecnologias. De fato, ao observarmos as características inerentes da computação móvel e dos *grids* computacionais, verifica-se que a integração das tecnologias é potencial. Entretanto, é conhecido que ambas as tecnologias possuem limitações que não podem ser ignoradas nos projetos e desenvolvimento de sistemas.

Com o intuito de abordar os aspectos relacionados às falhas inerentes destes ambientes, muitas vezes, não considerados nos projetos de sistemas computacionais, a abordagem proposta neste trabalho de dissertação tratou a desconexão de dispositivos móveis que submetem aplicações com múltiplas tarefas para um ambiente *grid*. O presente trabalho de pesquisa é uma extensão da arquitetura de (Borges, 2006b) que permite a submissão e monitoração de aplicações em *grids* computacionais, usando o conceito de *workflow*. (Borges, 2006b) evidenciou que sua abordagem proporcionou um diferencial para a submissão de várias tarefas de forma organizada, controlada e automatizada, tendo em vista algumas prerrogativas tais como maior agilidade na submissão e monitoração de várias tarefas cooperantes e menor gasto de energia da bateria dos equipamentos móveis na submissão.

O Módulo TF proposto neste trabalho foi projetado e implementado com três componentes: o observador, o analisador e o adaptador. Tais componentes possibilitaram verificar o status de conexão e, se necessário, ajustar a execução da aplicação de acordo com os seus requisitos e configurações do usuário móvel. Neste sentido, grifam-se aplicações que possuem algum tipo de dependência do usuário móvel, ou ainda, aplicações, das quais a monitoração contínua é extremamente relevante para o usuário. Com este mecanismo é possível garantir a consistência das aplicações de

maneira transparente para o usuário, além de não desperdiçar recursos da *grid*. Em adição, utilizando um mecanismo de *checkpoint* para aplicações que são interrompidas, é possível reiniciar a execução do ponto que havia parado, sem perder todo o processamento já efetivado.

Desta forma, o protótipo desenvolvido utilizou uma rede *bayesiana* para determinar o tempo de observação acerca da desconexão, considerando o tempo de vida da bateria dos dispositivos, as características da aplicação e as prioridades do usuário. Os resultados experimentais indicaram que a arquitetura proposta permitiu consumir menos energia de dispositivos móveis para submeter e monitorar aplicações, tanto em relação à abordagem sem *workflow*, quanto à abordagem com *workflow*. Do mesmo modo, observou-se, uma redução no consumo de energia quando o tempo de envio das mensagens para verificação da conexão é maior. Neste ponto, destaca-se a importância de um mecanismo dinâmico para definição do tempo de verificação da conexão.

Até o momento do fechamento deste texto, não foi possível implementar no sistema a recuperação da carga de bateria do dispositivo, tendo em vista que a linguagem *j2me* não possui acesso às funções nativas dos dispositivos. Para realizar os testes, foi informada para o sistema a carga da bateria do dispositivo em termos de percentual. No entanto, uma solução já está em estudo e prevê a utilização da tecnologia *JNI (Java Native Interface)*(Sun Microsystems, 2007d) para integrar o *j2me* com aplicações criadas em outras linguagens, neste caso com *C++*, linguagem da API nativa do dispositivo usado para os testes.

Como trabalhos futuros, objetiva-se avançar a presente pesquisa em alguns aspectos: a) alterar a rede *bayesiana*, que define o intervalo de tempo para verificação do estado de desconexão, para considerar outros parâmetros de entrada, por exemplo, volume de tráfego na rede sem fio; b) realizar testes acerca do volume de tráfego na rede com os diferentes tempos de monitoramento e vários dispositivos acessando a rede; c) fazer testes com outros *workflows*; d) realizar testes com outros modelos de dispositivos; e) empregar esforços para alcançar outra solução a respeito do *checkpoint*; f) empregar mecanismos para fornecer uma comunicação mais segura (por exemplo,

SSL); g) recuperar a medida de tempo de vida da bateria no sistema *GUI Móvel*; f) efetuar experimentos com uma configuração de *grid* “completa”.

Referências

AALST, Wil M. P. van der; BARROS, Alistair P.; HOFSTEDE, Arthur H. M. ter; KIEPUSZEWSKI, Bartek. *Advanced workflow patterns*. In: CooplS '02: Proceedings of the 7th International Conference on Cooperative Information Systems. London, UK: Springer-Verlag, 2000. p. 18–29. ISBN 3-540-41021-X.

ALLIANCE, Globus. About the Globus Alliance. 2007. <http://www.globus.org/, acessado em: 23/04/2007>.

AVIZIENIS, Algirdas; LAPRIE, Jean-Claude; RANDELL, Brian. *Fundamental Concepts of Denpendability.* Research Report NO1145, LAAS-CNRS, April 2001.

BAKER, Mark; BUYYA, Rajkumar; LAFORENZA, Domenico. T*he Grid: International Efforts in Global Computing*. International Conference on Advances in Infrastructure for Electronic Business, Science, and Education on the Internet (SSGRR 2000), l`Aquila, Rome, Italy, July 31 - August 6. 2000.

BORGES, V. C. M; DANTAS, M. A. R. *Uma abordagem de submissão e monitoração de múltiplas tarefas para ambientes de grade computacional utilizando dispositivos móveis*. XXXIII SEMISH, 2006, pages 403–418.

BORGES, V.C.M. *Uma Abordagem de Submissão e Monitoração de Múltiplas Tarefas para Ambientes de Grade Computacional Utilizando Dispositivos Móveis*. Dissertação (Mestrado) – Universidade Federal de Santa Catarina (UFSC), 2006.

BROOKE, John; PARKIN, Michael. A pda client for the computational grid. In: *WETICE '05: Proceedings of the 14th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprise*. Washington, DC, USA: IEEE Computer Society, 2005. p.325–330. ISBN 0-7695-2362-5.

BRUNEO, Dario; SCARPA, Marco; ZAIA, Angelo; PULIAFITO, Antonio. Communication paradigms for mobile grid users. *3rd (CCGrid'03)*, p. 669–676, May 2003.

BUYYA, Rajkumar; ABRAMSON, David; VENUGOPAL, Srikumar. *The grid economy*. Proceedings of the IEEE, v.93, n.3, p.698–714, March 2005.

CANNATARO, Mario; COMITO, Carmela; GUZZO, Antonella; VELTRI, Pierangelo. Integrating ontology and workflow in proteus, a grid-based problem solving environment for bioinformatics. In: *ITCC* (2). [S.l.: s.n.], 2004. p. 90–94.

CHEN, Guanling; KOTZ, David. *A Survey of Context-Aware Mobile Computing Research*. Dartmouth College, Hanover, NH, 2000.

CHU, David C.; HUMPHREY, Marty. Mobile ogsi.net: Grid computing on mobile devices. In: *GRID '04: Proceedings of the Fifth IEEE/ACM International Workshop on Grid Computing (GRID'04)*. Washington, DC, USA: IEEE Computer Society, 2004. p. 182–191. ISBN 0-7695-2256-4.

CLARKE, B; HUMPHREY, M. Beyond the 'device as portal': Meeting the requirements of wireless and mobile devices in the legion grid computing system. *2nd International Workshop on Parallel and Distributed Computing Issues in Wireless Networks and Mobile Computing (associate with IPDPS 2002)*, p. 192–199, Abril 2002.

CONCEIÇÃO, Arlindo F.; KON, Fabio. O uso de pares de pacotes para monitoração da taxa de transmissão e da capacidade de vazão em redes IEEE 802.11. In 24th *Brazilian Symposium on Computer Networks (SBRC)*, pages 345–358, Curitiba-PR, Maio 2006.

COULOURIS, G.; DOLLIMORE, J.; KINDBERG, T. Sistemas Distribuídos: conceitos e projeto.[S.l.]:Bookman, 2007. ISBN 978-85-60031-49-8.

DANTAS, Mario. *Computação Distribuída de Alto Desempenho, Redes, Clusters e Grids Computacionais*. [S.l.]: Axcel Books do Brasil Editora, 2005. ISBN 85-7323-169-6.

FERREIRA, D. J.; DANTAS, M.A.R.; PINTO, A. R.; MONTEZ, C.; RODRIGUEZ, Martius. High Performance Computing Systems and Applications. *In HPCS 2007. 21st International Symposium,* May 2007.

FORMAN, George; ZAHORJAN, John. *The Challenges of Mobile Computing*. IEEE Computer, vol. 27, n. 4, p. 38-47, Apr., 1994.

FOSTER, Ian; KESSELMAN, Carl. *Globus: A Metacomputing Infrastructure Toolkit*. International Journal of Supercomputer Applications, 1997.

FOSTER, Ian; KESSELMAN, Carl. *The Grid: Blueprint for a New Computing Infrastructure*. [S.l.]: Morgan Kaufmann, 1999. ISBN 1-55860-475-8.

FOSTER, Ian; KESSELMAN, Carl; TUECKE, Steven. The anatomy of the grid: Enabling scalable virtual organizations. *International Journal of Supercomputer Applications*, v. 3, n. 15, 2001.

FOSTER, Ian; KESSELMAN, Carl; NICK, Jeffrey M.; TUECKE, Steven. *The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration*. Open Grid Service Infrastructure WG, Global Grid Forum, June 22, 2002.

FOSTER, Ian; KESSELMAN, Carl. *The Grid 2: Blueprint for a new Computing Infrastructure*. [S.l.]: John Wiley and Sons Ltd, 2003. ISBN 1-558-60933-4.

FOSTER, Ian. Globus toolkit version 4: Software for service-oriented systems. *IFIP International Conference on Network and Parallel Computing*, p. 2–13, 2005.

GIL, Antonio Carlos. *Como elaborar projetos de pesquisa*. São Paulo: Atlas, 1991.

GLOBUS. *The Globus Toolkit*. 2006. Http://www.globus.org/ e acessado em: 02/03/2006.

GOLDCHLEGER, A.; FON, F.; GOLDMAN, vel Lejbman, A., FINGER, M.; SONG, S. W. *Integrade: Rumo a um sistema de computação em grade para aproveitamento de recursos ociosos em máquinas compartilhadas*. Technical Report RT-MAC-2002-08, Departamento de Ciência da Computação, Instituto de Matemática e Estatística, Universidade de São Paulo, Outubro, 2002.

GONZALEZ-CASTANO, F; VALES-ALONSO, J; LIVNY, M. Condor grid computing from mobile handheld devices. *ACM SIGMOBILE Mobile Computing and Communications Review*, v. 7, n. 1, p. 117–126, Janeiro 2003.

HOLLINGSWORTH, D. *Workflow management coalition. Reference model and api specification*. WfMC-TC00-1003, 1996.

HOSKEN, A; DANTAS, Mario A. R. *The ATHA Environment: Experience with a User Friendly Environment for Opportunistic Computing*. HPCS, 2004, p. 85-88.

HUMMEL, K. A; BOHS, G; BREZANY, P; JANCIAK, I. *Mobility extensions for knowledge discovery workflows in data mining grids*. DEXA, 0:246–250, 2006.

KURKOVSKY, Stan; BHAGYAVATI, Arris Ray; YANG, Mei. Modeling a grid-based problem solving environment for mobile devices. *(ITCC'04)*, v. 2, n. 2, p. 135–136, Abril 2004.

IBM Software. *WebSphere Everyplace Micro Environment*. 2007. Disponível em: <http://www-306.ibm.com/software/wireless/weme/, acessado em: 22/03/2007>.

IMIELINSKI, T.; BADRINATH, B. R. *Data Management for Mobile Computing*. In Sigmod Record, Vol. 22, No. 1, p. 34-39, 1993.

IMIELINSKI, T.; VISWANATHAN, S.; BADRINATH, B. R. *Energy efficient indexing on air*. In: Proceedings of the 1994 ACM Sigmod International Conference on Management of Data, 1994. ACM Press. p.25–36.

JALOTE, Pankaj. *Fault tolerance in distributed systems*. Prentice Hall, Englewood Cliffs, New Jersey, 1994.

Jeroen Witteman. BatteryGraph Overview. 2003. Disponível em: <http://palm.jeroenwitteman.com/BatteryGraph/, acessado em: 19/10/2006>.

JING, Jin; Helal, ABDELSALAM S.; ELMAGARMID, Ahmed. *Client-server computing in mobile environments*. In ACM Computing Surveys, v.31, n.2, p.117–157, 1999.

LAKATOS, Eva Maria; MARCONI, Marina de Andrade. Metodologia Científica. São Paulo: Atlas, 2000.

LAPRIE, J-C. *Dependability of Computer Systems: From Concepts to Limits*. In: IFIP International Workshop on Dependable Computing and its Applications (DCIA 98). Johannesburg, 1998.

LASZEWSKI, G Von; FOSTER, Ian; GAWOR, J; LANE, P. A java commodity grid kit. *Concurrency and Computation: Practice and Experience*, v. 13, n. 8-9, p. 643–662, 2001.

LASZEWSKI, Gregor; HATEGAN, Mike. Workflow concepts of the java cog kit. *Journal of Grid Computing*, Springer Netherlands, v. 3, n. 3-4, p. 239–259, 2005. ISSN 0010-4620

LASZEWSKI, Gregor Von; HATEGAN, Mike. *Grid Workflow - An Integrated Approach*. Argonne, IL, USA, 2005. Disponível em: <Disponível em: http://www.mcs.anl.gov/ gregor/papers/vonLaszewski-workflow-draft.pdf e acessado em 01/05/2007>.

LEMOS, Melissa. *Workflow para Bioinformática*. Dissertação (Mestrado) — Pontifícia Universidade Católica do Rio de Janeiro (PUC-Rio), 2004. Disponível em: <www.inf.pucrio.br/Melissa/publicacao/download/tese_melissa/Tese_Melissa_Lemos.pdf>.

LITKE, A.; SKOUTAS, D.; VARVARIGOU, T. Mobile grid computing: Changes and challenges of resource management in a mobile grid environment. *PAKM 2004 Conference*, 2004.

KOURGANOFF, Wladimir. *A face oculta da universidade*. Tradução Cláudia Schilling; FátimaMurad. São Paulo : Editora da Universidade Estadual paulista, 1990.

MATEUS, G. R.; LOUREIRO, A. A. F. Introdução à Computação Móvel. 11ª Escola de Computação COPPE/Sistemas. Disponível em: <http://www.ime.usp.br/ brito/mac5743/, Acessado em 01/05/2007>

MOHAPATRA, Shivajit; CORNEA, Radu; OH, Hyunok; LEE, Kyoungwoo; KIM, Minyoung; DUTT, Nikil D.; GUPTA, Rajesh; NICOLAU, Alexandru; SHUKLA, Sandeep K.; VENKATASUBRAMANIAN, Nalini. A cross-layer approach for power-performance optimization in distributed mobile systems. In: *IPDPS*. [S.l.: s.n.], 2005.

NASSAR, Silvia M. *Sistemas Especialistas Probabilísticos*. Notas de Aula, In: Curso de Pós-Graduação em Ciência da Computação, UFSC, 69p, 2003.

OGF. OPEN GRID FORUM. Disponível em <http://www.ogf.org e acessado

02/08/2007>

OINN, Tom; ADDIS, Matthew; FERRIS, Justin; MARVIN, Darren; SENGER, Martin; GREENWOOD, Mark; CARVER, Tim; GLOVER, Kevin; POCOCK, Matthew R.; WIPAT, Anil; LI, Peter. Taverna: a tool for the composition and enactment of bioinformatics workflows. *Bioinformatics*, Oxford, UK, v. 20, n. 17, p. 3045–3054, 2004. ISSN 1367-4803.

PARK, Sang-Min.; KO, Young-Bae; KIM, Jai-Hoon. Disconnected operation service in mobile grid computing. In: *ICSOC*. [S.l.: s.n.], 2003. p. 499–513.

PERRY, Mark; O'HARA, Kenton; SELLEN, Abigail; BROWN, Barry; HARPER, Richard. *Dealing with mobility: understanding access anytime, anywhere*. ACM Trans. Comput.-Hum. Interact., [S.l.], v.8, n.4, p.323–347, 2001.

PITOURA, E.; BHARGAVA, B. *Building information systems for mobile environments*. In: Third International Conference on Information and Knowledge Management, 1994. [s.n.].

PITOURA, E.; SAMARAS, G. *Data management for mobile computing*. In: Kluwer Academic Publishers, 1998. [s.n.]. v.10.

PHAN, T; HUANG, L; DULAN, C. Challenge: Integrating mobile wireless devices into the computational grid. *8th (MOBICOM'02)*, p. 271–278, Setembro 2002.

RISTA, C; DANTAS, Mario A. R. *A Wireless Monitoring Approach for a HA-OSCAR Cluster Environment*. HPCS, 2005, p.302-306.

ROCHA, Ana Regina Cavalcanti. *Qualidade de software: teoria e prática*. São Paulo: Prentice Hall, 303 p. ISBN 8587918540, 2001.

RONG, Peng; PEDRAM, Massoud. Extending the lifetime of a network of battery-powered mobile devices by remote processing: a markovian decision-based approach. In: *DAC '03: Proceedings of the 40th conference on Design automation*. New York, NY, USA: ACM Press, 2003. p. 906–911. ISBN 1-58113-688-9.

ROSSETTO, Anubis G. M; DANTAS, Mario A. R. *Um agente para tratamento da desconexão de dispositivos móveis utilizadores de recursos de grid computacional*. In Escola Regiona de Alto Desempenho (ERAD), 2006.

ROSSETTO, A.G.M; BORGES, V.C.M; DANTAS, M.A.R; DIAS, J.S. *SuMMIT - An Architecture for Mobile Devices to Coordinate the Execution of Multiple Tasks in Grid Environments*. In WETICE 2007 - 16th IEEE International Workshops on Enabling Technologies: Infrastructures for Collaborative Enterprises, 2007.

ROSSETTO, A.G.M; BORGES, V.C.M; DANTAS, M.A.R; KNAESEL, J.S. *An Enhanced Approach for Submission and Monitoring of Applications in the Mobile Grid*. Fifth International Symposium on Parallel and Distributed Processing and Applications

(ISPA-07), 2007.

ROURE, D. De; BAKER, M.A; Jennings, N.R; Shadbolt, N.R. *The evolution of the grid*. In: Berman, F. e. (Ed.). Grid computing: making the global infrastructure a reality. New York, USA: John Wiley, p.65-100, 2003.

RUSSEL, Stuart J.; NORVING, Petter. *Inteligência Artificial*. Tradução da segunda edição. Rio de Janeiro, Elsevier, 2004.

SAIRAMESH, J.; GOH, S.; STANOI, I.; PADMANABHAN, S.; LI, C. S. Disconnected processes, mechanisms and architecture for mobile e-business. *Mobile Network Application*, Kluwer Academic Publishers, Hingham, MA, USA, v. 9, n. 6, p. 651–662, 2004. ISSN 1383-469X.

SATYANARAYANAN, M. *Fundamental Challenges in Mobile Computing*. In: Symposium on Principles of Distributed Computing, p.1–7, 1996.

SATYANARAYANAN, M. *Pervasive computing: vision and challenges*. [S.l.], v.8, n.4, p.10–17, 2001.

SHAHAB, Atif; CHUON, Danny; SUZUMURA, Toyotaro; LI, Wilfred W.; BYRNES, RobertW.; TANAKA, Kouji; ANG, Larry; MATSUOKA, Satoshi; BOURNE, Philip E.;MILLER, Mark A.; ARZBERGER, Peter W. *Grid portal interface for interactive use and monitoring of high-throughput proteome annotation*. In: *LSGRID*. [S.l.: s.n.], 2004. p. 53–67.

SILVA, Edna Lúcia; MENEZES, Estera Muszkat. *Metodologia da pesquisa e elaboração de dissertação.* 3ª Ed. rev. Florianópolis: UFSC/PPGEP/LED, 2001.

Sun Microsystems. *J2EE(TM) 1.4 Tutorial*. 2007. Disponível em: <http://java.sun.com/j2ee/1.4/docs/tutorial/doc/, acessado em: 10/05/2007>.

Sun Microsystems. *Java Platform, Micro Edition (Java ME)*. 2007. Disponível em: <http://java.sun.com/javame/index.jsp, acessado em: 10/05/2007>.

Sun Microsystems. *Java Platform, Standard Edition (Java SE)*. 2007. Disponível em:<http://java.sun.com/javase/index.jsp, acessado em: 10/05/2007>.

Sun Microsystems. *Java Native Interface (JNI)*. 2007. Disponível em:<http://java.sun.com/j2se/1.4.2/docs/guide/jni/, acessado em: 22/05/2007>.

THAIN, Douglas; TANNENBAUM, Todd; LIVNY, Miron. Distributed computing in practice: the condor experience. *Concurrency - Practice and Experience*, v. 17, n. 2-4, p. 323–356, 2005.

TUECKE, S.; CZAJKOWSKI, K.; FOSTER, I.; REY, J.; STEVE, F.; CARL, G. *Open Grid Service Specification*. 2002. Disponível em: <http://www.globus.org/toolkit/draft-ggf-ogsi-gridservice-33_2003-06-27.pdf e acessado em 02/08/2007>.

W3C. *World Wid Web Consortium.* 2007. Disponível em: <http://www.w3c.org, acessado em 29/07/2007>

WEBER, Taisy S. *Um roteiro para exploração dos conceitos básicos de tolerância a falhas.* 2002. Disponível em: <http://www.inf.ufrgs.br/~taisy/disciplinas/textos/Dependabilidade.pdf, acessado em 01/05/2007>.

WSRF. *Globus: WS Resource Framework*. 2004. Disponível em: <http://www.globus.org/wsrf/ e acessado em 03/05/2007>.

YAMIN, A. Arquitetura para um Ambiente de Grade Computacional Direcionado às Aplicações Móveis, Distribuídas e Conscientes do Contexto da Computação Pervasiva. Tese de Doutorado. Instituto de Informática/UFRGS, Porto Alegre, agosto, 2004.

YU, J; BUYYA, R. A taxonomy of workflow management systems for grid computing. *(SIGMOD' 05)*, Chicago, IL, USA, v. 34, n. 3, p. 44–49, July 2005.

Apêndice A – Publicações

Título: Um agente para tratamento da desconexão de dispositivos móveis utilizadores de recursos de grid computacional
Evento: Escola Regional de Alto Desempenho - ERAD
Local: Ijuí-RS
Data: 10/01/2006 a 14/01/2006
Autores: Anubis Graciela de Moraes Rossetto, Mario Antônio Ribeiro Dantas

Título: SuMMIT - An Architecture for Mobile Devices to Coordinate the Execution of Multiple Tasks in Grid Environments
Evento: Fourth International Workshop on Emerging Technologies for Next-generation GRID (ETNGRID 2007)
Local: Paris - França
Data: 18/06/2007 a 20/06/2007
Autores: Anubis Graciela de Moraes Rossetto, Vinicius da Cunha Martins Borges, Mario Antônio Ribeiro Dantas
Qualis CC: C Internacional

Título: An Enhanced Approach for Submission and Monitoring of Applications in the Mobile Grid
Evento: Fifth International Symposium on Parallel and Distributed Processing and Applications (ISPA-07)
Local: Niagara Falls - Canadá
Data: 29/08/2007 a 31/08/2007
Autores: Anubis Graciela de Moraes Rossetto, Vinicius da Cunha Martins Borges, Mario Antônio Ribeiro Dantas
Qualis CC: A Internacional

Título: SuMMIT - A Framework for Coordinating Applications Execution in Mobile Grid Environments

Evento: The 8th IEEE/ACM International Conference on Grid Computing (Grid 2007)

Local: Austin, Texas

Data: 19/09/2007 a 21/09/2007

Autores: Anubis Graciela de Moraes Rossetto, Vinicius da Cunha Martins Borges, Alexandre Parra Carneiro da Silva, Mario Antônio Ribeiro Dantas

Qualis CC: A Internacional

Apêndice B – Diagramas da arquitetura de Borges (2006b)

A figura 30 apresenta o diagrama de classes do Gerenciador *Workflow* da abordagem de Borges (2006b). As classes que fazem parte do diagrama possuem as seguintes atribuições:

Controlador - recebe os pedidos de conexão dos dispositivos móveis, cria um identificador e instancia um *ControladorThread* para cada pedido de conexão. A partir deste momento a comunicação é realizada com o *ControladorThread*.

ControladorThread - recebe os pedidos de submissão e monitoração do dispositivo. Quando recebe um pedido de submissão de *workflow* instancia o Motor;

Motor - esta classe executa o *workflow*. Ele usa as classes *Loader* e *ElementTree* do motor *Karajan* para executar os *workflows*. Após a submissão o coletor é instanciado;

ColetorSeqSearch - coletor de um *workflow* específico (*Sequence Workflow*). Extrai os status de cada tarefa do *worklfow*;

ColetorComGem - coletor de um *workflow* específico (*Genoma Complete Workflow*) . Extrai os status de cada tarefa do *worklfow*;

DOM - esta classe cria o documento *xml* que contém os status das tarefas;

XMLParser - faz o *parser* do arquivo *xml*.

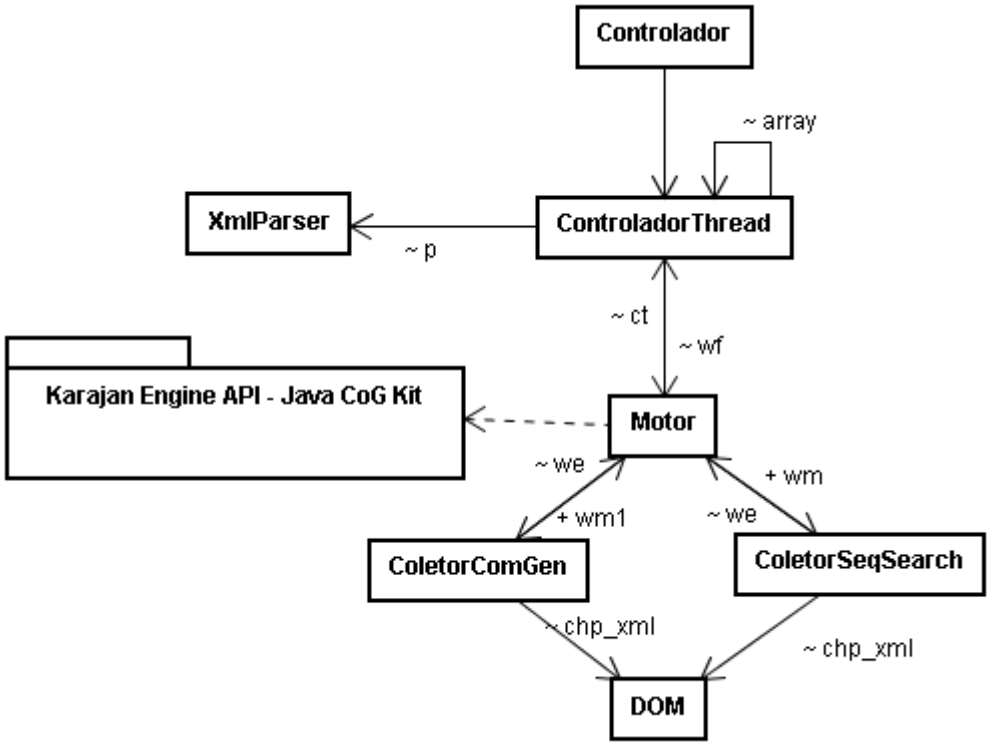

Figura 30. Diagrama de classes do Gerenciador *Workflow*

A figura 31 apresenta o diagrama de classes do GUI Móvel da abordagem de Borges (2006b). As classes que fazem parte do diagrama possuem as seguintes atribuições:

Cliente - contém uma lista dos *workflow* para submissão e monitoração;

DisplayManager - gerencia diversos displays da GUI Móvel;

WfDataComGen - mostra resultados ou problemas de tarefas específicas;

WfDataSeqSearch - mostra resultados ou problemas de tarefas específicas;

WfExecutionTime - mostrar o tempo de execução do *workflow;*

WfDescription - mostra a descrição do *workflow;*

InterfaceWfSeqSearch - classe que faz a submissão e monitoramento do *Sequence Search Workflow;*

InterfaceWfComGen - classe que faz a submissão e monitoramento do *Complete Genome workflow*;

SubWfComGen - classe que desenha parte do *workflow*.

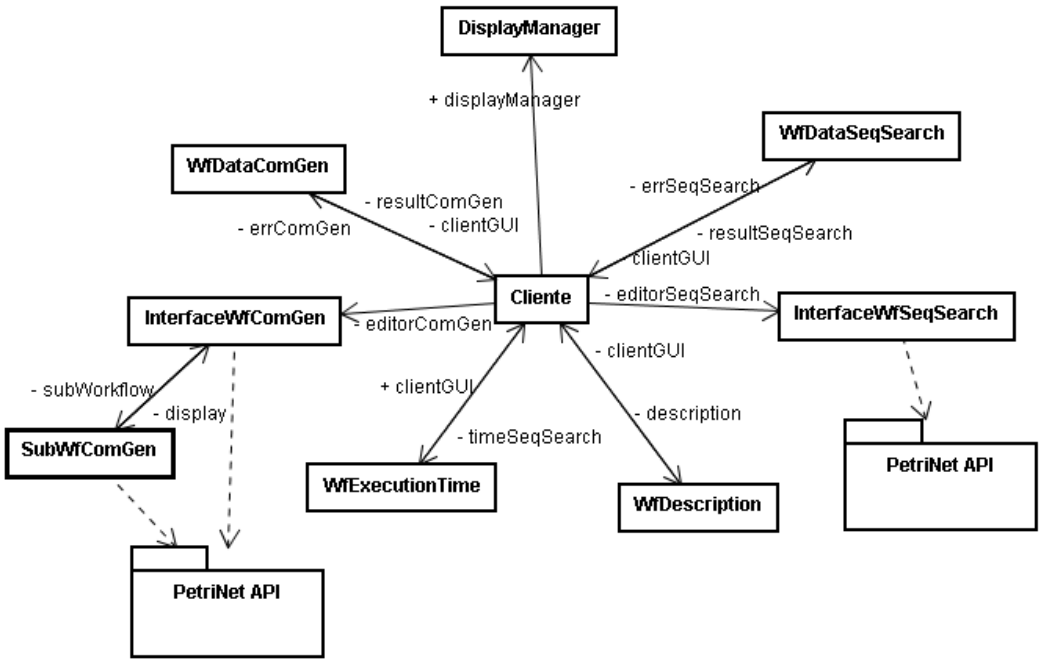

Figura 31. Diagrama de classes do GUI Móvel

Apêndice C – Diagrama de Classes

A figura 32 apresenta o diagrama de classes da abordagem proposta com as suas propriedades e métodos. As classes criadas para o módulo TF são apresentadas em cor azul.

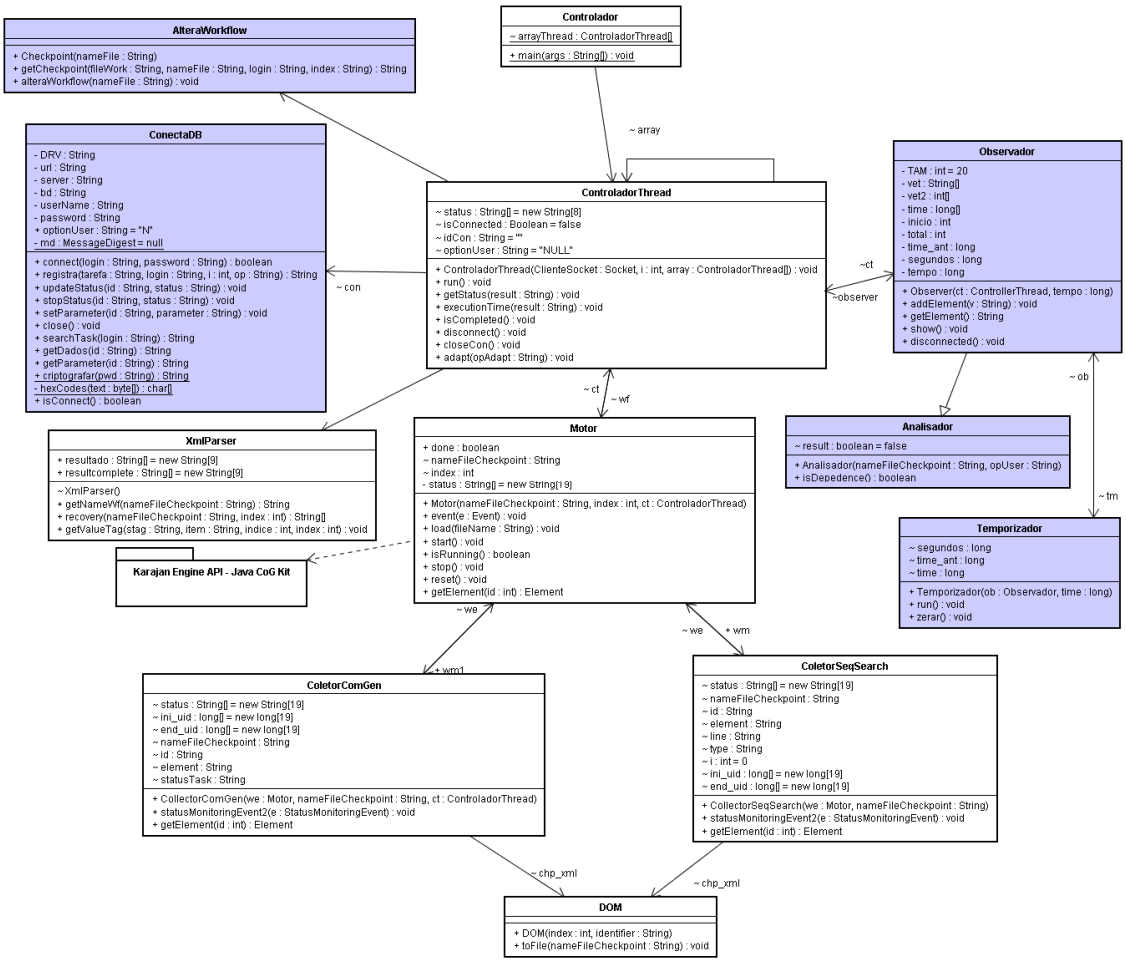

Figura 32. Diagrama de classes com o Módulo TF

Apêndice D – *Script Workflow*

A figura 33 apresenta o *script workflow* especificado na linguagem Karajan. O *workflow* possui sete tarefas que tem como objetivo analisar a bactéria *Gluconacetobacter diazotrophicus* do projeto Genoma (Lemos, 2004).

```
<project>
     <include file="sys.xml"/>
     <include file="task.xml"/>
     <set name="chromatdir" value="/mnt/win/">
     <set name="teste">
          <host name="teste">
               <service type="execution" provider="gt4" url="
               https://labweb02.inf.ufsc.br:8443/wsrf/services/ManagedJobFa
               ctoryService" jobManager="Fork"/>
          </host>
     </set>
        <execute executable="/mnt/win/phred/genome/workflow/clean.exe"
        host="{teste}" provider="gt4" arguments=""
        stdout="/mnt/win/phred/genome/workflow/clean.out"
        stderr="/mnt/win/phred/genome/workflow/clean.err"/>

        <execute executable="/mnt/win/phred/genome/bin/phred"
        host="{teste}" provider="gt4" arguments="{chromatdir} -pd
        /mnt/win/phred/genome/phd_dir/"
        stdout="/mnt/win/phred/genome/workflow/phred.out"
        stderr="/mnt/win/phred/genome/workflow/phred.err"/>

        <execute executable="/mnt/win/phred/genome/bin/phd2fasta"
        host="{teste}" provider="gt4" arguments="-id
        /mnt/win/phred/genome/phd_dir/ -os
        /mnt/win/phred/genome/workflow/seqs_fasta -oq
        /mnt/win/phred/genome/workflow/seqs_fasta.screen.qual"
        stdout="/mnt/win/phred/genome/workflow/phd2fasta.out"
        stderr="/mnt/win/phred/genome/workflow/phd2fasta.err"/>

        <execute executable="/mnt/win/phred/genome/bin/phrap"
        host="{teste}" provider="gt4"
        arguments="/mnt/win/phred/genome/workflow/seqs_fasta -minmatch 20
        -new_ace" stdout="/mnt/win/phred/genome/workflow/phrap.out"
        stderr="/mnt/win/phred/genome/workflow/phrap.err"/>
        <parallel>
          <execute executable="/mnt/win/glimmer2.13/run-glimmer2"
          host="{teste}" provider="gt4"
          arguments="/mnt/win/phred/genome/workflow/seqs_fasta.contigs"
          stdout="/mnt/win/phred/genome/workflow/run-glimmer2.out"
          stderr="/mnt/win/phred/genome/workflow/run-glimmer2.err"/>

          <execute executable="/mnt/win/phred/genome/bin/tRNAscan-SE"
          host="{teste}" provider="gt4"
          arguments="/mnt/win/phred/genome/workflow/seqs_fasta.contigs"
          stdout="/mnt/win/phred/genome/workflow/tRNAscan-SE.out"
          stderr="/mnt/win/phred/genome/workflow/tRNAscan-SE.err"/>
        </parallel>
        <parallel>
          <execute executable="/mnt/win/phred/genome/bin/TransTerm"
          host="{teste}" provider="gt4" arguments="-s
          /mnt/win/phred/genome/workflow/seqs_fasta.contigs -c
          /mnt/win/phred/genome/workflow/tmp.train -o
          /mnt/win/phred/genome/workflow/transterm.out"
          stdout="/mnt/win/phred/genome/workflow/TransTerm.out"
```

```
        stderr="/mnt/win/phred/genome/workflow/TransTerm.err"/>
        <execute executable="/mnt/win/phred/genome/bin/rbs_finder.pl"
        host="{teste}" provider="gt4"
        arguments="/mnt/win/phred/genome/workflow/seqs_fasta.contigs
        /mnt/win/phred/genome/workflow/tmp.coord
        /mnt/win/phred/genome/workflow/rbs.out 50"
        stdout="/mnt/win/phred/genome/workflow/rbs_finder.pl.out"
        stderr="/mnt/win/phred/genome/workflow/rbs_finder.pl.err"/>

        <execute executable="/mnt/win/phred/genome/bin/hmmpfam"
        host="{teste}" provider="gt4"
        arguments="/mnt/win/phred/genome/databases/Pfam_fs
        /mnt/win/phred/genome/workflow/tmp.train"
        stdout="/mnt/win/phred/genome/workflow/hmmpfam.out"
        stderr="/mnt/win/phred/genome/workflow/hmmpfam.err"/>

        <execute executable="/mnt/win/blast/bin/blastall" host="{teste}"
        provider="gt4" arguments="-p blastp -i
        /mnt/win/phred/genome/workflow/seqs_fasta.contigs"
        stdout="/mnt/win/phred/genome/workflow/blastp.out"
        stderr="/mnt/win/phred/genome/workflow/blastp.err"/>
      </parallel>
</project>
```

Figura 33. *Script workflow* na linguagem Karajan

Apêndice E – Descrição do *Workflow*

A figura 34 apresenta o arquivo em padrão *xml* de descrição do primeiro exemplo de *workflow* implementado no protótipo. O *xml* mostra a funcionalidade, banco de dados e programas utilizados em cada tarefa.

```
<?xml version="1.0" encoding="UTF-8"?>
<descricao_workflow>
    <funcao_principal >O workflow tem a funcionalidade de pesquisar nucleotideos dentro de
alguns banco de dados de proteinas      em nodos de uma configuração grade. O pacote de
software Blast foi usado para pesquisar as sequências </funcao_principal>
       <tarefa nome="A">
            <funcao >É o começo da execução, responsável por inicializar o módulo Grid
Provider como sendo o Globus Toolkit 4.0.0</ funcao>
            <programa>NENHUM</programa>
            <bancodedados>NENHUM</bancodedados>
    </tarefa>
      <tarefa nome="B">
            <funcao>Deleta todos os arquivos gerados na execução anterior</ funcao>
            <programa>clean</programa>
            <bancodedados>NENHUM</bancodedados>
    </tarefa>
    <tarefa nome="C">
            <funcao >Invoca a aplicação Blast para pesquisar por uma específica sequencia
em uma banco de dados de proteinas </ funcao>
            <programa>Blastn</programa>
            <bancodedados>Ecoli</bancodedados>
    </tarefa>
      <tarefa nome="D">
            <funcao >Invoca a aplicação Blast para pesquisar por uma específica sequencia
em uma banco de dados de proteinas</ funcao>
            <programa>Blastn</programa>
            <bancodedados>MITO</bancodedados>
    </tarefa>
    <tarefa nome="E">
            <funcao >Invoca a aplicação Blast para pesquisar por uma específica sequencia
em uma banco de dados de proteinas</ funcao>
            <programa>Blastn</programa>
            <bancodedados>IGSEQPROT</ bancodedados>
    </tarefa>
      <tarefa nome="F">
            <funcao >transfere todos os arquivos (que resultam das atividades C,D e E) para
uma localização onde o usuário móvel pode acessar</ funcao>
            <programa>cp</programa>
            <bancodedados>NENHUM</bancodedados>
    </tarefa>
</descricao_workflow>
```

Figura 34. Descrição do primeiro *workflow*

A figura 35 ilustra o arquivo em padrão *xml* da descrição do segundo exemplo de *workflow* implementado no protótipo de avaliação do trabalho de dissertação.

```
<?xml version="1.0" encoding="UTF-8"?>
<descricao_workflow>
    < funcao_principal   >Workflow para analisar e anotar a bactéria Gluconacetobacter
diazotrophicus.</funcao_principal>
      <tarefa nome="A1">
            <funcao >O primeiro passo é gerar os reads e contigs dos cromatogramas da
bactéria, obtidos nos laboratórios.</funcao>
            <programa>Phrep, Phd2fasta, Phrap</programa>
            <bancodedados>NENHUM</bancodedados>
    </tarefa>
      <tarefa nome="A2">
            <funcao>Ele trata da predição de genes. Como Glimmer pode gerar falso positivos
e falso negativos, pesquisadores analisam outras informações para identificar um gene, como
por exemplo a presência de tRNAs (um outro tipo gene não detectado pelo Glimmer),
terminador (tarefa A4) e sites de binding ribossome (tarefa A5). Na presente tarefa ocorre a
busca por tRNAs.</funcao>
            <programa>tRNAscan-SE</programa>
            <bancodedados>NENHUM</bancodedados>
    </tarefa>
    <tarefa nome="A3">
            <funcao >Ele obtém ORFs (open reading frames), que são possíveis genes</
funcao>
            <programa>Glimmer</programa>
            <bancodedados>NENHUM</bancodedados>
    </tarefa>
      <tarefa nome="A4">
            <funcao>Procura terminador</funcao>
            <programa>Transterm</programa>
            <bancodedados>NENHUM</bancodedados>
    </tarefa>
    <tarefa nome="A5">
            <funcao>Analisa sites de binding ribosome</funcao>
            <programa>rbs_finder.pl</programa>
            <bancodedados>NENHUM</bancodedados>
    </tarefa>
      <tarefa nome="A6">
            <funcao >Trata do descobrimento da função do gene, que é feita através da
comparação da sequência de ORF com outras sequências já existentes em dados públicos.
Usando um programa para coleção de família de proteínas e domínios armazenados em um
banco de dados</funcao>
            <programa>HMMPFam</programa>
            <bancodedados>PFam</bancodedados>
    </tarefa>
    <tarefa nome="A7">
            <funcao >Trata do descobrimento da função do gene, que é feita através da
comparação da sequência de ORF com outras sequências já existentes em dados públicos.
Usando um programa para coleção de família de proteínas e domínios armazenados em um
banco de dados</funcao>
            <program>Blastp</program>
            <bancodedados>NR</bancodedados>
    </tarefa>
</descricao_workflow>
```

Figura 35. Descrição do segundo *workflow*

Apêndice F – Configuração do Ambiente de Desenvolvimento

Nesta seção são apresentados, de maneira mais detalhada, alguns dos programas utilizados para configuração do ambiente de desenvolvimento.

D.1 Ambiente de Software

D.1.1 *Java 2 Micro Edition* (J2ME)

A plataforma *Java Plataform, Micro Edition* (J2ME) é destinada para o desenvolvimento de aplicativos para dispositivos com recursos restritos, tais como, memória, tela e poder de processamento limitados. Tal plataforma inclui a máquina virtual Java e um conjunto de APIs padrão do Java, definidos através da *Java Community*.

A arquitetura J2ME define configurações, perfis e APIs opcionais, como ilustrado na figura 36. Esta classificação indica as informações específicas sobre diferentes dispositivos. Uma configuração J2ME é composta de uma máquina virtual e um conjunto reduzido de bibliotecas. As bibliotecas do J2ME fornecem funcionalidades básicas para diversos modelos de dispositivos que compartilham características similares.

O J2ME prevê duas configurações, uma para dispositivos com maior capacidade computacional, denominado *Connected Device Configuration* (CDC), e a outra com menor capacidade computacional, denominado *Connected Limited Device Configuration* (CLDC).

- ***CLDC**:* consiste de uma maquina virtual reduzida (KVM) e um conjunto de classes mais apropriadas para dispositivos de conexões de rede intermitentes, processadores lentos e memória limitada. Incorpora também um *garbage collector*, que é otimizado para um ambiente de memória reduzida. A KVM é geralmente usada em

diversas implementações, mas existem outras máquinas virtuais que poderiam ser utilizadas, tais como J9 VM da IBM. Neste trabalho foi adotada a configuração CLDC 1.1, em virtude da sua configuração com maior capacidade de adaptação as características de hardware dos dispositivos móveis usados;

- ***CDC:*** são direcionados para a necessidade de dispositivos que se posicionam entre aqueles discutidos pelo CLDC e os completos sistemas *desktops* executando J2SE. Estes dispositivos tem mais memória e processadores com mais capacidade, o que permite suportar um ambiente mais completo do Java. O CDC é encontrado em PDAs de alta capacidade, *smartphones*, telefones *web*, *gateways* residenciais e *set-top boxes*.

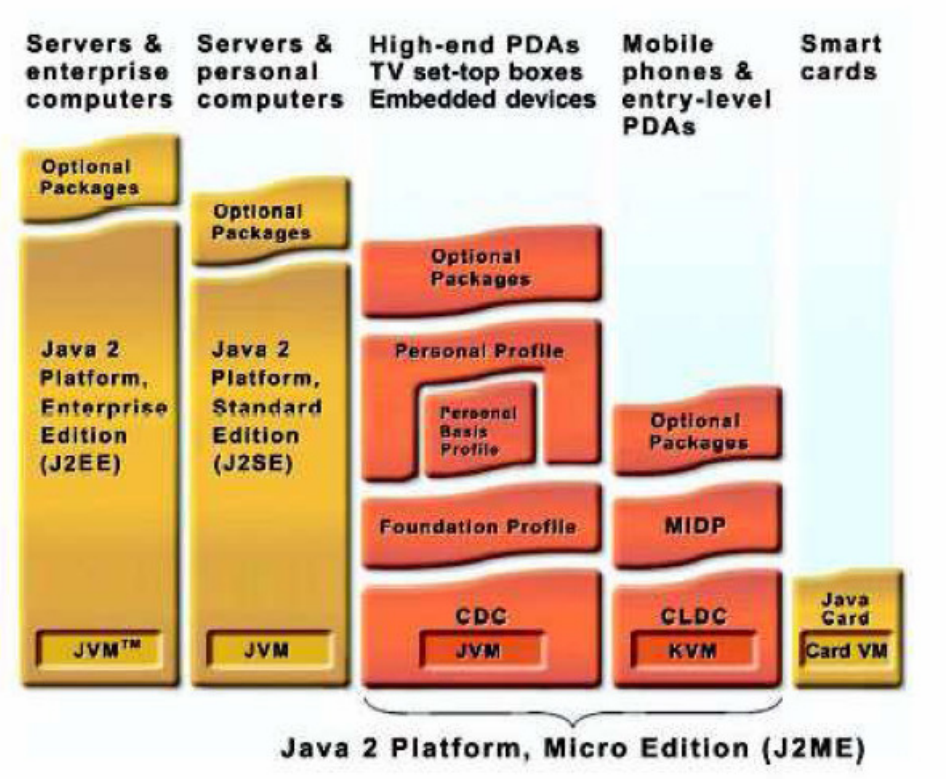

Figura 36. Arquitetura do J2ME

O perfil é composto por um conjunto de classes que permite os desenvolvedores implementarem as aplicações segundo as convenções das aplicações dos pequenos dispositivos. Os perfis existentes são:

- *Foundation Profile:* é a base para dispositivos em rede sem interface gráfica,

usualmente usada com o CDC;

- *Personal Basis* e *Personal Profile:* é a base para dispositivos com suporte gráfico e alta capacidade, usualmente usado em conjunto com o CDC;

- *Mobile Information Device Profile* (MIDP): provê as funcionalidades requeridas pelas aplicações móveis, tais como interface com o usuário, acesso a rede, armazenamento local e segurança. Juntamente com CLDC, o MIDP fornece um ambiente de execução java e eleva as capacidades dos dispositivos móveis, além de minimizar o consumo de memória e energia;

A plataforma J2ME permite a combinação de vários pacotes com o CLDC, CDC e seus correspondentes perfis. As APIs opcionais permitem a utilização de tecnologias como, *bluetooth, web services, wireless messaging*, multimídia, e conexão com banco de dados. Para o desenvolvimento da aplicação no cliente móvel deste trabalho, optou-se pela utilização do MIDP 2.0. Tal escolha é decorrente do fato desta versão fornecer suporte a uma maior variedade de protocolos de comunicação.

D.1.2 Protocolos de Comunicação

A versão 1.0 do MIDP não suporta rede de baixo nível para *sockets* TCP/IP. Esta deficiência foi suprida na especificação MIDP 2.0, como resposta às necessidades das redes de computadores.

Este suporte está baseado no *Generic Connection Framework* (GCF) do CLDC. O GCF provê uma interface genérica para diferentes formas de comunicação (por exemplo, *socket*, *datagrama* e *http*) e URL padronizadas para indicar o tipo de conexão criada. Uma outra forma de comunicação utilizada na plataforma Java é a RMI (*Remote Method Invocation*). No entanto, esta não é suportada pela J2ME/CLDC, apenas na J2ME/CDC.

D.1.3 Máquina Virtual do Dispositivo Móvel

A máquina virtual instalada no dispositivo móvel é a disponibilizada pela

WebSphere Everyplace Micro Environment (IBM Software, 2006), que é uma ambiente de execução que fornece fundamentos para o desenvolvimento de aplicações em pequenos dispositivos móveis. Este ambiente contém em seu núcleo a máquina virtual J9 da IBM que suporta aplicações J2ME em vários dispositivos (por exemplo, PDA, telefones celulares e outros dispositivos embarcados), diferentes sistemas operacionais (por exemplo, PalmOS, PocketPC, WinCE, QNX, OSE, Linux e outros) e com diferentes perfis e configurações (por exemplo, MIDP, CLDC e CDC). É uma máquina virtual menor e mais rápida que a KVM da *Sun*. Suporta adição de cores, além de possuir um *linker* que remove classes, métodos e campos desnecessários da aplicação nas bibliotecas de classes. Suporta depuração remota (executa e depura a aplicação de um PDA em um PC). Assim, este ambiente propicia a portabilidade necessária para as aplicações móveis.

D.1.4 Java 2 Standard Edition *(J2SE)*

A implementação do Gerenciador *Workflow* e do Módulo TF foi realizada usando a ferramenta J2SE. O J2SE possui dois produtos principais: *Java 2 Runtime Environment Standard Edition* (JRE) e *Java 2 Software Development Kit Standard Edition* (SDK). O JRE fornece as APIs, a máquina virtual, e outros componentes necessários na execução de aplicações escritas na linguagem de programação java. O Java 2 SDK contém ferramentas como compiladores e *debuggers* necessários para o desenvolvimento de aplicações. A versão utilizada na implementação foi a 1.5.0_04-b05 do J2SE.