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

Alex Pinho Magalhães

BUSCA DE INFORMAÇÃO DISTRIBUÍDA USANDO HEURÍSTICAS ADAPTATIVAS PARA AGENTES MÓVEIS EM TEMPO REAL

Florianópolis

2010

Alex Pinho Magalhães

BUSCA DE INFORMAÇÃO DISTRIBUÍDA USANDO HEURÍSTICAS ADAPTATIVAS PARA AGENTES MÓVEIS EM TEMPO REAL

Dissertação submetida ao Programa de Pós-Graduação da Universidade Federal de Santa Catarina para a obtenção do Grau de Mestre em Ciência da Computação.

Orientador: Prof. Dr. Lau Cheuk Lung
Co-orientadora: Profª. Dra. Luciana de Oliveira Rech

Florianópolis

2010

Catalogação na fonte elaborada pela biblioteca da
Universidade Federal de Santa Catarina

M188b Magalhães, Alex Pinho
Busca de informação distribuída usando heurísticas adaptativas para agentes móveis em tempo real [dissertação] / Alex Pinho Magalhães ; orientador, Lau Cheuk Lung. – Florianópolis, SC : 2010.
112 p.: il., grafs.

Dissertação (mestrado) - Universidade Federal de Santa Catarina, Centro Tecnológico, Programa de Pós-Graduação em Ciência da Computação.

Inclui referências

1. Ciência da computação. 2. Agentes móveis (software). 3. Arquitetura de computador. 4. Sistemas distribuídos. 5. Inteligência artificial. 6. Recuperação da informação. I. Lung, Lau Cheuk. II. Universidade Federal de Santa Catarina. Programa de Pós-Graduação em Engenharia de Automação e Sistemas. III. Título.

CDU 681

Alex Pinho Magalhães

BUSCA DE INFORMAÇÃO DISTRIBUÍDA USANDO HEURÍSTICAS ADAPTATIVAS PARA AGENTES MÓVEIS EM TEMPO REAL

Esta Dissertação foi julgada adequada para obtenção do Título de Mestre em Ciência da Computação, área de concentração Sistemas Distribuídos e aprovada em sua forma final pelo Programa de Pós-Graduação de Ciência da Computação da UFSC.

Florianópolis, 09 de Dezembro de 2010.

_____
Prof. Mário Antônio Ribeiro Dantas, Dr.
Coordenador do Curso

Banca Examinadora:

_____
Prof. Lau Cheuk Lung, Dr.
Orientador, Universidade Federal de Santa Catarina

_____
Profª. Luciana Rech, Dra.
Co-Orientadora, Universidade Federal de Santa Catarina

_____
Prof. Luiz Eduardo S. Oliveira, Ph.D.
Universidade Federal do Paraná

_____
Prof. Fábio Favarim, Dr.
Universidade Tecnológica Federal do Paraná

_____
Profª. Patrícia Plentz, Dra.
Universidade Federal de Santa Catarina

Dedico este trabalho à minha família que sempre apoiou meus estudos, em especial à minha esposa Adriana, que foi obrigada a me aturar em casa durante os diversos fins de semana de sol que eu passei estudando. Dedico também aos meus amigos cariocas e aos novos amigos que encontrei em Florianópolis, os quais permitiram que mesmo em uma nova cidade eu nunca perdesse a sensação do aconchego de um lar. Em especial dedico este trabalho aos meus parentes e amigos que partiram desta vida durante esses meus tantos anos de estudo. Que a redução de nossos contatos em vida possa ser recompensada com um futuro melhor para todos que habitam o nosso planeta.

AGRADECIMENTOS

Agradeço a todos os meus amigos do Laboratório de Pesquisa em Sistemas Distribuídos (LaPeSD/UFSC), em especial ao meu orientador Prof. Lau Lung e à minha co-orientadora Prof. Luciana Rech, que sempre viram com bons olhos as mudanças sugeridas ao meu tema de trabalho e, principalmente, aceitaram minha condição de tempo parcial, necessária para que eu garantisse meu sustento. Agradeço ainda à toda equipe do INE/UFSC, que sempre esteve pronta para me auxiliar em todos os aspectos.

“A melhor maneira de prever o futuro é inventá-lo.”

(Alan Kay, 1971)

RESUMO

Com a contínua expansão da Internet, a busca de informação relevante deve atender requisitos de tempo e qualidade. A área da inteligência artificial possui soluções de busca informada sofisticadas que tratam essas variáveis: tempo e qualidade. Esses sistemas de busca de informação são sistemas distribuídos de tempo real, uma área emergente de pesquisa e que tem nos apresentado soluções de arquitetura computacional com alta escalabilidade. Uma alternativa para a busca de informação em tempo real é a tecnologia de agentes móveis, a qual vem sendo objeto de pesquisa desde antes da popularização da Internet, mas vem ganhando força com a padronização desta tecnologia através da FIPA, uma fundação filiada ao IEEE. A introdução de agentes móveis no cenário de sistemas distribuídos traz diversas vantagens e possuem características que são altamente desejáveis para sistemas distribuídos de tempo real. Os agentes móveis também utilizam técnicas originadas na inteligência artificial para melhorar seu desempenho em cenários com restrição temporal. Uma dessas técnicas é a de *anytime algorithms*, que permite ao algoritmo controlar a qualidade da resposta em função do tempo de execução.

Neste trabalho são apresentados cenários envolvendo busca de informação em servidores distribuídos. Para realizar experimentos nestes cenários, são apresentadas novas heurísticas para determinação de itinerário de agentes móveis na busca de informação em ambientes distribuídos com características de tempo real. São utilizadas plataformas de agentes móveis existentes no mercado, padrões de tecnologia de agentes e também técnicas de controle de execução de algoritmos em ambientes de tempo real.

ABSTRACT

As the Internet continuously grows, the search for relevant information becomes a matter of great importance. Artificial intelligence provides elegant solutions for informed search problems that optimizes both time and quality variables. Therefore, informed search systems are also real time distributed systems, an emerging research field that has presented highly scalable computer architecture solutions. An alternative for the real time information search current solutions is the use of mobile agents technology, which is subject of research since the beginning of the Internet, but has grown in interest after the standardization of this technology through the FIPA, an IEEE standards committee. The use of mobile agents to solve distributed systems problems brings up many advantages and it has characteristics that are highly desirable for real time distributed systems. Mobile agents can also use artificial intelligence techniques to improve its performance in deadline related problems. One of these techniques is the anytime algorithms, which allows the algorithm to control the quality of the response in trade of a faster execution time.

In this work, it is presented information search problems in distributed servers. To carry through the experiments, it will be presented new heuristics to determine the itinerary of mobile agents in search of information in distributed environments within real time problems. Existing mobile agents platforms are used, as well as mobile agents technology standards and algorithms execution control techniques for real time environments.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

ACL – Agents Communication Language
AM – Agentes Móveis
BPEL – Business Process Execution Language
CA – Collector Agent
CALTECH – California Institute of Technology
FA – Fusion Agent
FIPA – Foundation for Intelligent Physical Agents
IA – Inteligência Artificial
IdA – Identifier Agent
GA – Genetic Algorithm
GFU – Gulosa com Função de Utilidade
GLVP – Gulosa com Limite Variável na Partida
IEEE – Institute of Electrical and Electronic Engineers
JADE – Java Agent DEvelopment Framework
MASIF – Mobile Agent System Interoperability Facilities
MIS – Multi-modal Information System
MOP – Multi Objective Problem
NASA – National Aeronautics and Space Administration
OMG – Object Management Group
PCVP – Preguiçosa com Crescimento Variável na Partida
RM – Rede Multimodal
SA – Scheduler Agent
SOA – Service-Oriented Architecture
TR – Tempo Real
TSP – Travelling Salesman Problem
UDDI – Universal Description, Discovery and Integration

SUMÁRIO

1 INTRODUÇÃO

1.1 CONTEXTUALIZAÇÃO

Com o contínuo crescimento da Internet e a disseminação de informação e conteúdo em todo o planeta, surge a necessidade em se coletar informação relevante e dentro de um espaço de tempo em que a informação seja útil para quem a solicitou. Soluções para busca de informação estão disseminadas nos mais diferentes tipos de problemas. O campo da inteligência artificial possui soluções para busca informada desde muito antes da criação da Internet, mas com o crescimento da informação disponível e a complexidade de distribuição da mesma, essas soluções de busca informada ficaram mais sofisticadas para incluírem a variável tempo em seus algoritmos [Koenig 2006]. Equipamentos para auxílio de navegação também apresentam soluções de busca de informação onde a variável tempo é muito importante. Durante uma viagem de um veículo até seu destino, esses equipamentos precisam monitorar a posição do veículo com o auxílio de um GPS (*Global Positioning System*), tratar outras informações auxiliares como rotas disponíveis e tráfego, para conseguirem conduzir uma viagem segura e dentro do prazo solicitado [Kanoh 2008].

Esses sistemas de busca de informação têm em comum o fato de serem sistemas distribuídos de tempo real, uma área emergente de pesquisa e que tem nos apresentado soluções de arquitetura computacional bastante escaláveis, capazes de atender desde um ambiente confinado de intranet de uma empresa, até ambientes altamente heterogêneos de redes *peer-to-peer* pela Internet [Barker 2008].

Uma alternativa para a busca de informação em tempo real é a tecnologia de agentes móveis. Agente móvel é um *software* auto contido capaz de perceber o ambiente ao seu redor e agir ou não, de acordo com seus próprios objetivos. A tecnologia de agentes móveis vem sendo objeto de pesquisa desde antes da popularização da Internet, mas vem ganhando força com a padronização desta tecnologia [FIPA 2002] e também com o reconhecimento de importantes sociedades acadêmicas e institutos, como o IEEE [FIPA 2010].

A introdução de agentes móveis no cenário de sistemas distribuídos traz diversas vantagens, como a possibilidade de execução assíncrona e autônoma, a diminuição de tráfego e latência na rede, a economia de

carga em dispositivos móveis, a manipulação de grandes volumes de dados no local onde eles residem e a facilidade de adaptação a ambientes heterogêneos como a Internet [Martins 2002].

Apesar da diminuição da latência na rede e a natureza de execução *in loco* serem características altamente desejáveis para sistemas distribuídos de tempo real, os agentes móveis ainda utilizam técnicas originadas na inteligência artificial para melhorar seus desempenhos em cenários de tempo real. Uma dessas técnicas é a de *anytime algorithms*. Esta técnica consiste em permitir que um algoritmo seja interrompido em qualquer etapa de sua execução e o resultado parcial obtido até o momento é considerado como o resultado de retorno, sem erro para o solicitante [Garvey 1994]. Esta técnica permite que o agente possa equilibrar as diversas variáveis do sistema, como reduzir a qualidade de um resultado, em troca de tempo para executar as demais tarefas de sua missão.

Neste trabalho será apresentado um cenário de busca de informação em servidores distribuídos, como a busca de preços em sites de comércio eletrônico. Dispositivos móveis precisam economizar tanto em energia, como no acesso à Internet e, portanto são fortes candidatos à abordagem de agentes móveis. Neste cenário, a missão do agente é buscar o melhor preço no maior número de sites possível.

Também será apresentada uma variação desse cenário que necessita da técnica de *anytime algorithms*. No novo cenário os servidores admitem que agentes inteligentes negociem o preço com eles, usando como base as ofertas encontradas em outros servidores de comércio eletrônico. O preço passa a ser um recurso variável, pois um agente pode gastar mais tempo em um servidor e assim melhorar a qualidade de sua resposta.

Estes cenários utilizarão uma arquitetura de sistemas conhecida como orquestração de serviços, o tipo de arquitetura predominante em sistemas dedicados a serviços. Porém, para lidar com os requisitos de desempenho que a orquestração tradicional já começa a não ser capaz de atender, será utilizada a orquestração descentralizada, uma abordagem que tem sido considerada na literatura como bastante promissora [Binder 2006] e [Barker 2008,2009]. A principal diferença nesta abordagem é que cada serviço que compõe o serviço composto possui um pedaço da lógica do processo e é capaz de invocar o próximo serviço do fluxo, sem ter que retornar para o orquestrador até que o fluxo esteja concluído.

Para realizar experimentos nestes cenários, são apresentadas novas heurísticas para agentes móveis na busca de informação em ambientes distribuídos com características de tempo real.

Apesar de a literatura tratar de temas similares usando agentes móveis, o uso de técnicas como *anytime algorithms* em ambientes com recursos variáveis não foram identificados na literatura e são extremamente relevantes para sistemas com restrição temporal.

1.2 OBJETIVOS

O objetivo principal deste trabalho é desenvolver novas heurísticas para determinação de itinerário de agentes móveis com restrição temporal e analisá-las em diferentes cenários reais, utilizando plataformas de agentes móveis existentes no mercado, padrões de tecnologia de agentes e também técnicas de controle de execução de algoritmos em ambientes de tempo real.

Para atender ao objetivo geral deste trabalho, os seguintes objetivos específicos foram definidos:

- Fazer um estudo sobre o estado da arte: agentes móveis, sistemas de tempo real e orquestração de serviços;
- Desenvolver heurísticas adaptativas capazes de guiar o agente móvel na escolha de seu itinerário, equilibrando as variáveis tempo de execução e qualidade do serviço;
- Realizar experimentos combinando a arquitetura e as heurísticas e, por fim, avaliar os resultados obtidos.

1.3 METODOLOGIA

De acordo com a classificação proposta por [Silva 2001], este trabalho possui as seguintes características:

- Pesquisa aplicada: os resultados aqui apresentados têm por finalidade a aplicação prática e a solução de problemas relacionados à busca de informação;
- Pesquisa quantitativa: os resultados deste trabalho são apresentados em números, a fim de que possam ser comparados e analisados;

• Pesquisa Exploratória: este trabalho visa proporcionar maior familiaridade com a classe de problema apresentada, a fim de permitir a construção de hipóteses e realização de experimentos;

• E, por último, do ponto de vista dos procedimentos técnicos, este trabalho é uma Pesquisa Bibliográfica e Experimental: é uma pesquisa elaborada a partir de material já publicado, mas também possui um objeto de estudo claro e definido, sobre o qual foram definidos e especificados experimentos para observação de seus efeitos.

A partir desta classificação apresentada, este trabalho foi realizado seguindo o seguinte roteiro:

• Revisão bibliográfica de literatura relevante ao trabalho. O resultado desta revisão se encontra concentrado nos Capítulos 2, 3 e 4;

• Análise de requisitos para desenvolvimento de novas heurísticas para determinação do itinerário de agentes móveis com restrição temporal;

• Desenvolvimento de uma arquitetura para sistemas de agentes móveis baseada em padrões acadêmicos e de mercado;

• Implementação e avaliação das heurísticas, aplicadas aos modelos computacionais desenvolvidos e obedecendo aos requisitos e padrões estudados;

• Escrita da dissertação, com base nas informações obtidas nos itens anteriores.

1.4 ORGANIZAÇÃO DO TEXTO

O texto desta dissertação está dividido e organizado em sete capítulos.

Este primeiro capítulo descreveu o contexto geral desta dissertação, bem como a metodologia adotada.

No Capítulo 2 são apresentados os conceitos sobre agentes móveis necessários para a compreensão desta dissertação. São introduzidos os padrões da indústria para agentes móveis, bem como uma arquitetura baseada na linguagem Java e trabalhos relacionados ao tema.

No Capítulo 3 são apresentados os conceitos de sistemas de tempo real, técnicas de computação imprecisa associadas ao tema de restrição temporal, em especial a técnica de *anytime algorithms*, e trabalhos relacionados ao tema.

No Capítulo 4 é apresentada a orquestração de serviços e suas variações, em especial a composição de serviços com orquestração descentralizada. Também é apresentado o tema da qualidade de serviços (QoS) e o escopo ao qual este tema será referido neste trabalho. Novos trabalhos relacionados ao tema são apresentados aqui.

O Capítulo 5 descreve os recursos que foram usados na formulação do problema e apresenta a primeira proposta de nova heurística, baseada na classe de heurísticas adaptativas. A heurística proposta aqui recebeu o nome de Lazy-Adaptativo, é levemente baseada na heurística Preguiçosa, porém possui características de adaptação dinâmica. Este capítulo ainda contém a modelagem do cenário e formulação do problema que servirão para realização dos experimentos. Também é apresentada uma análise comparativa entre a heurística Lazy-Adaptativo e outras heurísticas existentes na literatura.

O Capítulo 6 apresenta três novas heurísticas, pertencentes à classe de heurísticas com adaptação na partida. As três heurísticas são baseadas em heurísticas simples. A primeira é baseada na Heurística Gulosa com capacidade de adaptação a recursos variáveis, a segunda é baseada na Heurística Preguiçosa, também com capacidade de adaptação a recursos variáveis, e a terceira também é baseada na Heurística Gulosa, com a diferença de que sua capacidade de adaptação está associada a uma função de utilidade baseada no algoritmo de bipartição. É descrito um experimento comparativo entre as três novas heurísticas propostas, utilizando o conceito de recursos variáveis.

Finalizando, o Capítulo 7 apresenta as conclusões e os resultados obtidos.

1.5 CONTRIBUIÇÕES

Dentre as principais contribuições desta dissertação estão uma nova heurística adaptativa (Lazy-Adaptativo) e também três novas heurísticas capazes de se adaptarem em cenários com deadline fixos.

Ao longo do desenvolvimento deste trabalho, foram publicados os seguintes artigos:

MAGALHÃES, Alex; Rech, Luciana; Lung, Lau; Oliveira, Rômulo (2009). **A Heurística Lazy Adaptativa para Determinar Itinerários de Agentes Móveis Imprecisos**, SBRC 2009, XXVII Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos. UFPE, Recife, Brasil.

MAGALHÃES, Alex; Rech, Luciana; Lung, Lau; Oliveira, Rômulo (2010). **Using Intelligent Mobile Agents to Dynamically Determine Itineraries with Time Constraints**, ETFA 2010, 15th IEEE International Conference on Emerging Technologies and Factory Automation. Bilbao, Espanha.

MAGALHÃES, Alex; Rech, Luciana; Lung, Lau (2010). **Decentralized Services Orchestration Using Intelligent Mobile Agents with Deadline Restrictions**, AIAI 2010, 6th IFIP Conference on Artificial Intelligence Applications & Inovations. Larnaca, Chipre.

2 AGENTES MÓVEIS

Neste capítulo é apresentado o conceito de agentes móveis com suas características, especificações e padronizações, além de referências de implementação e *frameworks* populares no meio acadêmico e na indústria.

2.1 AGENTES

Segundo a visão da inteligência artificial, um agente é tudo que for capaz de perceber o ambiente ao seu redor e agir sobre este ambiente utilizando suas próprias forças e segundo suas próprias regras [Russel 1995].

Corroborando esta definição, [Martins 2002] diz que um agente de *software* é um processo em execução capaz de decidir de forma autônoma que ações tomar à luz de um evento externo, sem a necessidade de um operador.

2.1.1 Tipos de Agentes

Para melhor diferenciar os tipos de agentes existentes, [Russel 1995] propôs 4 tipos principais de agentes:

• agentes de reflexo simples: a ação dos agentes é iniciada por um estímulo externo, ao qual ele reage sempre da mesma forma;

• agentes de reflexo simples com estado interno: a ação dos agentes também é iniciada por um estímulo externo, mas esta pode variar com o tempo, de acordo com seu estado interno;

• agentes orientados a objetivo: a ação do agente não depende apenas do estímulo externo. Aqui o agente toma decisões baseadas em seu objetivo;

• agentes baseados em função de utilidade: o objetivo *per si* não é suficiente para gerar um comportamento de qualidade. Ainda estimulado pelo ambiente exterior e orientado por um objetivo, o comportamento do agente aqui é definido por uma função de utilidade, que é uma função capaz de mapear uma sequência de estados para números reais.

Essa divisão é bastante fundamentada na robótica clássica e na inteligência artificial, mas no contexto de agentes de *software* em

sistemas distribuídos, ela deixa de contemplar um aspecto dos agentes: a mobilidade de código (diferente da mobilidade robótica, a mobilidade de código migra o código em execução de hospedeiro em hospedeiro, sem precisar estar confinado a um único hóspede ou robô).

2.1.2 Mobilidade de Código

A mobilidade de código serve como uma alternativa às abordagens tradicionais (leia-se modelo cliente-servidor, computação concorrente e outros), focadas no modelo orientado à troca de mensagens ou compartilhamento de memória, e traz benefícios no contexto de vários domínios de aplicação.

Entre os benefícios esperados da mobilidade de código pode-se citar [Baek 2001]: maior grau de flexibilidade, escalabilidade e personalização; melhor utilização da infraestrutura de comunicação e a provisão de serviços de maneira autônoma sem a necessidade de conexão permanente. Tais benefícios justificam o contínuo interesse nesta área de pesquisa. Na Seção 2.5 são apresentados trabalhos recentes que mostram aplicações da mobilidade e a obtenção desses benefícios na prática.

Alguns domínios de aplicação para os quais a mobilidade de código tem sido aplicada na literatura são: recuperação de informação, serviços avançados de telecomunicações, controle de dispositivos remotos, gerência de fluxos de trabalho, comércio eletrônico, entre outros.

2.2 AGENTES MÓVEIS

Um agente móvel é um elemento de software auto contido, responsável pela execução de uma tarefa, e que não está limitado ao sistema onde começa a sua execução, sendo capaz de migrar autonomamente através de uma rede.

Agentes móveis reduzem a carga na rede, executam assincronamente e autonomamente, e adaptam-se dinamicamente, estabelecendo assim um novo paradigma para a programação em ambientes distribuídos.

2.2.1 FIPA/IEEE

FIPA (*Foundation for Intelligent Physical Agents*) é uma organização para padronização e promoção das tecnologias de agentes criada na Suíça em 1996. Em 2005, a organização foi aceita oficialmente na família IEEE como seu décimo primeiro comitê de padronização [FIPA 2010].

Além de promover a padronização das tecnologias de agentes, esta organização também promove a interoperabilidade dos agentes com outras tecnologias que não usam agentes, dentre as quais merece destaque a tecnologia de Web Services [FIPA 2010].

2.2.2 Interoperabilidade

Há pouco mais de uma década a tecnologia de agentes móveis podia ser considerada nova. Muitos experimentos com agentes vinham sendo desenvolvidos e cada qual com sua própria arquitetura. Neste cenário de divergências, o Instituto Fokus e a IBM [Fokus & IBM, 1997] se uniram para gerar uma especificação que serviria para dar o primeiro passo rumo a centralização dos trabalhos de padronização das tecnologias de agentes.

Uma preocupação central era a necessidade de interoperabilidade dos agentes, independente de sua plataforma de origem. Com isso, o grupo se concentrou em buscar a padronização dos agentes através das seguintes características: gerenciamento, mobilidade, estrutura do agente, nomenclatura dos agentes, tipos de agentes e sintaxe de localização.

Reconhecendo o resultado dessa especificação conjunta, a OMG (*Object Management Group*) definiu um padrão em 1997, denominado de MASIF (*Mobile Agent System Interoperability Facilities*), um estudo para a interoperabilidade de agentes móveis.

2.2.3 Arquitetura

Aproximadamente na mesma época em que foi publicado o padrão MASIF/OMG, a recém-criada FIPA, ainda sem a chancela da IEEE designava um grupo para a padronização de agentes móveis, com a tarefa de criar uma arquitetura que permitisse uma troca de mensagens entre agentes com relevância semântica, mesmo que entre canais de transporte distintos ou linguagens distintas [FIPA 2002].

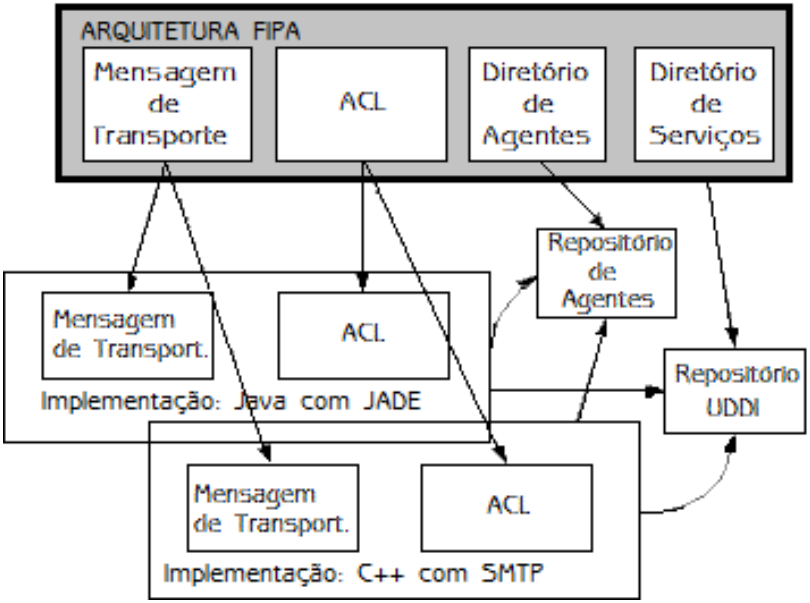

Figura 2.1. Abstração de Arquitetura FIPA [FIPA 2002]

Tendo como ponto de partida o padrão MASIF/OMG, os pontos de interoperabilidade foram expandidos para:

• Um diretório de serviços para agentes, similar ao UDDI (*Universal Description, Discovery and Integration*) existente nos Web Services [OASIS 2002];

• Interoperabilidade do Transporte de Mensagens;

• Suporte a várias representações de Linguagens de Comunicação de Agentes (ACL);

• Suporte a várias formas de representação de conteúdo;

• Suporte a várias formas de representação de diretórios de serviços para agentes.

A Figura 2.1 representa a abstração da primeira arquitetura criada como proposta de modelo para a interoperabilidade de agentes móveis.

2.2.4 Linguagem de Comunicação

Possuindo uma arquitetura focada na troca de mensagens e interoperabilidade, o padrão FIPA necessitava de uma linguagem comum a todos os agentes. Assim surgiu a FIPA-ACL (*Agent Communication Language*), o padrão FIPA para troca de mensagens entre agentes [FIPA 2002b].

Uma mensagem FIPA-ACL é composta de diversos parâmetros: ação (*performative*), remetente (*sender*), destinatário (*receiver*), conteúdo (*content*), linguagem, ontologia, identificador de conversação, codificação (*encoding*), protocolo, *reply-to*, *reply-with*, *in-reply-to* e *reply-by*.

Mas por simplicidade do padrão, apenas um parâmetro é obrigatório: a ação, muito embora a maioria das mensagens, na prática, também utilize os parâmetros: remetente, destinatário e conteúdo.

2.2.5 Complementando os Padrões

Apesar do fortalecimento e padronização da tecnologia de agentes na última década, em especial os padrões MASIF e FIPA, apenas em 2005 a organização FIPA destacou um subgrupo especificamente para continuar os trabalhos com agentes móveis. Anteriormente, o trabalho com agentes móveis fora realizado por grupos de escopo mais amplo, responsáveis também por assuntos como semântica, ontologia, comunicação e segurança [FIPA 2010].

Vários aspectos chave precisavam evoluir, pois as especificações existentes tratavam apenas de parte dos seguintes aspectos: mobilidade de código, linguagens de comunicação, infraestrutura, interoperabilidade e segurança.

Para continuar o trabalho iniciado pelos padrões MASIF e FIPA, o novo grupo trabalha na ampliação das especificações existentes e desenvolve implementações de referência para serem usadas pela comunidade. Muitos grupos ativos da comunidade têm auxiliado nesse trabalho em especial o grupo responsável pelo *framework* JADE [Bellifemine 2001].

2.3 JADE

JADE (*Java Agent DEvelopment Framework*) é uma plataforma que provê diversas funcionalidades para sistemas distribuídos baseados em agentes na forma de uma camada intermediária e que é independente dos sistemas que a utilizam [Bellifemine 2001].

A abstração de agente utilizada em JADE possui algumas características que difere os sistemas baseados em agentes dos demais sistemas sem agentes: um agente é autônomo e proativo, ele pode dizer não para uma requisição e são pares de um sistema *peer-to-peer*.

É importante ressaltar que JADE é compatível com o padrão FIPA e utiliza a linguagem FIPA-ACL. O fato de ser uma plataforma baseada em Java traz benefícios na flexibilidade de implementação, uma vez que a própria linguagem Java já possui características multissistemas. Mesmo assim, ao utilizar a padronização FIPA, ela permite que seus agentes possam ser interoperáveis com outros sistemas que não sejam o JADE e mesmo que estejam implementados em outras linguagens.

2.3.2 Arquitetura

A arquitetura de uma aplicação que utilize JADE deve prever uma estrutura organizacional dos nodos de uma rede, a qual pode ser visualizada na Figura 2.2. Essa é uma visão geral da arquitetura JADE, baseada nos padrões FIPA.

Existem alguns elementos na arquitetura JADE que merecem destaque: plataforma, contêiner principal, contêineres secundários e nodos.

A plataforma é o coração de uma arquitetura JADE. Uma aplicação ou aplicações podem conter uma ou mais plataformas. É nesta plataforma que os nodos são registrados e também os agentes. O serviço de descoberta de agentes e de nodos é dependente do conceito de plataforma.

O contêiner principal é aquele onde a plataforma reside fisicamente. Todos os demais contêineres se registram com ele para participarem daquela plataforma e assim também é com os agentes.

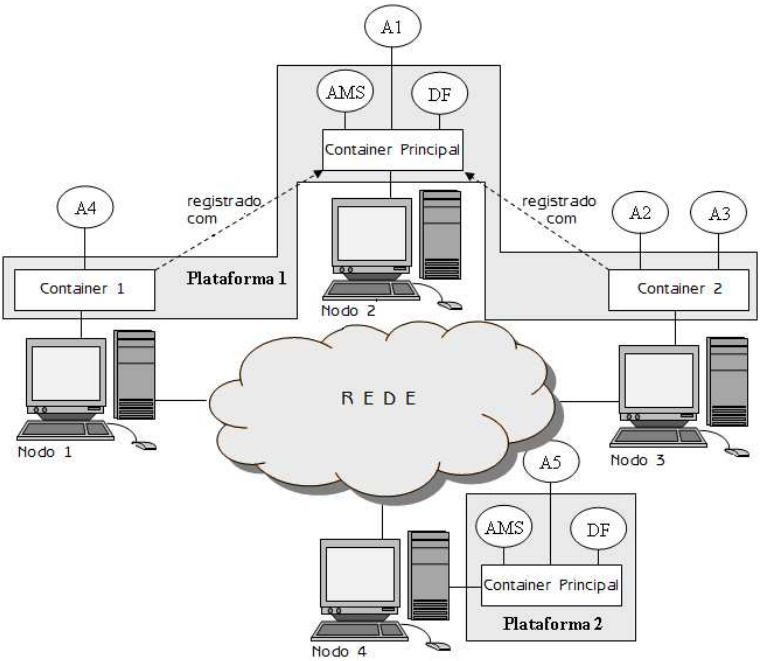

Figura 2.2. A Arquitetura JADE [Bellifemine 2001]

Na Figura 2.2 é possível verificar que o contêiner principal representa um ponto de falha central para a plataforma, mas existem soluções de *hardware* e *software* para minimizar este problema. A descrição dessas soluções está além do escopo dessa breve explanação.

Os demais contêineres representam os nodos onde também se encontram o *middleware* JADE e na prática são pontos de acesso para o agente dentro da plataforma. Os nodos são a parte física da plataforma. Todos os contêineres, principal ou secundário, precisam residir em nodos.

2.4 APLICAÇÕES COM AGENTES MÓVEIS

Nesta seção são apresentadas aplicações gerais com agentes móveis presentes na literatura.

2.4.1 Migração de Agentes em Sistema Multimodal

Sistemas Multimodais de Informação (MIS, Multimodal Information System) auxiliam clientes a tomarem boas decisões enquanto navegam. [Zgaya 2008] apresenta uma estratégia de migração de agentes móveis para satisfazer as requisições de clientes em uma rede para um sistema multimodal de informações.

O objetivo principal é obter a satisfação do usuário em termos de custo e tempo de resposta. É proposta ainda uma otimização em dois níveis. O primeiro é otimizar o número de agentes móveis (AM) necessários para explorar a rede multimodal (RM). É feita uma divisão entre rota inicial e final para deduzir as requisições simultâneas. O segundo é otimizar a seleção dos nodos para os AM e assim gerar as respostas baseado no custo total e tempo de resposta global.

O problema pode ser descrito como a atribuição de um plano de trabalho para cada AM na RM, representado pelos nodos que ele precisa visitar. Depois esses nodos são divididos em subgrupos de tarefas. Uma tarefa corresponde a uma sub-requisição que pertence a um ou mais usuários. Um usuário será satisfeito se for respondido rapidamente e com um custo razoável.

Tabela 2.1. Tabela Nodo x Tarefa

	S_1	S_2	S_3	S_4
T_1	(0, 0, 0)	(2, 5, 3)	(4, 3, 3)	(2, 5, 3)
T_2	(2, 4, 5)	(1, 5, 2)	(4, 5, 1)	(3, 8, 3)
T_3	(1, 7, 3)	(0, 0, 0)	(2, 5, 3)	(4, 2, 2)
T_4	(3, 2, 1)	(0, 0, 0)	(0, 0, 0)	(0, 0, 0)
T_5	(2, 3, 1)	(1, 1, 3)	(4, 5, 2)	(4, 5, 3)

Uma tabela Nodo x Tarefa é pré-definida através da coleta de dados e possui uma relação de tempo de processamento, custo da informação a coletar e tamanho dos dados. Sendo assim, uma mesma tarefa pode ser executada em diferentes nodos, com custo e tempos diferentes. Um exemplo desta relação pode ser visto na Tabela 2.1.

Para resolver o problema é proposta uma coordenação entre 5 tipos de agentes:

• IA – agentes de interface. Interagem com os usuários;

• IdA – agentes identificadores. Responsáveis em quebrar as requisições em sub-requisições;

• SA – agentes agendadores. Como vários nodos podem conter o mesmo serviço, cabe ao SA minimizar o custo e tempo total das missões ao assinalar os nodos;

• CA – agentes coletores. É o AM que efetivamente recuperará a informação distribuída;

• FA – agentes de fusão. Estes agentes organizam os dados coletados para responder às requisições simultâneas.

De posse dos agentes, podemos detalhar a solução de 2 níveis. O primeiro é a obtenção do número mínimo de CA com um plano de trabalho para explorar a RM, de forma a minimizar o tempo total, levando em consideração a latência da rede.

Para se obter um plano de trabalho com custo efetivo, presume-se que existe um módulo de monitoramento da rede informando sobre a latência da rede, tráfego, gargalos, falhas, etc.

Fica claro que o melhor tempo de computação é obtido enviando-se um AM para cada nodo, porém o objetivo é manter o melhor tempo de computação total, particionando os nodos, minimizando assim o número de AM. Em cada partição será aplicado um algoritmo clássico de TSP (Problema do Caixeiro Viajante) para otimizar as rotas.

Apesar da formulação do problema de migração em uma RM, diferentemente de [Baek 2001], esta solução não se preocupa com o custo da informação. Dessa forma a tabela Nodo x Tarefa pode ser reescrita com relações de Tempo de Processamento e Tamanho dos Dados, apenas.

No futuro se espera uma maior cooperação entre os diferentes agentes do sistema, a fim de melhorar o desempenho computando as requisições simultâneas que possuam similaridades.

2.4.2 Agentes Móveis para Monitoramento de Ambientes

Monitoramento de ambientes pode ser uma tarefa desafiadora. Por um lado, a frequência de atualização de dados pode ser muito alta. Por outro, a informação pode não estar disponível do ponto de vista de um computador centralizado.

Uma abordagem de monitoramento usando agentes móveis (AM), deadlines para coordenar os agentes e assim mantendo os dados tão

atualizados quanto possível já foi apresentada anteriormente [Ilarri 2008]. Foram realizados experimentos em ambientes reais e com sucesso.

Foi utilizada uma abordagem do tipo dividir e conquistar. Um agente raiz serve como interface para o usuário. O ambiente é então dividido em níveis (pode haver um único), onde vários agentes auxiliares atuam visitando os nodos, filtrando as informações e minimizando assim a comunicação e o tráfego de rede.

Na Figura 2.3 é apresentada esta arquitetura em níveis (camadas) e exemplificado o sentido de comunicação dos dados.

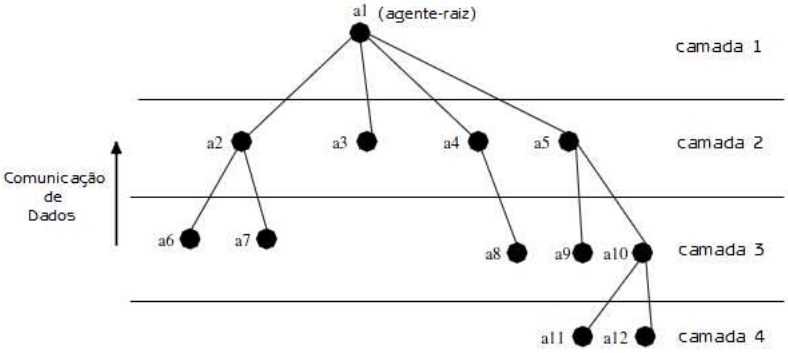

Figura 2.3. Arquitetura em Níveis [Ilarri 2008]

Um exemplo real desse tipo de aplicação é a detecção de aparelhos com Bluetooth em um campus universitário. Como um aparelho desses só pode ser percebido por um computador próximo e habilitado com Bluetooth, uma abordagem centralizada é impossível e uma solução distribuída é exigida.

A proposta de solução em camadas exigiria uma central para os usuários, depois uma segunda camada em cada prédio do campus e novas camadas para cada andar e sala de todos os prédios. Como os aparelhos são móveis e podem ser ligados e desligados a qualquer momento, a atualização dos dados precisa ser constante. Além disso, o momento da captura das informações precisa obedecer a uma janela de intervalo de tempo para simular uma "fotografia" (*snapshot*) real do ambiente.

As principais vantagens do uso de agentes para o monitoramento de ambientes são [Ilarri 2008]:

• A simplicidade da arquitetura. Não é preciso manter um registro global de todos os computadores envolvidos;

• Agentes estáticos causam *overhead* desnecessário, além da necessidade de configuração adicional sempre que um novo computador (nodo) for inserido na rede. Agentes móveis só precisam de uma plataforma AM disponível;

• A facilidade de incluir novas funcionalidades no monitoramento. É tão simples quanto criar um novo tipo de agente monitor, sem a necessidade de mexer nos computadores da rede;

• Com a mobilidade acontecendo próximo dos sensores, há uma consequente diminuição do tráfego e latência da rede;

• O desempenho dos AM é bom se comparado com os monitores tradicionais do tipo cliente/servidor.

Em ambientes dinâmicos, os AM precisam atualizar os dados com certa frequência. Além disso, o uso de diferentes AM exige que exista certa correlação de tempo na leitura dos dados, ou seja, é preciso sincronizar as tarefas dos agentes. Para isso, os agentes trabalham com deadlines. Os agentes informam para os seus coordenadores não só os dados, mas também a medida de qualidade dos dados. Existem dois indicadores de qualidade: o percentual de dados atualizados e a idade média dos dados. Em um cenário ideal, os dados estariam 100% atualizados. Sendo assim, não basta cumprir os deadlines. É preciso manter a qualidade dos dados num nível aceitável (por exemplo, acima de 80% com um *delay* mínimo).

Uma das técnicas utilizadas para atualização dos dados é atrasar a leitura dos dados para diminuir o *gap* da incerteza. O *gap* de incerteza é o tempo passado entre a leitura de um agente X depois da leitura de um agente Y. Atrasando as leituras é possível minimizar esse *gap*.

O deadline dos AM pode ser absoluto ou relativo. O deadline absoluto é a medida de tempo na qual ele deve retornar a resposta para seu coordenador. O agente raiz não possui um coordenador e, portanto, seu deadline absoluto é a frequência exigida pelo usuário. Os agentes auxiliares atrasarão ao máximo suas leituras na tentativa de diminuir o *gap* de incerteza e obter uma fotografia (*snapshot*) do ambiente numa janela de intervalo aceitável. Os deadlines relativos são utilizados para garantir a sincronia mesmo que os relógios dos computadores não estejam sincronizados.

2.4.3 Algoritmo de Roteamento para Agentes Móveis

Um algoritmo de roteamento de agentes móveis que considera o custo dos enlaces pode ser encontrado na literatura [Qu & Shen 2005]. Para auxiliar na tomada de decisão do agente, existe uma distribuição de probabilidade para o agente móvel selecionar um dos nodos vizinhos e mover-se até ele. O custo do enlace entre dois nodos é definido com base nas informações de tráfego conhecidas e na distribuição de probabilidade encontrada.

Com relação à forma como é definido o itinerário, vários agentes partem em busca do melhor itinerário, voltam ao nodo origem (servidor), e então é selecionado o melhor itinerário, sendo que apenas um agente sai novamente para cumprir sua missão e este já conhece seu itinerário.

Foi apresentado um estudo teórico, portanto não foi definida uma arquitetura que possa ser apresentada graficamente. O objetivo era demonstrar a viabilidade de um algoritmo de roteamento baseado no tráfego de rede.

2.5 COMENTÁRIOS GERAIS SOBRE AS ABORDAGENS

A busca de informação em sistemas distribuídos utilizando agentes móveis provou ser uma alternativa eficiente ao se utilizar uma estratégia de migração de agentes móveis para a recuperação de informação para diversos clientes [Zgaya 2008]. Cada cliente solicita uma informação para o sistema e esta é interpretada como uma tarefa. Cada tarefa é então dividida em sub-tarefas. O principal objetivo da estratégia proposta é o ganho de escala e a minimização do tempo de computação. Isto é obtido otimizando-se o número de agentes e os nodos que cada agente precisa visitar, identificando as sub-tarefas iguais de cada solicitação.

Os agentes móveis também são bastante utilizados para tarefas de monitoramento de ambientes. A dificuldade oriunda da combinação de uma alta frequência de atualização de dados (tempo real) com o fato de que nem sempre os dados são visíveis à distância torna o monitoramento de ambientes uma tarefa desafiadora. Em [Ilarri 2008], é proposta uma técnica com agentes móveis escalonável e capaz de atender requisitos de tempo real. O problema apresentado é o monitoramento de dispositivos

bluetooth em uma universidade e a estratégia dos agentes móveis é do tipo dividir e conquistar. A universidade é dividida em níveis e os agentes trabalham em hierarquias, respeitando as restrições temporais.

Um tópico que continua atraindo o interesse dos pesquisadores é a adaptação de agentes móveis para o roteamento de redes. Em [Qu e Shen 2005] é apresentado um algoritmo de roteamento de agentes móveis que considera o custo dos enlaces.

Diferentemente do que apresentaram as propostas anteriores, neste trabalho a estratégia de uso dos agentes busca combinar a escalabilidade da solução com o contínuo melhoramento do itinerário do agente, ao mesmo tempo em que não se faz distinção do tipo de equipamento em que se encontra o agente, assumindo-se que sempre é possível ter um bom desempenho em equipamentos de baixa capacidade de processamento e pouca memória. Em nenhuma das propostas anteriores existia a preocupação simultânea com esses tópicos.

2.6 CONCLUSÕES

Este capítulo teve por objetivo apresentar os conceitos de agentes móveis. Foram relacionadas características dos agentes, padrões de arquitetura (FIPA) e *frameworks* de mercado (JADE).

A mobilidade, a migração e o roteamento de agentes são temas de interesse desta dissertação, uma vez que estes tópicos serão necessários para desenvolver novas heurísticas de busca de itinerário em tempo real.

3 SISTEMAS DE TEMPO REAL

Neste capítulo são apresentados os sistemas de tempo real e suas principais características, bem como é apresentado o conceito de computação imprecisa e suas técnicas, como miopia de agentes móveis e *anytime algorithm*.

A computação imprecisa visa garantir que os sistemas de tempo real possam ter seus resultados produzidos no momento desejado, fazendo com que haja uma redução da qualidade desse resultado no caso de uma sobrecarga que possa comprometer o tempo de resposta. Neste aspecto, *anytime algorithm* é uma técnica de computação imprecisa que permite que um algoritmo seja interrompido em qualquer etapa de sua execução e mesmo assim produza uma resposta.

3.1 CARACTERIZAÇÃO

Sistemas de tempo real são aqueles em que o funcionamento do sistema depende não apenas do resultado computacional correto, mas também do tempo em que eles são produzidos.

Uma maneira de entender o que é um sistema de tempo real é entender o que ele não é. Existe uma tendência a se acreditar que tempo real significa rapidez. Ainda que um sistema de tempo real deva ser rápido, esta condição está longe de ser suficiente, pois ele precisa ser capaz de atender as suas restrições temporais e para tanto ele necessita ser previsível [Stankovich 1992].

Portanto, a palavra-chave para entender um sistema de tempo real é previsibilidade. Esta é a capacidade de se conhecer o comportamento de um sistema antes mesmo dele ser executado [Rech 2005].

A dificuldade de se implementar um sistema de tempo real varia de acordo com as dimensões desejadas. A literatura descreve as cinco dimensões que um projetista de sistema de tempo real precisa observar: granularidade da restrição temporal, rigidez da restrição temporal, confiabilidade, tamanho do sistema e ambiente [Stankovich 1992].

A granularidade diz respeito ao intervalo de execução das tarefas, do momento em que foi ativada, até ser concluída. Quando a granularidade é pequena, a restrição temporal é dita apertada. Em geral isso acontece em sistemas de tempo real do tipo *hard*. Em sistemas de tempo real do tipo *soft*, a granularidade é normalmente grande e a restrição temporal é dita folgada.

Já a rigidez diz respeito à execução das tarefas quando o prazo da restrição temporal estourou. Em um sistema com granularidade grande (*soft real time*), é comum permitir que as tarefas sejam executadas após o estouro do prazo. Em um sistema de missão crítica e granularidade pequena (*hard real time*), não há razão para se executar nada com o prazo estourado.

A confiabilidade diz respeito ao percentual de tarefas que devem atender às restrições temporais. Sistemas de missão crítica cuja falha possa levar a catástrofes necessitam ter uma confiabilidade de 100%. Sistemas com menor rigidez (*soft real time*) não precisam ser 100% confiáveis no que diz respeito à restrição temporal, pois em geral é admissível que as tarefas continuem sendo executadas após o estouro do tempo. A restrição temporal, nesses casos, é um parâmetro de qualidade do resultado.

As outras duas dimensões citadas por [Stankovich 1992] se referem ao projeto do sistema e seu ambiente de execução. Enquanto um sistema pode ser projetado para ser carregado totalmente em memória e assim depender menos de pontos de falha como disco rígido, entre outros, isso certamente é dependente do tamanho do sistema. Da mesma forma, o ambiente pode ser considerado determinístico se for um ambiente controlado, mas pode não ser e isso certamente adiciona um fator de complexidade ao sistema de tempo real e pode até mesmo torná-lo inviável.

3.1.1 Escalonamento

Uma das técnicas mais utilizadas em sistemas de tempo real é o escalonamento de tarefas [Rech 2005]. Uma vez que um sistema possui recursos limitados, faz-se necessário compartilhar esses recursos entre todas as tarefas em execução. Este compartilhamento é realizado através de técnicas de escalonamento de tarefas.

Assim como um escalonador de processos de um sistema operacional, o escalonador de tarefas do sistema de tempo real também precisa respeitar prioridades e evitar o efeito de inanição (*starvation*) [Tanenbaum 1997]. O efeito de inanição ocorre quando um processo nunca é executado, fica na fila para sempre sem ter chance de executar, pois processos de maior prioridade sempre são executados na sua frente. A diferença é a razão das prioridades, que no caso do sistema de tempo real estão associadas à restrição temporal de cada tarefa.

Isso mostra como o escalonador é essencial em um sistema de tempo real, pois é ele quem compartilhará os recursos dentre as tarefas, permitindo que elas cumpram suas restrições temporais.

3.2 ESCALONAMENTO EM TEMPO REAL

Nesta seção são apresentadas duas técnicas para escalonamento de tarefas em tempo real: computação imprecisa e *anytime algorithms*.

3.2.1 Computação Imprecisa

Para garantir sua característica de previsibilidade, os sistemas de tempo real precisam garantir que as respostas sejam obtidas no momento desejado. Para resolver o problema de escalonamento de tarefas em tempo real, foi proposta a técnica de Computação Imprecisa [Liu 1994], a fim de garantir os requisitos do sistema através da redução de qualidade da resposta da computação.

O conceito de agente móvel (AM) impreciso com restrição temporal foi originalmente apresentado em [Rech 2005], juntamente com algumas heurísticas simples para guiar o agente na definição do itinerário.

Dentre as características de um AM impreciso em cenários de tempo real, uma que certamente merece destaque é a miopia do agente.

Uma questão relevante para a determinação do itinerário de um AM é o seu conhecimento prévio ou não dos nodos que poderão ser visitados. Quando o AM não conhece os possíveis itinerários de antemão, ele é chamado de míope.

E uma vez que o AM é míope, a sua capacidade de controlar o tempo e respeitar sua restrição temporal fica reduzida, exigindo que técnicas de computação imprecisa o auxiliem, em especial a técnica conhecida como *anytime algorithm*.

3.2.2 Anytime Algorithms

No contexto de aplicações distribuídas, existem diversas abordagens que permitem às aplicações de tempo real se adaptar dinamicamente às várias plataformas. A técnica da computação imprecisa ou *anytime algorithms* [Garvey 1994], por exemplo, pode ser empregada com esse objetivo. Nessa técnica, cada tarefa de aplicação é capaz de gerar resultados com diferentes níveis de qualidade ou

precisão. Com esse objetivo, as tarefas são divididas em partes obrigatórias e partes opcionais. A parte obrigatória é capaz de gerar um resultado com qualidade mínima, necessária para manter o sistema operando de maneira segura. A parte opcional refina este resultado, até que ele alcance a qualidade desejada.

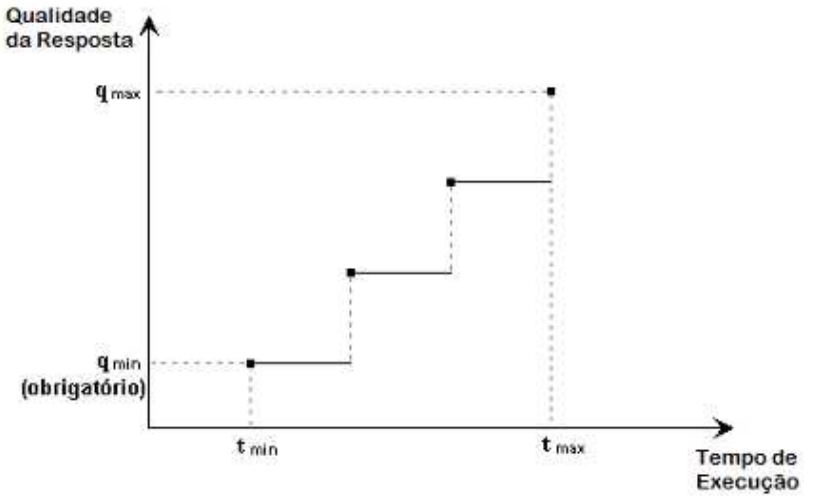

Figura 3.1. Gráfico Q x T para *Anytime Algorithms*

No contexto de aplicações de tempo real, o que muda é a razão de se optar pela qualidade mínima, máxima ou mesmo intermediária. Neste caso a razão é sempre a estratégia adotada pelo agente para não desrespeitar sua restrição temporal.

A Figura 3.1 mostra o comportamento de um algoritmo do tipo *anytime algorithm* ao longo de sua execução. Existe um tempo $\mathbf{t_{min}}$ para que ele produza alguma resposta e também um tempo $\mathbf{t_{max}}$, onde ele produzirá a resposta de qualidade máxima. A partir desse ponto, o algoritmo encerra e não há como melhorar a qualidade da resposta. Nas faixas intermediárias de qualidade, entre $\mathbf{q_{min}}$ e $\mathbf{q_{max}}$, existem pontos de parada específicos do algoritmo, de modo que por mais que o algoritmo possa ser interrompido a qualquer momento de sua execução, a qualidade da resposta só é melhorada nesses pontos de parada específicos.

3.3 APLICAÇÕES COM TEMPO REAL

Nesta seção são apresentadas aplicações com restrição temporal (sistemas de tempo real) presentes na literatura.

3.3.1 Predição de Tráfego no Mundo Real

Equipamentos de navegação para carros têm tratado o problema de planejamento de rotas focando apenas em achar uma rota. Uma abordagem diferente propondo a mudança de paradigma tratando o planejamento de rotas como um problema de otimização de múltiplos objetivos (MOP, em inglês) e propondo uma solução baseada em algoritmo genético (GA) pode ser encontrada na literatura [Kanoh 2008]. São traçados três objetivos: o tamanho da rota, o tempo da viagem e a dirigibilidade da rota.

Atualmente os equipamentos de navegação possuem informes em tempo real sobre congestionamentos e podem dizer tanto a sua localização como o tempo necessário para ultrapassá-lo.

Dos três objetivos do problema, apenas o tempo da viagem cresce muito rapidamente. Os demais não são tão voláteis e a solução proposta assume que os informes sobre o tráfego são suficientes para atualizá-los.

O algoritmo proposto é um híbrido de GA com Dijkstra [Golden 1976]. Dijkstra é uma solução clássica para problemas de origem-destino para minimizar o tempo de viagem. Para cada execução, Dijkstra retorna uma única resposta, portanto é preciso executá-lo com diferentes parâmetros para obter várias respostas e obter o conjunto ótimo de Pareto [Kanoh 2008]. Já o GA consegue o conjunto de respostas em uma única execução e daí a solução híbrida proposta.

GA tem sido utilizado com frequência em MOP para planejamento de rotas, mas nenhuma proposta anterior para ambientes dinâmicos. O uso de GA para MOP em ambientes dinâmicos já foram propostos em outras áreas que não o tráfego de rodovias. Outras propostas de solução para problemas de tráfego tratando os 3 objetivos com o mesmo peso não foram capazes de obter o conjunto ótimo de Pareto.

O GA proposto em [Kanoh 2008] é diferente dos demais por ser especializado para o tipo de problema descrito.

A população inicial é gerada pelo algoritmo de Dijkstra para favorecer vias expressas e estradas largas ao invés de vias arteriais (ruas municipais e outras vielas), a fim de prover soluções estáveis.

Um GA de “infecção” é introduzido para que soluções com combinações de vias arteriais apareçam em gerações sucessivas. A solução parcial de uma via arterial é chamada de vírus e por isso o nome de infecção.

A Figura 3.2 mostra a estratégia de “infecção” no GA, apresentando as rotas antes e depois da infecção, mostrando claramente que o vírus aumenta a capilaridade das rotas, o que representa a inclusão de vias arteriais no mapa de soluções possíveis.

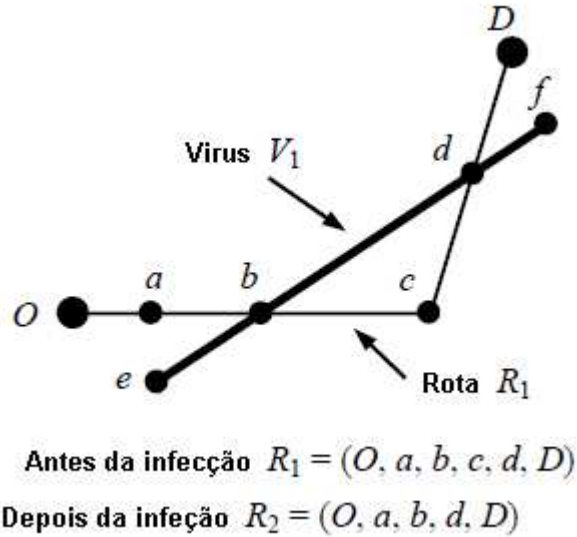

Figura 3.2. Desvio de rota por “infecção” com GA [Kanoh 2008]

A formulação do problema é feita adicionando à minimização do tempo de viagem, a predição de tráfego e um conjunto de restrições. As restrições são classificadas em “mandatórias” (regras de trânsito) e “flexíveis” (dirigibilidade).

Enquanto as restrições mandatórias não podem ser violadas, as flexíveis podem. As rotas com a menor quantidade de penalidades é a rota com melhor dirigibilidade. O custo da dirigibilidade é o somatório de restrições na rota, que são representados por: reduzir número de sinais, seleção de via arterial (municipal), seleção de via principal

(estradas e vias expressas), evitar engarrafamento e reduzir número de curvas.

Em um experimento realizado utilizando a parte central do mapa de Tóquio, obteve-se em dado momento os valores de penalidades para as restrições flexíveis conforme apresentado na Tabela 3.1.

Tabela 3.1. Penalidades para as restrições

Tipo de Penalidade	Pontos
Semáforo	2
Via arterial do estado	2
Via arterial do município	4
Via expressa dupla	1
Via expressa simples	3
Curvas para a esquerda	3
Curvas para direita	7
Curvas de 180 graus	20

O experimento comparou os resultados com dois outros algoritmos que não usam predição de tráfego: SOGA (objetivo simples) e MOGA (que usa MOP).

Com os resultados, constatou-se que ainda que MOGA seja superior a Dijkstra e SOGA, o algoritmo híbrido proposto foi superior a MOGA em todas as situações testadas, obtendo um índice de desempenho de 903 pontos de penalidade, contra 996 pontos do MOGA. A menor quantidade de pontos de penalidade é desejada, pois representa uma rota de melhor qualidade.

3.3.2 Busca A* Adaptativa de Tempo Real (RTAA*)

Agentes móveis precisam de métodos de busca diferentes daqueles utilizados estaticamente no ramo da Inteligência Artificial. Em se tratando de buscas em tempo real em um terreno desconhecido, como precisa fazer um personagem em um jogo de videogame, a busca se resume apenas ao início da trajetória total, limitando o espaço de busca. Para evitar que esta limitação do terreno de busca resulte em ciclos, o

algoritmo RTAA* proposto por [Koenig 2006] utiliza os estados passados para manter o foco no objetivo final.

A fim de atender o requisito de tempo real, a heurística proposta utiliza a técnica *anytime algorithm*, para que fosse possível reduzir o espaço de busca do algoritmo A* tradicional adaptativamente, o que resultou no nome de Busca A* Adaptativa de tempo real.

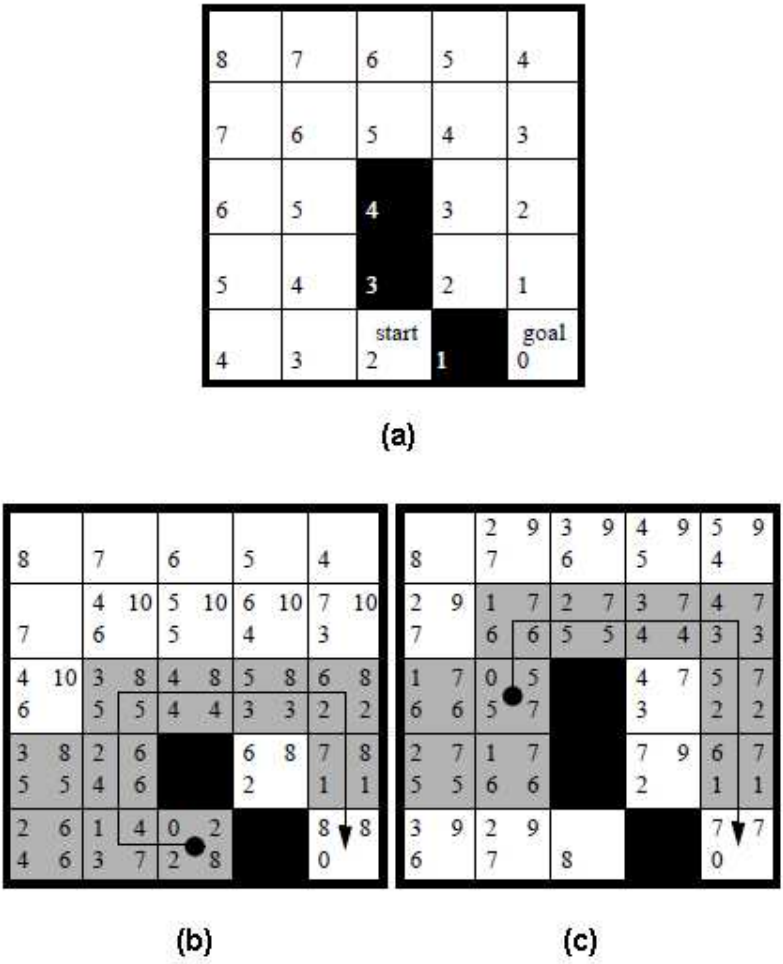

Figura 3.3. Busca com RTAA* [Koenig 2006]

A ideia básica do algoritmo é executar diversas buscas A* no mesmo espaço (terreno) e com o mesmo objetivo final, porém com

origens diferentes, resultando em mais informação para a heurística a cada nova busca.

A cada novo estado alcançado, é executada uma nova busca A* rumo ao objetivo final e uma quantidade limitada de estados à frente (miopia do agente) é testada e o melhor novo estado será escolhido. Essa ação se perpetua até o destino final, quando o algoritmo é encerrado.

A Figura 3.3 apresenta um exemplo do comportamento do algoritmo RTAA* usando um grau de miopia igual a 4, ou seja, o agente só enxerga 4 blocos à frente. Na Figura 3.3.(a) é apresentado o cenário de busca completo. O agente, no entanto, só conhece seu local de origem e o local desejado como destino final, sem saber onde estão os obstáculos, qual o melhor caminho ou mesmo se é possível chegar ao destino desejado. A Figura 3.3.(b) mostra o entendimento do agente no início da missão, onde ele só conhece o universo até um máximo de 4 blocos à frente e faz uma projeção do restante do cenário até o destino final. É importante ressaltar que o agente inclusive assume erros por causa de sua miopia, ignorando a existência de um obstáculo no centro do mapa. A Figura 3.3.(c) apresenta o passo seguinte do agente após ter deixado o bloco de origem. Nesse ponto da execução ele corrige sua percepção do mapa e passa a considerar o obstáculo no centro do mapa.

Com seus experimentos, [Koenig 2006] demonstra alguns teoremas a partir dos resultados: as heurísticas de um mesmo estado são monotonicamente não decrescentes ao longo do tempo, de modo que se tornam melhor informadas no decorrer do tempo, elas permanecem consistentes e o agente alcança o objetivo final.

O algoritmo RTAA* é similar a outra proposta da literatura chamada de LRTA* (Learning Real-Time A*), cuja única diferença é como esses algoritmos atualizam as suas heurísticas após cada busca A*. Essa diferença causa o LRTA* a ser um algoritmo com heurísticas melhor informadas ao longo do tempo, embora seja mais difícil de implementar e leve mais tempo para atualizar suas heurísticas. Mesmo assim, foi demonstrado que para uma miopia de um único estado à frente, RTAA* e LRTA* se comportam exatamente da mesma forma. A vantagem do LRTA* reside num grau de miopia menor (enxergando mais estados à frente), muito embora ao custo de mais tempo de processamento e por isso menor eficiência para cumprir prazos apertados.

Nos testes realizados contra o LRTA* e contra outros algoritmos de busca, verificou-se que o tempo de planejamento do RTAA* era melhor do que os demais, embora o custo da trajetória não fosse a menor.

A principal vantagem do RTAA* reside em sua alta velocidade de planejamento.

3.4 COMENTÁRIOS GERAIS SOBRE AS ABORDAGENS

A definição de itinerário em tempo real tem sido objeto de estudo há bastante tempo e a consolidação de tecnologias como o GPS tem aumentado o interesse dos pesquisadores por este tópico. Em [Kanoh 2008] é proposta uma quebra de paradigma para a busca da menor rota usando equipamentos de navegação de carros: minimizar o tempo da viagem e obter a melhor dirigibilidade possível. A dirigibilidade é definida como o total de penalidades de uma rota. A rota com o menor número de penalidades é a rota com melhor dirigibilidade. As penalidades são calculadas em função das paradas e curvas que um motorista fará pela rota. Engarrafamentos também são computados, assumindo-se que os informes de trânsito são recebidos em tempo real. Um algoritmo híbrido de inteligência artificial combinado a uma solução clássica de problemas de rotas é proposto pelo autor para resolver o problema.

Ainda no ramo da inteligência artificial, algoritmos clássicos de busca têm ganhado versões para tempo real, como é o caso do algoritmo RTAA* [Koenig 2006], uma versão do algoritmo de busca A* para problemas de tempo real. Uma das novidades da proposta é a presença de miopia no agente, o que permite que o algoritmo RTAA* seja utilizado em cenários maiores do que os utilizados pela busca A*, uma vez que não é necessário que o agente conheça todo o mapa de busca. A introdução de técnicas como *anytime algorithms* também se mostrou uma valiosa adição para problemas de tempo real, muito embora o conceito de tempo real adotado em [Koenig 2006] só é válido para cada passo da missão individualmente e não para a missão como um todo.

Diferentemente do que foi apresentado nas propostas anteriores, neste trabalho existe a preocupação de que o conceito de tempo real se aplique à missão inteira do agente móvel, capaz de atender mesmo as missões com deadlines apertados. Da mesma forma, a flexibilidade de adaptação das heurísticas deste trabalho a vários tipos de problema também é um objetivo, diferentemente da alta especialização das propostas anteriores.

3.5 CONCLUSÕES

Este capítulo teve por objetivo apresentar os conceitos de tempo real, em especial as técnicas Computação Imprecisa e *Anytime Algorithms*.

Foram apresentados também trabalhos de busca de itinerário em tempo real utilizando técnicas sem agentes móveis, como é o caso do mecanismo de busca para equipamentos de navegação focados em melhorar a dirigibilidade do motorista [Kanoh 2008], e também utilizando agentes, como é o caso do algoritmo de busca RTAA* [Koenig 2006].

4 ORQUESTRAÇÃO DE SERVIÇOS

Neste capítulo são apresentados os tipos de orquestração existentes, tradicional e híbrida, bem como apresentada uma nova proposta de orquestração híbrida usando agentes móveis.

4.1 COMPOSIÇÃO DE SERVIÇOS

4.1.1 Orquestração Tradicional

O crescimento da Web trouxe também a necessidade de sistemas de informação mais elaborados e, principalmente, mais escaláveis. Para atender a essas necessidades surgiu o conceito da orquestração de serviços, um processo para encadear serviços necessários para a execução de serviços compostos, mais complexos. Por exemplo, uma orquestração que possua um serviço para formação de preço de um produto pode ser composto de diversos outros serviços, como cálculo de imposto, cálculo de frete, entre outros, e cada um destes serviços pode fazer parte de uma aplicação distinta, residindo em um nodo distinto da rede corporativa. Na orquestração centralizada, o orquestrador consultaria cada um desses serviços para retornar com o preço para o cliente. A orquestração de serviços pode ainda prover uma lógica adicional de processamento dos dados obtidos com cada serviço. A orquestração é um dos pilares da arquitetura orientada a serviços (SOA) e é fortemente baseada em XML e Web Services [Fokus & IBM, 1997].

Com a popularização da orquestração de serviços, esta passou a ser utilizada em cenários com requisitos de desempenho mais críticos e passou a demandar novas soluções para atender a estes requisitos. A abordagem tradicional de centralização dos serviços é baseada em linguagens como BPEL (*Business Process Execution Language*) e pressupõe que todo o fluxo do serviço esteja visível e possa ser controlado pelo orquestrador [Kyprianou 2008]. O orquestrador deve verificar os resultados de cada etapa do fluxo do serviço composto e assim direcionar o fluxo conforme as regras do negócio.

4.1.2 Coreografia de Serviços

Conforme a orquestração se aproxima dos sistemas de missão crítica, ela também se aproxima do seu limite de escalabilidade. Na

orquestração tradicional, todas as requisições passam pelo orquestrador, o que muitos acreditam resultar em um fluxo desnecessário de dados pela rede [Barker 2009b].

Para evitar que o orquestrador se torne um gargalo na rede, surge a arquitetura conhecida como coreografia, que embora seja mais complexa, evita que requisições de dados possuam um único destino central e permitem que essas mesmas requisições sejam endereçadas diretamente para onde são exigidas. Um melhor detalhamento da coreografia de serviços pode ser encontrado na literatura [Barker 2009b] e foge ao escopo deste trabalho.

4.1.3 Orquestração Descentralizada

Atualmente, a orquestração continua sendo a arquitetura predominante, pois a coreografia de serviços ainda carece de linguagens e *frameworks* disponíveis. E para lidar com os requisitos de desempenho que a orquestração tradicional já começa a não ser capaz de atender, surge a orquestração descentralizada, uma abordagem que tem sido considerada na literatura como bastante promissora [Binder 2006] e [Barker 2008,2009].

Na orquestração descentralizada, o orquestrador continua sendo a referência corporativa dos serviços compostos, mas não precisa mais controlar cada etapa do fluxo do serviço. Nesta abordagem, cada serviço que compõe o serviço composto possui um pedaço da lógica do processo e é capaz de invocar o próximo serviço do fluxo, sem ter que retornar para o orquestrador até que o fluxo esteja concluído. Trata-se de uma arquitetura híbrida entre a orquestração e a coreografia de serviços.

Um dos motivadores da orquestração descentralizada é o desempenho de sistemas de missão crítica e sistemas de tempo real com restrição temporal (*deadline*). Estes sistemas com requisitos fortes de desempenho precisam obter os benefícios da orquestração centralizada, como a existência de um ponto focal para prover serviços complexos dentro de uma corporação, e ao mesmo tempo atender aos requisitos de restrição temporal. Com esse objetivo já foram propostas soluções híbridas baseadas em tecnologias já consagradas [Barker 2008] [Kyprianou 2008].

4.2 QUALIDADE DE SERVIÇO

Um ambiente real está sujeito a mudanças em tempo real e um agente móvel precisa estar a par destas mudanças para ser capaz de se adaptar e tomar suas decisões de itinerário [FIPA 2002c].

A FIPA define uma ontologia de qualidade de serviço (QoS) para que os agentes possam ser informados das variações no ambiente, seja de forma ativa (fipa-query) ou passiva (fipa-subscribe). Por exemplo, para problemas de itinerário com deadline, um agente precisa ser informado sobre flutuações nas condições de rede ou mesmo sobre a sobrecarga de algum nodo.

A ontologia FIPA [FIPA 2002c] versa sobre protocolos de transporte e canais de comunicação. Alguns dos objetos que podem ter seus QoS monitorados são os seguintes:

• largura (*line-rate*): A largura de banda de um canal de comunicação, em um determinado sentido;

• vazão (*throughput*): O número de bits transferidos com sucesso em um determinado sentido do canal de comunicação. Transferência com sucesso significa que nenhum bit foi perdido, adicionado ou invertido durante a transferência;

• RTT (*round trip time*): tempo necessário para um segmento de dados ser transmitido para uma entidade e a confirmação de recebimento (*ack*) ser enviada de volta à entidade de origem;

• atraso (*delay*): o atraso nominal para um segmento de dados ser enviado para outra entidade;

• estado (status): estado da conectividade de um canal de comunicação. Pode assumir os seguintes valores: conectado, desconectado ou conectando.

É importante ressaltar que a qualidade de serviço especificada pela FIPA serve para monitorar o ambiente e não para estabelecer um nível de qualidade de serviço para a orquestração de serviços ou para os serviços orquestrados. Cada solução deve propor métricas adicionais a fim de monitorar os serviços do sistema adequadamente.

4.3 SOLUÇÕES DESCENTRALIZADAS

Nesta seção são apresentadas soluções de orquestração descentralizada presentes na literatura que não utilizam agentes móveis.

4.3.1 Solução baseada em *Triggers*

A arquitetura orientada a serviços permite a construção de aplicações distribuídas através da composição de serviços já existentes na Internet [Papazoglou 2003].

Em [Binder 2006] é apresentada outra solução híbrida utilizando o conceito de *Service Invocation Triggers*, ou simplesmente *Triggers* (gatilhos). Nesta solução é proposta uma tecnologia leve de *Triggers* para executar os repasses do fluxo de dados, utilizando um controle descentralizado do serviço.

Na Figura 4.1.(a) é apresentada a arquitetura tradicional de orquestração, enquanto que na Figura 4.1.(b) é apresentada a arquitetura baseada em *triggers*, a fim de melhor compará-las. Na Figura 4.1.(a), o orquestrador C solicita a execução de um serviço Sa e recebe como resposta as saídas Ob e Oc, para então chamar os serviços Sb e Sc. Esses passos adicionais de recebimento de respostas intermediárias não existem no fluxo da Figura 4.1.(b), pois as saídas Ob e Oc são enviadas diretamente para os gatilhos Tb e Tc, os quais solicitam a execução dos serviços Sb e Sc antes de retornarem para o orquestrador.

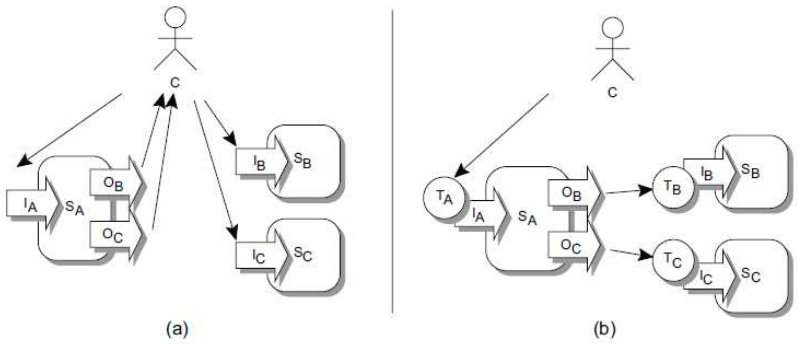

Figura 4.1. Arquitetura baseada em Triggers [Binder 2006]

O principal objetivo desta arquitetura é demonstrar as vantagens da descentralização. Neste modelo, uma *Trigger* realiza diferentes tarefas para alcançar a orquestração descentralizada:

• Recebe os parâmetros de entrada para cada um dos serviços;
• Atua como um *buffer*, já que os parâmetros de entrada podem ter origens distintas;
• Dispara a chamada ao serviço, após receber e sincronizar os parâmetros de entrada;
• Define o roteamento de cada parâmetro de saída do serviço, já que eles podem ter destinos distintos e até mesmo múltiplos destinos.

Na prática, o modelo de *Triggers* funciona como um *proxy* especializado. Ele difere de um *workflow* tradicional, pois funciona como um *workflow* distribuído, uma vez que as *Triggers* são encadeadas e não retornam para um mesmo orquestrador central.

[Binder 2006] apresenta uma implementação das *Triggers* usando Java RPC. Chega até mesmo a mencionar a tecnologia de agentes móveis como opção, mas focam as questões de segurança dos agentes móveis para justificar a sua não utilização.

Apesar da arquitetura de *Triggers* ser bastante consistente, a implementação proposta já se encontra defasada e os próprios criadores desta arquitetura apontam problemas de configuração do ambiente, uma vez que cada nodo da rede de serviços precisa de uma configuração inicial independente.

O maior êxito do trabalho foi sugerir uma orquestração descentralizada e elaborar uma arquitetura compatível com a tecnologia de agentes, mas ao contrário do que acreditava [Binder 2006] à época, a tecnologia de agentes evoluiu bastante nas questões de segurança (muitos *frameworks* de agentes hoje em dia utilizam os padrões de segurança de *Web Services*), enquanto que a tecnologia Java RPC utilizada na implementação de *Triggers* já não é tão popular.

4.3.2 Solução Peer-to-Peer

Enquanto uma arquitetura centralizadora como a orquestração tradicional deve ser evitada em sistemas em que trafegam altos volumes de dados, [Barker 2008] apresenta os benefícios de *frameworks* baseados em Web Services, como sua ampla difusão e contínuo suporte. A razão dessa constatação é apresentar uma solução híbrida utilizando parcialmente o modelo *peer-to-peer*. A solução apresentada é baseada nos padrões de *workflow* do Montage [Jacob 2004], um *software* para análise de imagens astronômicas desenvolvido pelo Instituto de Tecnologia da Califórnia (Caltech) para a NASA (*National Aeronautics and Space Administration*).

Assim como no Montage, [Barker 2008] propõe o controle centralizado do *workflow*, mas admite um fluxo de dados descentralizado, de forma que as tarefas possam repassar dados uma para a outra durante a execução de um serviço, sem com isso tirar o controle do *workflow* do orquestrador.

A arquitetura sugerida é mostrada na Figura 4.2, onde se utiliza um conjunto de *Proxies* para permitir que as respostas de controle retornem para o orquestrador, mas os dados possam ser transmitidos diretamente para um próximo *proxy*, responsável pela próxima tarefa.

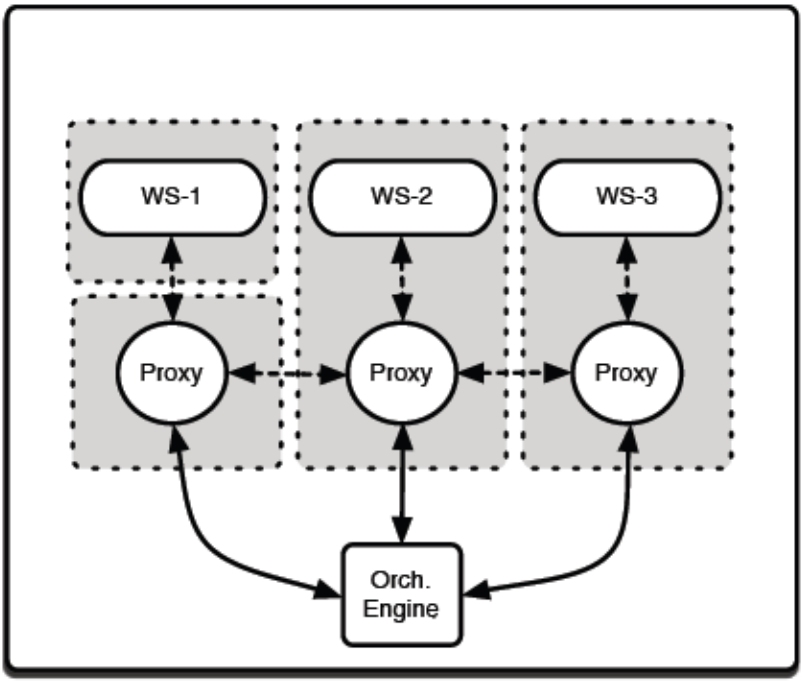

Figura 4.2. Arquitetura peer-to-peer [Barker 2008]

Para este fluxo de dados descentralizado é utilizada uma rede de *Proxies* baseada no modelo *peer-to-peer*. Embora superior à orquestração centralizada em termos de desempenho, a complexidade para manutenção do ambiente e o fato de que cada par da rede *peer-to-peer* é um orquestrador per si torna o resultado mais complexo do que atrativo.

No entanto, existem problemas específicos em que esta possa ser uma solução vantajosa, particularmente os problemas em que o arquiteto disponha de bastante controle do ambiente, inclusive dos serviços que farão parte da composição de serviços do orquestrador.

4.4 COMENTÁRIOS GERAIS SOBRE AS ABORDAGENS

Diversos trabalhos na literatura propõem alternativas à orquestração de serviços, procurando endereçar o problema de desempenho e escalabilidade causado pela centralização da orquestração tradicional. Em [Barker 2008] é apresentada uma solução híbrida utilizando o modelo peer-to-peer como forma de descentralizar a orquestração de serviços. Ela é baseada nos padrões de workflow do Montage, um sistema de análise de imagens astronômicas da Caltech.

Embora encontremos em [Binder 2006] uma arquitetura consistente, o fato da nova tecnologia de triggers apresentada não estar embasada em nenhum padrão maduro de mercado torna a tecnologia em si um problema, fazendo com que a maior contribuição de [Binder 2006] seja a arquitetura da solução e não a tecnologia desenvolvida. Os autores chegam a mencionar a tecnologia de agentes móveis, mas focam as questões de segurança dos agentes ao invés de avaliá-los como potencial solução para o problema.

Neste trabalho também não abordamos as questões de segurança e permissão de acesso dos agentes. Existem muitos outros trabalhos específicos sobre o assunto e trata-se de um campo com pesquisa contínua, o que nos permite avaliar a tecnologia de agentes móveis para orquestração de serviços isoladamente, mostrando que a mesma é uma excelente alternativa para a orquestração descentralizada, com um ambiente mais fácil de configurar e manter do que as propostas anteriores.

4.5 CONCLUSÕES

Este capítulo teve por objetivo apresentar os conceitos da Orquestração de Serviços, tradicional (centralizada) e descentralizada.

Foram apresentadas opções de orquestração descentralizada sem agentes móveis, que servirão para contrapor a orquestração com agentes móveis que será apresentada no próximo capítulo.

5 HEURÍSTICAS ADAPTATIVAS

Neste capítulo são revisadas as heurísticas para definição do itinerário de agentes móveis presentes na literatura e que servirão de base de comparação para a nova heurística apresentada aqui: a heurística *Lazy-Adaptativo*.

5.1 DESCRIÇÃO DOS RECURSOS

Os projetos de engenharia de software costumam se preocupar com os aspectos funcionais do sistema, mas em geral deixam em segundo plano os aspectos não funcionais, em especial aqueles associados com a qualidade do serviço (QoS), como confiabilidade, disponibilidade, performance, segurança e tempo de resposta [Frolund 1998].

Um dos motivos para que a qualidade de serviço seja subestimada pela maioria dos projetistas até os dias de hoje é que a definição de quais propriedades de QoS devem ser controladas no sistema é variável de sistema para sistema.

Neste trabalho estamos interessados em serviços com qualidade de serviço variável, ou seja, compatíveis com a estratégia de *anytime algorithms* [Garvey 1994]. Para cada nível de QoS é definida uma pontuação que um serviço retorna ao ser executado. Este serviço está diretamente associado a um ou mais recursos. Um recurso é uma abstração e pode corresponder a um processador, dispositivo, arquivo, estrutura de dados, etc.

Por exemplo, um determinado recurso R1 pode ter um nível de QoS máximo igual a 10. Para que ele possua qualidade variável é necessário que ele seja composto de partes e que apenas uma dessas partes seja de execução obrigatória. Suponha que R1 seja composto de cinco partes iguais, de modo que um algoritmo do tipo *anytime algorithm* poderia obter um nível de QoS mínimo igual a 2, dependendo do tempo de execução disponível. Esta é a estratégia dos *anytime algorithms*, executar um serviço dentro de um limite de tempo preestabelecido e interromper a execução do serviço assim que esse tempo é atingido (na prática, espera-se a execução do bloco atual e só então se interrompe o serviço).

O nível de qualidade de serviço também pode ser mensurado em uma escala arbitrária única para todos os recursos, a fim de que seja possível executar diferentes recursos e obter um nível global de QoS.

Suponha agora que um dispositivo móvel necessita fazer consulta de preços de um produto na Internet. Como esse dispositivo móvel possui restrições de acesso à Internet, ele dispara um agente móvel para que este faça a consulta de preços na Internet em tempo real, enquanto o dispositivo se desconecta e economiza energia. O agente móvel possui um deadline apertado, pois precisa concluir sua pesquisa até o momento em que o dispositivo móvel voltará a se conectar à Internet para receber a resposta. A missão do agente é buscar no maior número de sites possível o produto desejado para que possa indicar o menor preço como resposta. Ao receber missões sucessivas, o agente pode melhorar seu itinerário priorizando os sites que regularmente forneçam os melhores preços. Na prática, a missão é formada por um grupo de recursos que devem ser utilizados respeitando-se um deadline.

Para levar este exemplo ao nível dos recursos variáveis, seria necessário, além de fornecer os preços dos produtos, que os servidores admitissem que agentes inteligentes negociassem o preço com eles, fazendo uma espécie de leilão reverso com ofertas encontradas em outros servidores de comércio eletrônico. Neste novo cenário, o preço passa a ser um recurso variável, pois um agente pode gastar mais tempo em um servidor e assim melhorar a qualidade de sua resposta. Cada servidor possui um limite de barganha e também diferentes estratégias de barganha, de modo que cabe ao agente descobrir as melhores rotas para obter as melhores negociações.

5.2 SOLUÇÃO DESCENTRALIZADORA

Na orquestração tradicional (centralizada), o orquestrador pode se tornar o gargalo dos sistemas que dependam dele, pois todos os serviços compostos estão centralizados nele. Essa centralização certamente será um problema quando os requisitos de desempenho forem muito fortes. Na abordagem híbrida usando agentes móveis (AM), o orquestrador dispara um AM na rede que passa a ter como missão a formação do preço. Imaginando uma rede muito heterogênea, o AM pode ter opções como buscar a cotação mais atual do dólar diretamente do Banco Central ou usar um servidor local que atualiza este dado de seis em seis horas. Neste cenário, a seqüência de nodos visitados por um AM para

execução da missão é chamada itinerário, e o itinerário será definido pelo balanceamento entre deadline da missão e nível de QoS desejado.

É com este mesmo objetivo que é apresentada uma solução híbrida para a orquestração utilizando AM.

Por ser capaz de migrar para a localização do serviço desejado, um agente móvel pode responder a estímulos rapidamente e pode continuar sua interação com o serviço se a conexão de rede cair temporariamente. Estas características fazem com que os agentes móveis tornem-se atrativos para o desenvolvimento de aplicações móveis, as quais frequentemente se deparam com largura de banda limitada, alta latência e enlaces de rede não confiáveis.

A solução híbrida de orquestração com AM atende aos requisitos de uma orquestração descentralizada para ganho de desempenho e cumprimento dos deadlines. No entanto, está fora do escopo deste trabalho detalhar a arquitetura da orquestração descentralizada utilizando agentes móveis. A arquitetura base utilizada para os AM é similar às soluções apresentadas em [Binder 2006] e [Barker 2008,2009], com a vantagem de que os agentes móveis continuam sendo controlados de forma centralizada (Através do contêiner de AM), resolvendo um dos principais problemas da descentralização: a configuração do ambiente.

5.3 HEURÍSTICAS NA LITERATURA

Antes de apresentar a nova heurística para definir dinamicamente o itinerário de agentes móveis míopes com restrição temporal, são apresentadas nesta seção as heurísticas existentes na literatura e que serão usadas posteriormente para comparação dos resultados.

A definição de heurísticas adaptativas utiliza três das heurísticas não-adaptativas que são minuciosamente descritas em [Rech 2005] e avaliadas em [Rech 2008a].

Em [Rech 2008] é descrita e avaliada a heurística PG (Preguiçoso-Guloso) com adaptação na partida, onde uma vez escolhido o comportamento o agente permanecerá com ele até o final da missão.

5.3.1 Heurística Preguiçosa (Lazy)

Esta heurística tenta executar os recursos o mais rápido possível, ignorando o nível de QoS máximo de cada recurso. Os

recursos opcionais não são executados e os recursos com estereótipo <<variável>> são executados com o menor tempo possível.

5.3.2 Heurística Gulosa (Greedy)

Sempre que existir uma rota alternativa, ele irá escolher a que representar em um maior nível de QoS para a missão, sem nenhuma preocupação com o tempo de execução (deadline). Recursos opcionais sempre serão executados e recursos do tipo <<variável>> serão executados com o tempo de computação necessário para atingir a maior pontuação do nível de QoS possível.

5.3.3 Heurística Ponderada (Higher Density)

Esta heurística escolhe como próximo recurso a ser executado aquele que apresentar a melhor relação custo/qualidade. Recursos opcionais somente serão executados se sua densidade (relação entre a pontuação de QoS adquirida e o custo de execução) for superior à densidade média da missão até aquele momento.

5.3.4 Heurística Preguiçoso-Guloso (Lazy-Greedy)

Esta heurística é baseada na utilização de duas heurísticas simples: Preguiçoso e Guloso. A heurística constrói um histórico formado apenas pelos tempos de resposta das execuções anteriores (para a formação do histórico desta heurística, os deadlines das execuções anteriores e a informação se os deadlines foram cumpridos ou não são irrelevantes).

Na sua primeira execução, o agente móvel sempre escolhe o Algoritmo Guloso – esta missão é então salva no histórico. Quando surge uma nova missão, seu deadline é comparado com os tempos de resposta anteriores armazenados no histórico, e assim estima-se $P(R \leq D)$, ou seja, a probabilidade da missão atual ter o seu deadline atendido. Esta estimativa é comparada com um limiar e o resultado desta comparação decide o comportamento que o agente irá exibir.

Quando a probabilidade de sucesso está abaixo do limiar, o agente assume o comportamento Preguiçoso. Mas se a probabilidade de sucesso estiver acima do limiar, o comportamento Guloso é adotado pelo agente. O valor de $P(R \leq D)$ é calculado simplesmente através da contagem do número de missões do histórico que teriam atendido o deadline atual e da divisão desta contagem pelo número total de missões

do histórico. Neste cálculo é desconsiderado qual heurística foi utilizada no passado pelas missões registradas no histórico.

O valor escolhido como limiar para a troca de algoritmos entre Guloso e Preguiçoso é uma constante arbitrária. A qualidade dos resultados obtidos está diretamente relacionada à calibragem desta constante. Apesar dessa desvantagem, a complexidade computacional deste algoritmo é *O(n)*, sendo que n é o tamanho do histórico. Além disso, esse algoritmo apresenta robustez com relação à escolha do algoritmo do agente, mesmo com poucas execuções e com histórico vazio. Caso uma escolha anterior tenha sido feita erroneamente no algoritmo Preguiçoso, a média dos tempos de resposta armazenados no histórico diminui em função do comportamento Preguiçoso, e a probabilidade de escolha do Guloso aumenta na sua próxima execução. Fato similar ocorre quando há uma escolha "errada" no algoritmo Guloso. A Figura 5.1 apresenta um esboço do algoritmo da heurística PG.

```
const threshold = 0,9;

Scan_log_to_compute P( R  ≤ D )

if P(R ≤ D ) > threshold
    behavior = guloso;
else
    behavior = preguiçoso;
```

Figura 5.1. Pseudocódigo da Heurística PG.

5.4 HEURÍSTICA LAZY-ADAPTATIVO

Este algoritmo é baseado na utilização da heurística simples Preguiçoso, porém com a capacidade de melhorar a escolha dos próximos nodos usando seu histórico. O objetivo principal desta heurística é atingir os maiores índices de pontuação global do nível de QoS permitidos para qualquer deadline, inclusive os deadlines muito apertados. A heurística constrói um histórico formado pelos deadlines alcançados para comportamentos utilizados em cada missão, a fim de poder repeti-los para cumprir deadlines semelhantes.

```
{Agente}
Variáveis compartilhadas entre agentes:
 1: initialStage = 0;  { nodo inicial da missão }
 2: untestedBehavior = lazy; {comportamento
agressivo nao testado }

Variávies locais de cada agente:
 3: behavior = null; {comportamento padrão do agente}
 4: nextStage = null; {proximo nodo}
 5: futureStage = null; {nodo escolhido para futuro
comportamento agressivo}

procedure on ArrivalCalcNextStage (currentStage,
stageGraph, missionDeadline, history)

 6: if isNewMission(currentStage, initialStage) then
 7:  if isDeadlineReachable(missionDeadline, history)
then
 8:   behavior <- getHistoricalBehavior(history)
 9:  else
10: behavior <-
getUntestedMoreAgressiveBehavior(untestedBehavior)
11:  end if
12: end if

13: nextStage <- calcNextStage(behavior, stageGraph)
14:   futureStage   <-   calcFutureStage(nextStage,
stageGraph)

15:  if  isMissionFinished(nextStage,  initialStage)
then
16: updateHistory(behavior, history);
17:  untestedBehavior <-
     createMoreAgressiveUntestedBehavior
                              (behavior,futureStage)
18: end if
```

Figura 5.2. Pseudocódigo da Heurística Lazy-Adaptativo.

O histórico armazena apenas 50 comportamentos de missão e deadlines. Uma questão importante é o tamanho mínimo para o histórico poder ser considerado confiável. As heurísticas adaptativas foram

avaliadas utilizando históricos de tamanhos variados entre 10 e 1000 com o objetivo de estabelecer um tamanho de histórico consistente, que seja confiável e evite armazenamento desnecessário, consequentemente economizando processamento. Partindo dos resultados obtidos pelas avaliações realizadas, percebeu-se que o histórico contendo de 50 à 60 execuções passadas, seria suficiente para garantir que as heurísticas adaptativas pudessem escolher o melhor comportamento para a missão atual.

Inicialmente, o agente sempre escolhe o comportamento do algoritmo Preguiçoso, isto é, executa o mínimo possível do recurso variável, e à medida que percebe a possibilidade de mudar seu comportamento para obter maior pontuação global do nível de QoS, utiliza o histórico das missões para melhorar sua escolha em um nodo determinado em missão anterior. Ao alcançar esse nodo, ele não irá utilizar o comportamento Preguiçoso e sim um comportamento um pouco mais agressivo (intermediário entre o comportamento Preguiçoso e o Guloso) para obter maior pontuação naquele ponto. Esta troca de comportamento é prevista no início da missão, dependendo do deadline, porém apenas no decorrer do itinerário é que será possível confirmar se houve chance de se utilizar esse comportamento mais agressivo. A Figura 5.2 descreve o algoritmo da heurística *Lazy-Adaptativo*.

Em comparação com a heurística PG apresentada anteriormente, existe aqui um pequeno *overhead* de processamento no decorrer da missão devido ao cálculo do comportamento do agente efetuado a cada nodo, porém este *overhead* é mínimo, não afetando o resultado global da heurística e nem o objetivo do agente.

O método ArrivalCalcNextStage verifica sempre se uma nova missão está sendo iniciada para assim estabelecer o comportamento que será adotado pelo agente para cumprir a missão: comportamento Preguiçoso, comportamento presente no histórico ou comportamento mais agressivo.

Após estabelecer o comportamento inicial da missão, a cada novo estágio do itinerário, o método calcula o próximo estágio (calcNextStage) e também verifica se deseja marcar algum estágio do itinerário para um comportamento mais agressivo no futuro (calcFutureStage). Em seguida, acontece uma nova verificação para saber se a missão terminou. Caso tenha terminado, ele atualiza o histórico com o novo comportamento e deadline, e também monta um novo comportamento mais agressivo, com o objetivo de alcançar mais benefícios nas missões futuras, utilizando o comportamento da missão

corrente, mais o estágio marcado como futureStage, o estágio que representa o melhor avanço em busca de mais benefícios.

5.4.1 Exemplo com Lazy-Adaptativo

Para melhor entender a heurística Lazy Adaptativo, vamos propor um exemplo de uma plataforma contendo 12 nodos de recursos, sendo um dos nodos a origem do agente, que também é o nodo final da missão, ou seja, o agente retorna para a origem ao final da missão. A missão é composta por 4 tipos de recursos, que podem ser encontrados em mais de um nodo da rede. Para facilitar a compreensão, os recursos desse exemplo não são do tipo variável, ou seja, são compostos de uma única parte e ela deve ser executada em sua totalidade.

No exemplo, os nodos N1, N2 e N3 são do recurso tipo 1 (R=1), os nodos N4 e N5 são do recurso tipo 2 (R=2), os nodos N6, N7, N8 e N9 são do recurso tipo 3 (R=3) e os nodos N10 e N11 são do recurso tipo 4 (R=4). A Figura 5.3 mostra os nodos dessa plataforma divididos por tipos de recursos e com o caminho da primeira viagem (comportamento preguiçoso) marcado graficamente.

A missão do agente consiste em navegar por um nodo de cada tipo de recurso no menor tempo possível (dentro da restrição temporal) e com a maior pontuação global de QoS possível. Apesar de que na prática o tempo de execução de um recurso não esteja relacionado somente à qualidade de serviço do mesmo, em nosso exemplo existe a equiparação entre QoS e tempo de execução para todos os nodos.

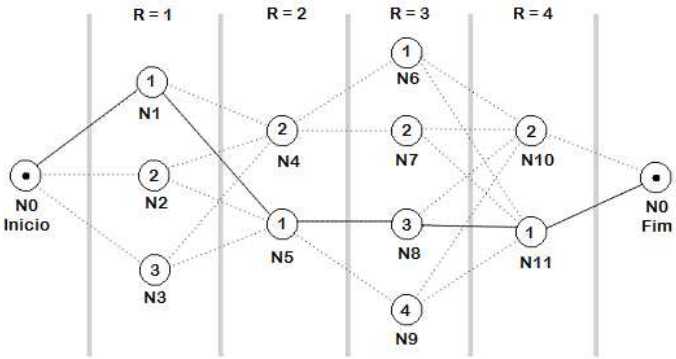

Figura 5.3. Primeira viagem com Lazy-Adaptativo.

Na Figura 5.3, os nodos possuem o tempo (em Unidades de Execução) gravado dentro do círculo do nodo, e cada unidade de tempo corresponde a um ponto do nível de qualidade do serviço, ou seja, maior o tempo de execução, maior a pontuação de QoS obtida. Por exemplo, o nodo N2 gasta 2 unidades de tempo para obter 2 pontos de QoS para a missão, o nodo N9 gasta 4 unidades de tempo para obter 4 pontos de QoS, e assim acontece com todos os nodos da figura.

Vamos supor agora uma missão com restrição temporal de 9 unidades de tempo. A heurística Lazy Adaptativo começa como uma heurística Lazy tradicional e a cada viagem torna-se mais agressiva, conforme exista tempo de sobra para cumprir a missão com um benefício acumulado maior. Essa maior agressividade é obtida mapeando os nodos da viagem e escolhendo aquele capaz de dar uma maior pontuação de QoS em um dos tipos de recursos da missão.

A Figura 5.4 mostra o gráfico dessa plataforma, os itinerários de viagem e também os desvios de itinerário das viagens consecutivas. Quanto mais grossa a linha do itinerário, mais recente o desvio. Por exemplo, a linha mais fina corresponde à primeira viagem e a linha mais grossa corresponde ao desvio gerado na última viagem.

Na primeira viagem, seguindo um comportamento preguiçoso tradicional, o agente percorre os nodos da seguinte forma: N1, N5, N8 e N11, com uma qualidade total de 6. Na segunda viagem, o algoritmo marcou os recursos do tipo 1 para ser mais agressivo, e por isso fará um desvio apenas quando buscar este recurso. O resultado é um novo itinerário com uma qualidade total de 7: N2, N5, N8 e N11. Na terceira viagem, como ainda houve sobra de tempo, o agente escolhe o recurso de tipo 2 para ser mais agressivo. Ao fazer isso, o agente se encontra em outro enlace da rede ao ir para os recursos do tipo 3. O novo itinerário é N2, N4, N6 e N11. A nova qualidade total é de 6, ou seja, houve um retrocesso na pontuação total de QoS. Mas a heurística identifica uma possibilidade de continuar progredindo neste novo enlace e segue para uma quarta viagem, obtendo um novo itinerário com qualidade total de 7: N2, N4, N7 e N11. E para finalizar o exemplo, em sua quinta viagem, o agente consegue uma pontuação total de QoS igual a 8, com o seguinte itinerário: N2, N4, N7 e N10.

Repare que na heurística *Lazy-Adaptativo*, o agente executa um passo de melhoria por vez a fim de evitar problemas com diferentes enlaces da rede. No exemplo, existia o risco de estourar o deadline da missão na terceira viagem, pois a troca de enlace poderia jogá-lo para uma parte da rede com recursos computacionalmente mais pesados e o

tempo total da missão poderia passar das 9 unidades (pontuação máxima) preestabelecidas.

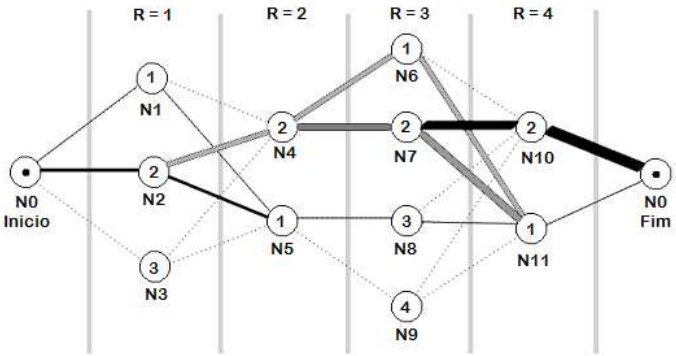

Figura 5.4. Viagens consecutivas com a heurística Lazy-Adaptativo.

5.5 MODELO DO EXPERIMENTO

Nesta seção é apresentado o modelo do experimento em que serão utilizadas as novas heurísticas e também são apresentadas as avaliações de desempenho das heurísticas adaptativas e das heurísticas com adaptação na partida propostas neste trabalho. O objetivo destes experimentos é comparar o comportamento das heurísticas e avaliar sua eficiência em diferentes cenários e diferentes faixas de deadline.

Para estes experimentos foram utilizados 18 computadores de arquitetura x64 com velocidade de processamento entre 2.0GHz e 3.2GHz, com 2.0 Gb de memória RAM.

A rede utilizada foi uma rede *wireless* padrão 802.11g com velocidade de transmissão de 54 Mb/s. Cada um dos computadores foi equipado com uma máquina virtual Java com JDK 6.15 (*Java Development Kit*) para suportar a plataforma JADE, o ambiente virtual padrão FIPA/IEEE (*Foundations of Intelligent Physical Agents*) para executar aplicações baseadas em agentes móveis.

O principal objetivo desta modelagem é determinar o cenário para novas heurísticas para definição do itinerário de AM durante o

fluxo de um serviço composto. Os fluxos dos serviços compostos são aqui representados pela missão do AM. Cada missão é composta de tarefas, que correspondem aos serviços internos de cada serviço composto. Neste contexto, as heurísticas de definição de itinerário foram projetadas de forma a garantir que as missões dos agentes sejam realizadas em tempo para que seus resultados tenham utilidade para a aplicação.

Para demonstrar a complexidade dos ambientes corporativos que podem ser tratados com maior facilidade pelos AM, neste trabalho admite-se que os serviços possuam redundância, *clusters* e balanceamento de carga, ou seja, o AM pode escolher um itinerário distinto para completar uma mesma missão, dependendo do deadline de sua missão ou do nível de QoS desejado.

Esta modelagem é equivalente ao modelo computacional apresentado em [Rech 2005]. Este modelo descreve o comportamento de aplicações baseadas em agentes móveis imprecisos com restrição temporal, em um sistema distribuído formado por um conjunto de nodos conectados através de um serviço subjacente de comunicação.

Neste modelo computacional, cada agente móvel possui uma missão associada com a visita a um determinado grupo de nodos do sistema. Nestes nodos encontram-se os recursos necessários para o agente completar sua missão. Será assumido que os agentes não se comunicam.

Um agente móvel impreciso é aquele capaz de reduzir a qualidade do resultado para conseguir atender o deadline da missão. Neste trabalho o diagrama de recursos corresponde a uma rede de serviços em contêineres de uma plataforma JADE.

A ordem de precedência dos recursos a serem utilizados pelo AM é definida dinamicamente para cada missão. Para melhorar a quantidade de opções em uma missão, vários recursos podem aparecer replicados no sistema, a fim de que quando o AM for visitar um recurso R ele terá diferentes opções de nodos, com granularidades e níveis de qualidade diferentes.

Cada recurso executado representa uma pontuação adicional para o nível de QoS global da missão do agente. O objetivo é maximizar o somatório dos níveis de qualidade obtidos ao longo do itinerário. Entretanto, o agente móvel é suposto míope, ou seja, ele conhece apenas as opções imediatas dentre os nodos da sua missão. A miopia do agente impede que o problema seja tratado através de técnicas clássicas de otimização.

Outro requisito que deve ser considerado pelo agente móvel é a restrição temporal de cada missão, salientando a importância da escolha do melhor itinerário para cada situação apresentada. Esta sequência de nodos visitados pelo agente móvel entre o nodo origem e o nodo destino (itinerário) é definida dinamicamente.

5.6 FORMULAÇÃO DO PROBLEMA

Cada agente móvel possui uma missão M que define a função de pontuação Q do nível de QoS para cada tipo de recurso R. Ou seja, esta função Q determina a pontuação máxima q_i do nível de QoS que cada tipo de recurso r_i contribui para a missão do agente. Na prática, estabeleceu-se uma escala arbitrária para comparação dos diferentes tipos de recursos (dispositivo, arquivo, estrutura de dados, etc.) em relação à missão do agente, de forma que a função de pontuação Q possa ser otimizada e sirva como parâmetro de comparação entre os diferentes itinerários possíveis.

A ordem de precedência dos recursos em uma missão é preestabelecida antes do início da mesma. Qualquer execução de recurso em desacordo com esta precedência não traz nenhum benefício para a missão. O diagrama de nodos é baseado nos contêineres de uma plataforma JADE e cada nodo é responsável por um ou mais recursos do sistema.

O cumprimento de uma missão M está relacionado à escolha de um itinerário I. Para definição do itinerário são considerados: o tempo de computação para benefício máximo de cada tarefa; a latência de comunicação entre os nodos; as dependências entre os recursos que aparecem na forma do diagrama de recursos da missão. Devem ser analisados os custos para que o agente chegue ao nodo (vindo do nodo anterior) e para adquirir aquele recurso ($C_i + T_i$ em que C_i é o tempo de execução no nodo e T_i é o tempo na fila do processador local esperando para executar).

Para todo recurso $r_i \in R$, a pontuação do nível de QoS $q_{i,t}$ obtida por executar o recurso r_i durante o intervalo de tempo t é dado por:

$$q_{i,t} = q_i * min(1, t/Ci) \qquad (1)$$

Sendo q_i a pontuação máxima obtida de r_i e C_i o tempo máximo de computação associado com r_i (C_i é o tempo que o agente deve ficar com o recurso para ganhar q_i e t é o tempo que ele realmente ficou).

O nível de QoS global Q da missão é obtido através do somatório das pontuações dos níveis de QoS obtidos a partir da execução de cada recurso no itinerário:

$$Q = \sum q_i(y) \quad \text{para todo} \quad r_i \in R \qquad (2)$$

Sendo $q_i(y)$ a pontuação obtida pela utilização do recurso r_i que o agente visitou para o cumprimento de sua missão ao seguir o itinerário y, Q é a pontuação total obtida pela utilização de todos os recursos da missão.

O objetivo do algoritmo de definição de itinerário é maximizar o valor de Q, respeitando a restrição temporal D da missão.

Os agentes móveis carregam os resultados obtidos nos nodos visitados, mas com relação ao seu tamanho é assumido que o crescimento é pequeno. Por este motivo, o tamanho do agente não é considerado no modelo computacional.

5.7 EXPERIMENTO COM HEURÍSTICA ADAPTATIVA

Neste primeiro experimento, a heurística Lazy-Adaptativo é avaliada e seu desempenho é comparado às heurísticas Preguiçosa, Gulosa, Ponderada e Preguiçosa-Gulosa, com o objetivo de encontrar qual a faixa de restrição temporal que ela melhor se adéqua. As condições do experimento são similares às encontradas na simulação apresentada em [Rech 2008].

5.7.1 Condições do Experimento

Para este experimento foi considerando um sistema composto por 10 nodos e 15 recursos. Os recursos estão alocados de forma fixa nos nodos, sendo que alguns são replicados em outros nodos. A arquitetura utilizada foi a arquitetura padrão JADE e embora os nodos estejam ligados fisicamente, foram definidas missões contendo relações de precedência entre nodos e recursos. A missão do agente é definida por

sorteio e não admite reposição de recursos. Este sorteio oferece 1.000.000 combinações diferentes [Rech 2008].

Para o cálculo do tempo total da missão foram considerados: o tempo de fila do processador, o tempo de uso do recurso, com valor fixo por recurso (porém entre os recursos existe uma distribuição uniforme entre 1 e 15) e o tempo de fila nos enlaces de comunicação.

Após a execução de cada recurso, um valor referente ao nível da qualidade do serviço (QoS) é recebido e somado ao valor da qualidade total da missão. O objetivo da missão é alcançar o maior nível de qualidade total, respeitando a restrição temporal especificada.

Os valores do nível de QoS adquiridos pela execução de cada recurso são arbitrários e fixos, sendo $Q_1=1$, $Q_2=2$, $Q_3=3$, $Q_4=4$, $Q_5=5$, $Q_6=6$, $Q_7=7$, $Q_8=8$, $Q_9=9$, $Q_{10}=10$, $Q_{11}=11$, $Q_{12}=12$, $Q_{13}=13$, $Q_{14}=14$, $Q_{15}=15$.

A qualidade do algoritmo (qualidade global do algoritmo G) é calculada através da equação:

$$G = \sum (Q\_RTA / NT) \quad (3)$$

Sendo Q_RTA a qualidade total obtida pela execução dos recursos com restrição temporal atendida e NT o número de tentativas.

5.7.2 Resultados do Experimento

Foram sorteadas 100 configurações de missões diferentes (combinações de recursos) e cada uma delas testada com 14 faixas de *deadlines* (restrições temporais) distintas. Foram lançados 100 agentes, com 100 diferentes missões e 14 deadlines diferentes, o que resultou na execução de 140.000 missões usando cada uma das heurísticas.

A Tabela 5.1 apresenta os valores do nível de Qualidade Global (usando uma unidade arbitrária criada apenas para compararmos os valores obtidos no experimento) alcançados pelos algoritmos utilizando as 100 diferentes configurações e os 14 diferentes deadlines testados. Os valores em negrito indicam as heurísticas que alcançaram os maiores índices de desempenho para cada faixa de deadline.

A Figura 5.5, baseada nos dados descritos na Tabela 5.1, apresenta o gráfico da Qualidade Global G obtida por cada Algoritmo. A amostra utilizada neste experimento envolve uma ampla variação de configurações de missões dos agentes móveis, fundamentando assim a credibilidade dos resultados apresentados.

Tabela 5.1. Valores da Qualidade Global das Heurísticas.

Abord. \ DL	150	200	250	300	350	400	450	500	550	600	650	700
Preguiçoso	0	0	0	94,55	148,8	155	155	155	155	155	155	155
Guloso	0	0	0	0	17,28	103,68	**167,04**	**186,24**	**192**	**192**	**192**	**192**
Ponderado	0	0	9,3	86,8	148,8	155	155	155	155	155	155	155
Lazy Adap.	0	0	**12,4**	**114,7**	144	135,32	153,61	170,33	**192**	**192**	**192**	**192**
PG	0	0	4,65	89,17	**151,42**	**160,12**	162,88	182,44	**192**	**192**	**192**	**192**

Analisando o gráfico da Figura 5.5, percebe-se que a Heurística Lazy-Adaptativo alcançou os mais altos índices de QoS para situações em que o deadline da missão é considerado apertado (faixa onde o algoritmo guloso não consegue alcançar nenhum resultado), cumprindo assim seu objetivo inicial. Percebe-se também que esta heurística forma a camada superior do gráfico com deadline folgados.

A partir do instante em que os deadlines de todos os agentes da missão são atendidos, o benefício global obtido pelos algoritmos permanece estável, independente do tempo restante entre o deadline e o tempo gasto com a missão.

Em relação ao gráfico da Figura 5.5, percebe-se claramente a evolução das heurísticas com relação ao desempenho. As heurísticas Preguiçosa, Ponderada e Gulosa apresentam um bom resultado apenas para faixas de deadline específicas, respectivamente, deadlines apertados, justos e folgados. A heurística PG apresentou uma melhora significativa quanto ao seu desempenho, apresentando bons resultados para mais de uma faixa de deadline, esta vantagem é consequência de sua característica de adaptação na partida. É importante lembrar que uma vez escolhido o comportamento do agente para a missão atual ele permanecerá com este até o final da missão.

A heurística Lazy-Adaptativo suporta adaptação dinâmica, ou seja, é capaz de mudar seu comportamento em tempo de execução, de forma que na mesma missão é possível que o agente móvel assuma comportamentos diferentes. Por esta razão, esta heurística mostrou uma curva de evolução mais rápida que a Heurística PG para deadlines apertados.

Outra situação interessante a ser discutida é a questão do tempo de resposta. A Figura 5.6 exibe o gráfico com os tempos médios de resposta utilizados pelas heurísticas. Observando o gráfico, nota-se que as heurísticas simples gastam o mesmo tempo de resposta para todas as faixas de deadline, sendo que no Algoritmo Guloso há um maior consumo de tempo, o que comprova que esta heurística necessita de deadlines mais folgados para obter benefícios relevantes. As demais heurísticas simples movem-se alternadamente dentro de um intervalo de tempo com valores mais baixos.

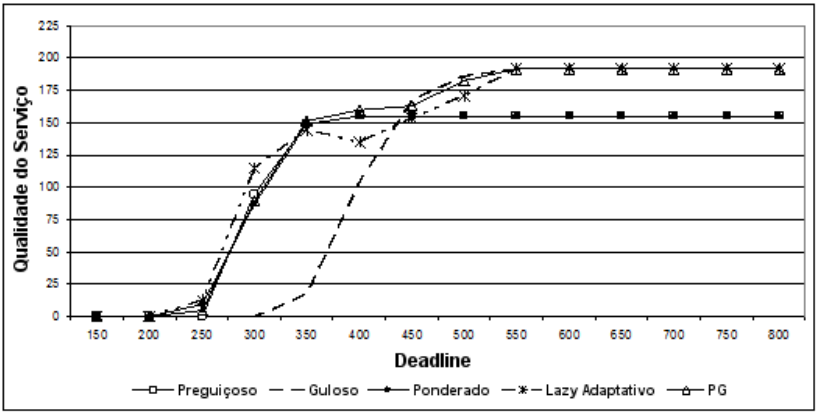

Figura 5.5. Qualidade Global das Heurísticas.

No gráfico da Figura 5.6 é possível confirmar outra vantagem no uso de heurísticas adaptativas. Para faixas de deadlines apertados estas heurísticas consomem menos tempo, percebendo uma mudança na faixa de deadline, as heurísticas com característica adaptativa procuram aproveitar essa "folga" no deadline e, buscando melhores níveis de QoS, permitem-se aumentar o consumo de tempo para concluir a missão.

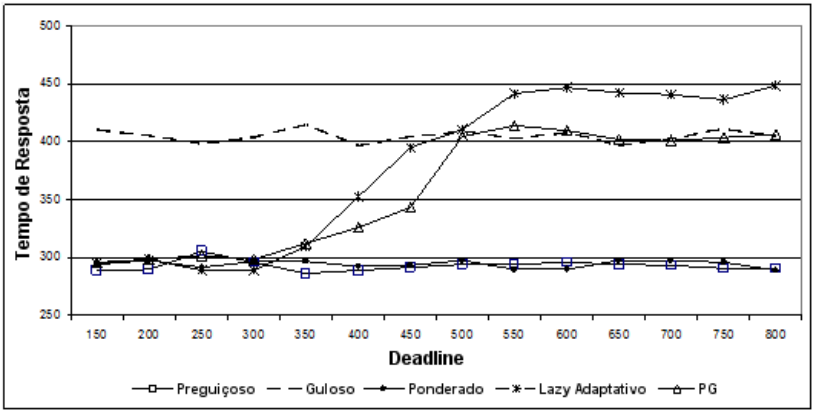

Figura 5.6. Tempo Médio de Resposta das Heurísticas.

Tabela 5.2. Valores do Tempo Médio de Resposta das Heurísticas.

Abord. \ DL	150	200	250	300	350	400	450	500	550	600	650
Preguiçoso	288,46	289,27	304,78	295,61	286,53	288,85	291,23	293,95	294,22	295,72	294,13
Guloso	410,04	405,56	398,25	403,37	414,67	396,84	404,44	408,99	402,6	407,34	396,97
Ponderado	294,17	297	291,56	296,55	296,59	292,45	293,53	297,23	289,59	290,06	297,77
Lazy Adap.	295,09	298,26	288,99	288,77	309,64	352,89	394,49	410,47	440,96	446,7	442,53
PG	295,51	298,25	302,02	298,28	312,33	325,73	343,45	404,3	413,65	409,15	401,46

A Tabela 5.2 apresenta estes valores. A Heurística Lazy-Adaptativo consome o tempo que lhe é permitido, oferecendo bom resultados desde deadlines muito apertados à deadlines mais folgados.

Baseando-se nos dados das tabelas e nos gráficos apresentados, pôde-se notar que o algoritmo Lazy-Adaptativo apresentou um desempenho eficiente para deadlines apertados (característica desejável para grande parte das aplicações tempo real), mostrando um desempenho ligeiramente superior aos algoritmos sem adaptação (o Preguiçoso, o Ponderado e o Guloso) nesses cenários. O comportamento da heurística mostrou uma característica peculiar de oscilação de resultados, causada principalmente pela limitação de memória (implementado dessa forma para atender cenários de baixa disponibilidade de recursos e melhorar o tempo de viagem pela rede dos agentes móveis).

Os agentes utilizando a Heurística Lazy-Adaptativo seguem a linha de aprendizado contínuo, visto o tamanho do histórico (armazena as 50 últimas execuções). O algoritmo descarta o histórico de execução mais antigo para guardar novos resultados, causando esse comportamento de oscilação ao se deparar com novas faixas de deadline mais folgadas.

Eventualmente, próximo aos cenários extremamente folgados, o Algoritmo Guloso (de baixo valor de dominância) mostra seu potencial. A Heurística Lazy-Adaptativo também conseguiu se aproximar rapidamente dele e obteve de forma semelhante os resultados máximos.

A Heurística Lazy-Adaptativo apresentou um comportamento positivo para deadlines apertados alcançando os melhores resultados nessa faixa. Para deadlines intermediários (faixa onde o algorítimo guloso alcança resultados positivos, mas estes ainda não são máximos), demonstrou sua capacidade de aprendizado. E por fim, para deadlines folgados, voltou a atingir os resultados com máximo nível de QoS.

5.8 CONCLUSÕES

Este capítulo teve por objetivo apresentar a nova heurística adaptativa Lazy-Adaptativo.

A nova heurística foi comparada com outras heurísticas existentes na literatura através de um experimento. Os resultados deste experimento também foram apresentados neste capítulo.

6 HEURÍSTICAS COM ADAPTAÇÃO NA PARTIDA

Nesta seção são apresentadas três novas heurísticas com adaptação na partida para determinação do itinerário de AM com o principal objetivo de atenderem aos requisitos de desempenho e restrição temporal da orquestração descentralizada.

Esta seção apresenta uma nova abordagem desenvolvida para auxiliar o AM na definição do itinerário a cada nova missão. Dentro do conceito da orquestração, cada requisição para execução do serviço composto para o orquestrador corresponde a uma missão do AM. O objetivo das heurísticas apresentadas é alcançar bons resultados para todas as faixas de deadlines, ou seja, atender diferentes faixas deadlines (folgados, justos ou apertados) mantendo um nível de QoS mínimo definido pelo cliente.

Em [Rech 2005], depois de efetuada a avaliação de desempenho, pôde-se relacionar cada tipo de deadline a uma determinada heurística. Partindo do deadline maior para o menor: o deadline folgado é definido como aquele onde o Algoritmo Guloso supera os demais; o deadline justo (intermediário) é definido como aquele onde o Algoritmo Ponderado supera os demais; e a partir deste ponto, em direção a deadlines cada vez menores, temos os deadlines apertados, onde o Algoritmo Preguiçoso obteve o melhor desempenho.

6.1 HEURÍSTICAS PROPOSTAS

São apresentadas novas heurísticas com adaptação na partida e para facilitar a compreensão das mesmas, comparativamente, será utilizado um único cenário de exemplo para demonstrar o comportamento das heurísticas em funcionamento.

O exemplo é composto de sete nodos e três tipos de recursos. Os tipos de recurso representam as diferentes tarefas que compõem os serviços da orquestração descentralizada. Os nodos N1 e N2 são do recurso tipo 1 (R=1), o nodo N3 é do recurso tipo 2 (R=2) e os nodos N4, N5 e N6 são do recurso tipo 3 (R=3).

Além disso, todos os recursos são do tipo variável, de modo que fique claro qual o nível de qualidade de serviço que está sendo obtido de cada recurso. Para simplificar os cálculos, cada célula dos recursos possui um tempo de execução fixo igual a uma unidade de tempo.

A Figura 6.1 mostra este cenário com os recursos, caminhos possíveis e a divisão em células de cada recurso variável. No início da missão, por exemplo, o AM pode executar o recurso do tipo 1 no nodo N1 ou N2. Em seguida ele deverá executar o recurso do tipo 2 no nodo N3, o único com esse tipo de recurso. Na sequência, o AM terá os nodos N4, N5 e N6 como opções para executar o recurso do tipo 3.

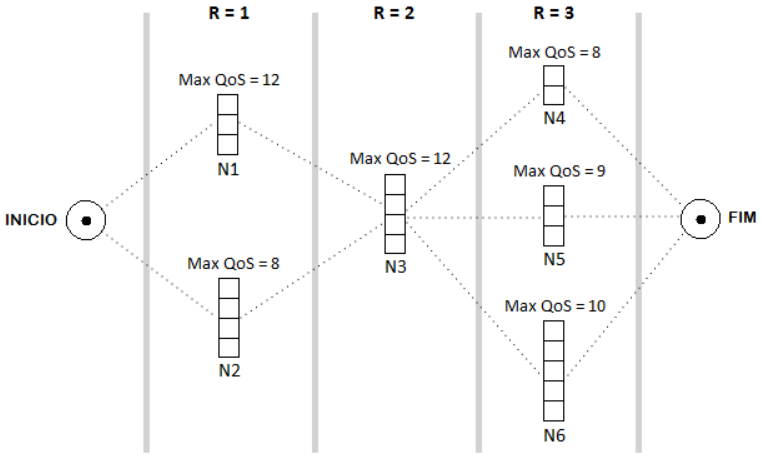

Figura 6.1. Cenário com recursos variáveis.

6.1.1 Heurística Gulosa com Limite Variável (GLVP)

O Algoritmo Guloso com Limite Variável na Partida é uma variação do Algoritmo Guloso já anteriormente adaptado para agentes móveis [Rech 2005]. Esta nova versão se baseia no conceito de *anytime algorithms* [Garvey 1994], de forma que o nível de QoS que pode ser obtido com a execução de cada serviço é variável.

A estratégia do algoritmo GLVP é iniciar com o comportamento original do algoritmo guloso [Rech 2005] e se este comportamento não se mostrar eficiente para a missão em questão, parte-se para uma redução gradual do tempo de execução máximo de cada serviço, o que refletirá no nível de QoS a ser obtido. Esta redução gradual está diretamente associada a um fator de ajuste – neste trabalho o fator de ajuste foi de 10% a cada nova viagem do agente. A heurística inicia com

um objetivo de alcançar o nível de QoS máximo (percentual de 100%) e percebendo o não cumprimento do deadline, passa por um período de ajuste, onde reduz seu percentual de execução para cada serviço até cumprir a missão com sucesso.

Por se tratar de uma missão com característica de repetição (ou seja, sabe-se que futuramente o agente móvel terá que repetir esta missão em diferentes circunstâncias) e considerando que o agente móvel é míope [Rech 2008], o objetivo desta heurística é encontrar o "percentual ótimo", que proporcionará alcançar o nível máximo de QoS, respeitando o deadline da missão atual e gastando o menor número de viagens (missões consecutivas) possíveis.

O algoritmo GLVP inicia determinando um fator de ajuste percentual que será utilizado durante todas as viagens do agente até que ele encontre o ponto de equilíbrio entre nível de QoS e restrição temporal. O objetivo é maximizar o nível de QoS sem estourar o tempo estipulado para a missão. O agente inicia as viagens com o algoritmo guloso e vai reduzindo o percentual de consumo de cada recurso da missão usando o fator de ajuste. Por exemplo, usando um fator de ajuste de 10%, em sua terceira viagem, caso ainda não tenha encontrado o ponto ótimo, o agente poderá executar apenas 80% dos recursos da missão. Isso é possível pois estamos utilizando recursos variáveis e o conceito de *anytime algorithms*.

6.1.1.1 Exemplo com a Heurística GLVP

Utilizando o mesmo cenário do exemplo anterior, é apresentado aqui o comportamento da heurística GLVP.

Na configuração da heurística é utilizado um percentual arbitrário de decréscimo da taxa de execução durante as viagens. Neste exemplo usaremos o percentual de 25%. Além disso, será utilizada uma taxa de restrição temporal de 10 unidades de tempo.

Na primeira viagem, o percentual de execução é equivalente ao esperado por uma heurística gulosa, ou seja, 100%. Com esse comportamento a heurística obtém uma pontuação global de QoS de 34 e um custo de execução de 12 unidades de tempo. A Figura 6.2 mostra o itinerário da primeira viagem.

Como o limite de tempo foi atingido, é necessária uma segunda viagem. Na segunda viagem já ocorre uma variação do percentual de execução, o qual é reduzido para 75%. Com este novo comportamento, a heurística consegue respeitar a restrição temporal e obtém uma

pontuação global de QoS de 30. A Figura 6.3 mostra o itinerário da segunda viagem.

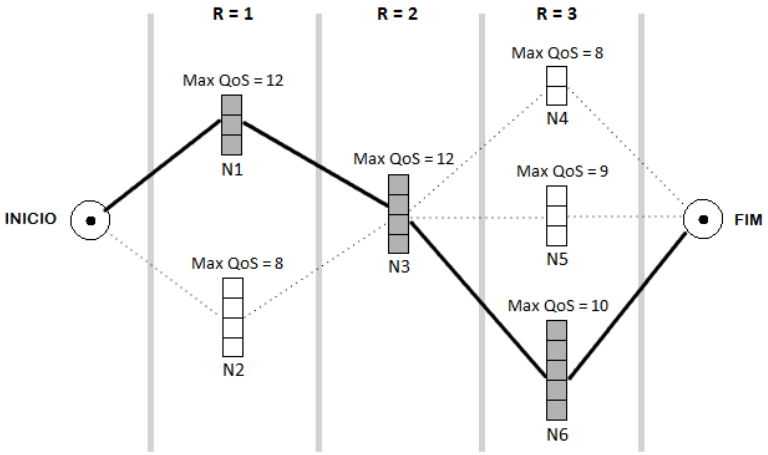

Figura 6.2. Itinerário da primeira viagem de GLVP.

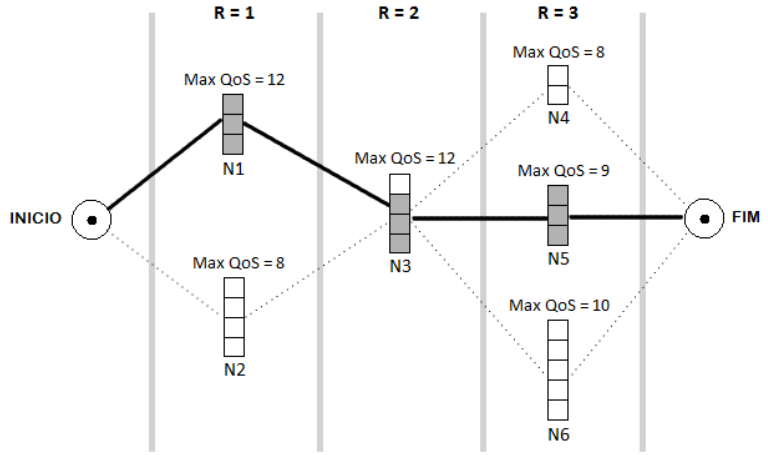

Figura 6.3. Itinerário da segunda viagem de GLVP.

Nesta segunda viagem, apesar do nodo N2 demandar a execução de apenas 75% do recurso e o nodo N1 demandar 100%, por arredondamento para cima, o consumo do recurso do tipo 1 permanece o mesmo, já que o tempo de execução de cada célula é de uma unidade de tempo. E para desempate foi determinada uma regra arbitrária onde o nodo que forneça o maior QoS será o escolhido dentre aqueles de menor tempo de execução.

No recurso de tipo 3, ocorre a troca de N6 para N5 na segunda viagem. Isso acontece, pois com o decréscimo de 25%, o nodo N6 deixa de ser o de maior QoS. Ao baixar 25%, seu benefício cai para 8 e o tempo de execução cai para 4. A heurística sempre faz o arredondamento para cima, então no caso de N6, (100-25)% fará com que sejam executadas 80% das células, daí a queda do benefício. No nodo N5, por só possuir 3 células, a queda de 25% não é suficiente para que ele deixe de executar alguma célula, com isso ele mantém seu benefício de 9. Para que ele caísse, seria preciso uma queda percentual maior que um terço, pois N5 possui 3 células.

6.1.2 Heurística Preguiçosa com Crescimento Variável (PCVP)

O Algoritmo Preguiçoso com Crescimento Variável na Partida é uma variação do Algoritmo Preguiçoso já anteriormente adaptado para agentes móveis [Rech 2005]. Assim como o Algoritmo GLVP, esta nova versão também baseia-se no conceito de *anytime algorithms* [Garvey 1994], de forma que o tempo de execução dos serviços é variável, não exigindo que o algoritmo obtenha o nível de QoS máximo de cada serviço da missão.

O Algoritmo Preguiçoso descrito em [Rech 2005] tem como principal objetivo concluir a missão no menor tempo possível, desconsiderando os níveis de QoS que serão alcançados. O principal problema detectado neste algoritmo é a folga percebida entre o tempo de resposta da missão e seu deadline na maior parte das execuções, ou seja, este tempo que "sobrou" poderia ser utilizado com o objetivo de contribuir para o refinamento/melhoria do resultado (para isso, os serviços teriam seus tempos de computação aumentados o que refletiria num maior nível de QoS adquirido).

A necessidade em se realizar mais de uma vez a mesma missão permitiu uma adaptação ao Algoritmo Preguiçoso com o objetivo de maximizar a pontuação do nível de QoS adquirido. Como já mencionado anteriormente, o nível de QoS percentual adquirido a cada

visita do agente a um nodo é proporcional ao tempo de execução que ele dispôs a cada serviço.

O Algoritmo PCVP possui o comportamento inverso do Algoritmo GLVP descrita anteriormente. Inicialmente, ela utiliza o comportamento Preguiçoso tradicional (executando o tempo de execução mínimo permitido para cada serviço que compõe a missão) e nas próximas execuções aumenta gradativamente o percentual de execução de cada serviço, considerando também um fator de ajuste pré-definido. A estratégia deste algoritmo é iniciar com o comportamento do algoritmo preguiçoso tradicional, porém busca o aumento do nível de QoS em cada missão, viagem após viagem. Após a primeira viagem da missão, a heurística aumenta o percentual do nível de QoS a ser obtido até que a missão não cumpra o deadline ou alcance o nível de QoS máximo de cada serviço. Caso a missão não cumpra o deadline em alguma execução, o algoritmo assume o penúltimo resultado como o percentual ótimo do nível de QoS a ser executado.

O algoritmo PCVP também inicia determinando um fator de ajuste percentual que será utilizado durante todas as viagens do agente até que ele encontre o ponto de equilíbrio entre nível de QoS e restrição temporal. A diferença em relação ao GLVP é que aqui o algoritmo inicia com o comportamento Preguiçoso para maximizar o nível de QoS sem estourar o tempo estipulado para a missão. Com isso, o percentual de recurso inicial é considerado 0%, sendo que na prática o algoritmo sempre executa o mínimo obrigatório de cada recurso. Um exemplo seria um recurso composto de 5 partes, sendo uma obrigatória e 4 variáveis, segundo os conceitos de *anytime algorithm*. Mesmo na primeira viagem, o agente executa 20% deste recurso, pois é obrigado a executar a parte obrigatória de cada recurso. Além disso, o agente vai aumentando o percentual de execução de todos os recursos da missão usando o fator de ajuste. Com isso, o agente levaria pelo menos 4 missões para executar 2 blocos dos cinco possíveis do recurso do exemplo, pois teria o seguinte percentual de execução nas primeiras viagens: 0%, 10%, 20% e 30%. Repare que o percentual da quarta viagem é 30% e não 40%, mas mesmo assim o agente executa o segundo bloco. Isso se deve a um fator de arredondamento utilizado pelo algoritmo para acelerar o processo de busca pelo ponto ótimo. E o algoritmo segue aumentando o fator de ajuste até encontrar o ponto ótimo.

6.1.2.1 Exemplo com a Heurística PCVP

Continuando o uso do cenário de exemplo apresentado anteriormente, é apresentado aqui um exemplo do comportamento da heurística PCVP.

Assim como no exemplo anterior, na configuração desta heurística é utilizado um percentual arbitrário de acréscimo da taxa de execução durante as viagens. Neste exemplo usaremos o percentual de 25%. Além disso, será utilizada uma taxa de restrição temporal de 10 unidades de tempo.

Na primeira viagem, o comportamento é equivalente ao esperado por uma heurística preguiçosa, ou seja, o caminho mais rápido. Como foi estabelecido que o tempo de execução de uma célula é sempre igual a uma unidade de tempo, a heurística optará pelo caminho que possuir a maior pontuação de nível de QoS como fator de desempate. Dessa forma, na primeira viagem, a heurística obtém uma pontuação global de QoS de 11 e um custo de execução de 3 unidades de tempo. A Figura 6.4 mostra o itinerário da primeira viagem. Por exemplo, se para o recurso de tipo 1, a heurística optasse no desempate para o nodo N2, ela obteria as mesmas 3 unidades de tempo de execução, mas com um QoS inferior, de apenas 9.

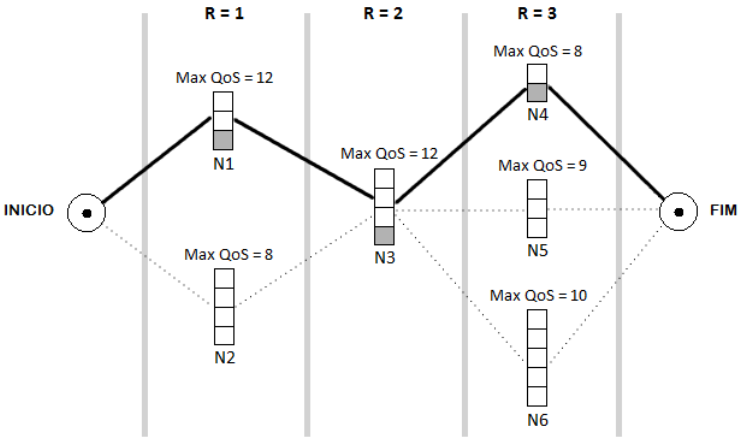

Figura 6.4. Itinerário da primeira viagem de PCVP.

Como há muito espaço para ser mais agressiva, a heurística parte para uma segunda viagem. A Figura 6.5 mostra o itinerário dessa segunda viagem. Nela ocorre uma variação do percentual de execução, o qual é aumentado para 50%. Apesar de a primeira viagem ser uma viagem com comportamento preguiçoso, é estipulado que ela possui pelo menos o percentual de execução arbitrário adotado, ou seja, 25%, fazendo com que na segunda viagem o percentual suba para um máximo de 50%. Com este novo comportamento, a heurística consegue melhorar sua pontuação global de QoS e ainda respeitar a restrição temporal. A nova pontuação global de QoS é 14 e o custo de execução sobe para 5 unidades de tempo, ou seja, ainda há espaço para continuar a melhorar. A Figura 6.5 mostra o itinerário da segunda viagem usando PCVP.

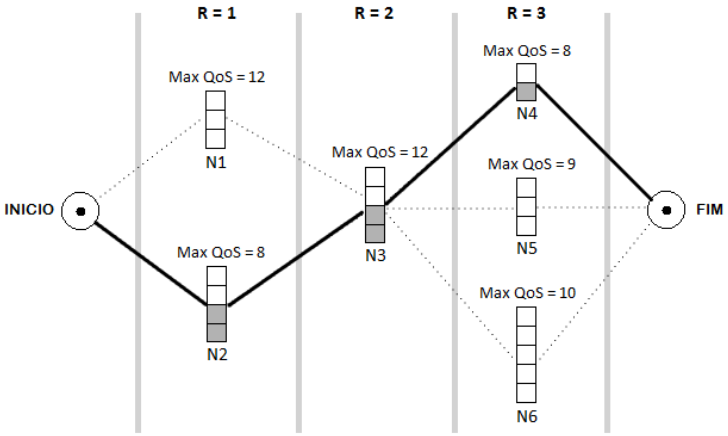

Figura 6.5. Itinerário da segunda viagem de PCVP.

Nesta segunda viagem, ocorre a escolha do nodo N2 para o recurso do tipo 1, pois diferentemente da heurística GLVP mostrada anteriormente, o arredondamento aqui é para baixo, ou seja, com um máximo de 50%, o nodo N1 continuará executando apenas uma única célula. Neste caso, o critério arbitrário de desempate para QoS iguais é o recurso com maior quantidade de partes, pois a heurística entende que será mais fácil evoluir neste nodo, já que o percentual cresce gradativamente a cada viagem.

6.1.3 Heurística Gulosa com Função de Bipartição (GFU)

O Algoritmo Guloso com Função de Utilidade baseada em Bipartição apresenta uma nova proposta para este tipo de problema. Esta nova versão baseia-se no Algoritmo Guloso de [Rech 2005], no conceito de *anytime algorithms* [Garvey 1994] e também no conceito de bipartição da teoria dos grafos [Gutin 2002].

Esta heurística, além da capacidade de executar os serviços parcialmente, utiliza uma estratégia (função de utilidade) de bipartição do nível de QoS a ser obtido, a fim de cumprir o deadline e obter o nível de QoS máximo para cada faixa de deadline.

A estratégia deste algoritmo é iniciar como um algoritmo guloso tradicional partindo em busca do ponto ótimo do nível de QoS a ser obtido com a execução (parcial ou total) de cada serviço. Para isso, a heurística GFU utiliza uma função de utilidade baseada em bipartição, de forma a acelerar a descoberta deste ponto ótimo.

Ao executar sua primeira viagem, caso a missão não tenha seu deadline cumprido, a heurística biparticiona o intervalo percentual de blocos dos serviços a serem executados. Na segunda viagem (execução), esse intervalo corresponde a (0, 100), logo a heurística buscará a execução de 50% dos blocos de cada serviço. Na terceira viagem, caso o deadline ainda não tenha sido cumprido, a heurística particiona novamente o intervalo no sentido de cumprir o deadline, logo o intervalo corresponde a (0, 50) e o novo percentual de blocos para execução é de 25%. Ainda na terceira viagem, caso o deadline da missão tenha sido obedecido, a heurística busca melhorar o nível de QoS acumulado da missão e para isso biparticiona o intervalo superior, ou seja, o intervalo correspondente a (50, 100) e assim o novo percentual de execução é 75%.

A heurística vai seguindo esta lógica de bipartição até que ela consiga cumprir o deadline da missão com uma folga de tempo inferior ao fator de folga estabelecido, um valor arbitrário atribuído com o único objetivo de evitar que a heurística continue se ajustando infinitamente, alternando as execuções entre obediência e falta ao deadline.

A diferença aqui está no fato de que a heurística GFU utiliza o tempo de execução para direcionar o sentido do ajuste a ser feito no limite percentual de blocos a serem executados, ou seja, o ajuste pode ocorrer na forma de acréscimo ou decréscimo.

6.1.3.1 Exemplo com a Heurística GFU

Mais uma vez utilizando o cenário de exemplo apresentado anteriormente, é apresentado aqui um exemplo do comportamento da heurística GFU.

Assim como nas demais heurísticas, na configuração desta é utilizado um percentual arbitrário de decréscimo da taxa de execução durante as viagens. Neste exemplo voltaremos a usar o percentual de 25%. Além disso, será utilizada uma taxa de restrição temporal de 10 unidades de tempo.

Diferentemente das outras heurísticas, a GFU pede, além do percentual de execução, um intervalo dos percentuais válidos. Na primeira viagem esse intervalo é desconsiderado.

Na primeira viagem, o percentual de execução é mais uma vez o equivalente ao esperado por uma heurística gulosa, ou seja, 100%. Com esse comportamento a heurística obtém uma pontuação global de QoS de 34 e um custo de execução de 12 unidades de tempo, que foram os mesmos valores da primeira viagem da heurística GLVP. A Figura 6.6 mostra o itinerário dessa primeira viagem.

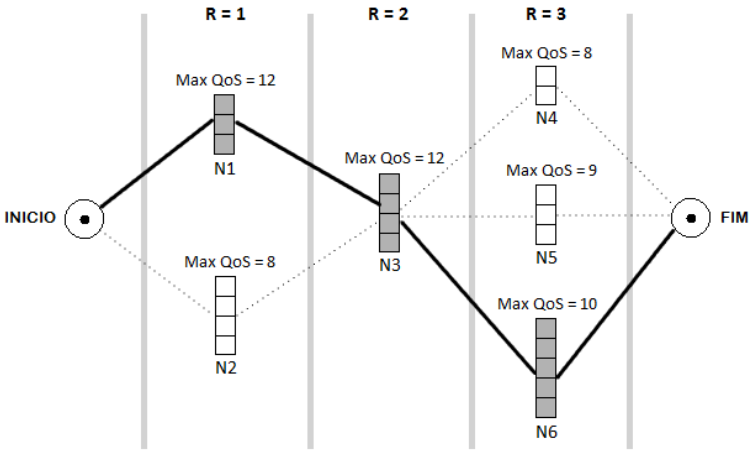

Figura 6.6. Itinerário da primeira viagem de GFU.

Como o limite de tempo foi atingido, é necessária uma segunda viagem. Na segunda viagem já ocorre uma variação do percentual de execução, mas primeiro é preciso atualizar o intervalo de percentuais válidos. Como o primeiro intervalo era nulo, na segunda viagem admite-se o intervalo máximo (0, 100). Como a GFU é uma heurística que utiliza bipartição, o percentual de execução é o ponto médio do intervalo, ou seja, no máximo 50%.

Com este novo comportamento, a heurística consegue respeitar a restrição temporal com um custo de execução igual a 6 e obtém uma pontuação global de QoS de 14. A Figura 6.7 mostra o itinerário da segunda viagem.

Nesta segunda viagem, a heurística escolhe N6 para o recurso do tipo 3 usando a mesma regra de desempate que a heurística PCVP utiliza para QoS iguais: o recurso com maior quantidade de partes é preferido.

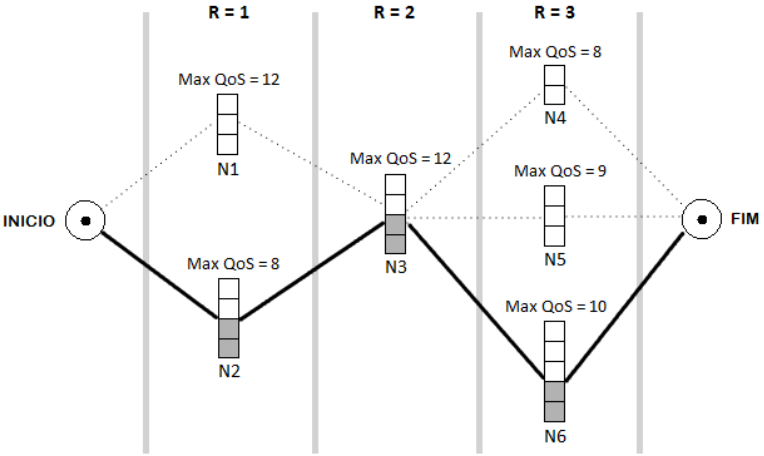

Figura 6.7. Itinerário da segunda viagem de GFU.

Como há espaço para melhoria, a heurística parte para uma nova viagem. Mais uma vez, o primeiro passo é atualizar o intervalo. Como o percentual de execução não estourou o prazo de execução, o novo intervalo utiliza o percentual da primeira viagem como limite inferior e mantém o percentual superior. Dessa forma, o novo intervalo é (50,100) e o percentual de execução é no máximo 75%.

Com este novo comportamento, a heurística ainda consegue respeitar a restrição temporal com um custo de execução igual a 8 e obtém uma pontuação global de QoS de 23. A Figura 6.8 mostra o itinerário da terceira viagem.

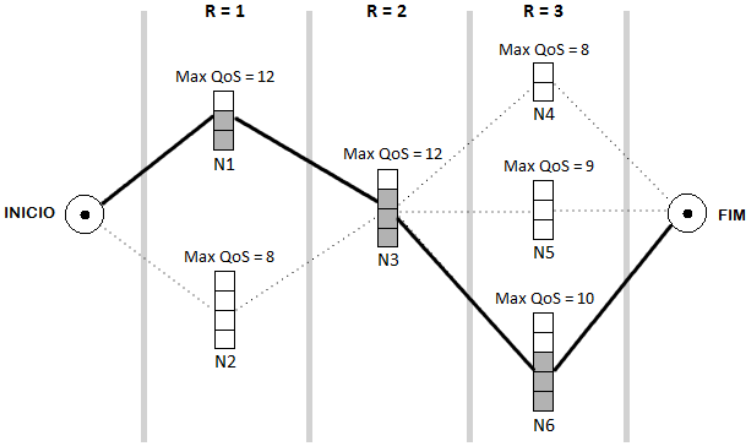

Figura 6.8. Itinerário da terceira viagem de GFU.

Para exemplificar brevemente o que aconteceria caso o deadline fosse mais apertado e a heurística não cumprisse o prazo na segunda viagem, vamos exemplificar a terceira viagem mudando a restrição temporal para 5 unidades temporais.

Neste caso há estouro do prazo na segunda viagem e é necessária uma terceira viagem. Como o percentual de execução estourou o prazo de execução, o novo intervalo utiliza o percentual da primeira viagem como limite superior e mantém o percentual inferior. Dessa forma, o novo intervalo é (0, 50) e o percentual de execução máximo é 25%.

Com este novo comportamento, a heurística consegue respeitar a restrição temporal apertada com um custo de execução igual a 3 e obtém uma pontuação global de QoS de 11. A Figura 6.9 mostra o itinerário da terceira viagem com variação da restrição temporal.

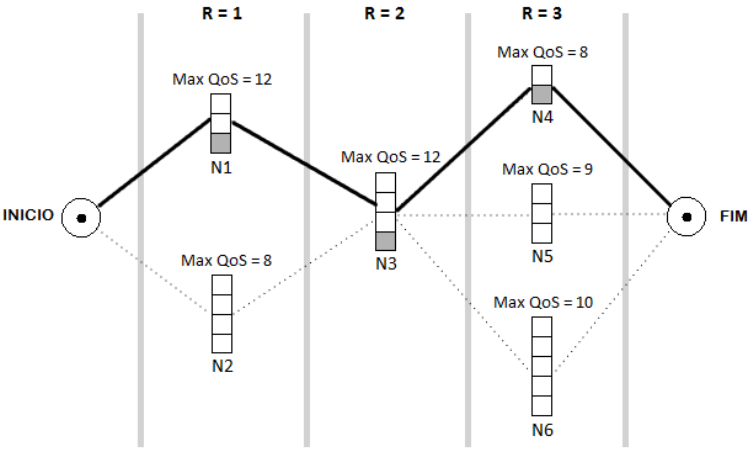

Figura 6.9. Variação na terceira viagem de GFU.

6.2 EXPERIMENTO COM RECURSOS VARIÁVEIS

Neste experimento são avaliadas as heurísticas com adaptação na partida apresentadas neste trabalho. O objetivo deste experimento é comparar o comportamento das heurísticas e avaliar sua eficiência em diferentes cenários e para diferentes faixas de deadline.

6.2.1 Condições do Experimento

Para este experimento foram considerados 18 serviços e 18 nodos. Quanto à alocação dos serviços, cada nodo hospeda um único serviço no ambiente do experimento. Os serviços são uma composição de recursos de vários tipos. A diferença de nomenclatura se deve exclusivamente ao fato de que para o experimento foram usados serviços que correspondem a um único recurso, de modo que os termos aqui são intercambiáveis.

Para que as heurísticas pudessem demonstrar sua flexibilidade e potencial, é necessário que elas possam escolher o melhor itinerário para suas missões, de modo que os 18 serviços estão divididos em apenas sete tipos (cada tipo contém um mesmo tipo de informação), ou seja, cada tipo de serviço está representado em mais de um computador da

rede. Sempre que possível colocamos cada nodo de um mesmo tipo de serviço em computadores com velocidades de processamento diferentes, a fim de melhor representar um ambiente real.

O tempo de execução de um serviço replicado (que possui o mesmo tipo de informação) em diferentes computadores também foi considerado diferente, para representar um cenário real em que diferentes máquinas possuem diferentes versões de software pertinentes ao serviço, umas mais atualizadas do que as outras.

Cada missão dos AM é composta de cinco tipos de serviços a serem obtidos, podendo haver ordem de precedência de execução ou não. Cada uma das heurísticas avalia o itinerário segundo seus algoritmos, serviço após serviço, ou avalia mais de um serviço por vez, caso não exista ordem de precedência.

Foram sorteadas 20 missões, todas compostas por cinco diferentes tipos de serviços escolhidos dentre os sete tipos de serviços disponíveis, sem repetição. A ordem de precedência de execução entre os tipos de serviço de cada missão também é variável, podendo uma missão ter grupos de tipos de serviços com e sem ordem de precedência. Este sorteio de tipos de serviços para as missões oferece milhares de arranjos possíveis.

Em cada missão são computados o tempo total da missão e o nível de QoS acumulado da missão. O tempo total da missão é composto pelo tempo de execução dos serviços, representados pelo tempo de fila do processador e tempo de execução do serviço, e pelo tempo de trânsito do AM de um nodo a outro, este associado à latência da rede no instante da viagem. O nível de QoS acumulado da missão é representado pela soma dos níveis de QoS individuais de cada serviço executado. Cada serviço é dividido em blocos de execução e cada bloco efetivamente executado representa um acréscimo no nível de QoS individual do serviço executado. Esta divisão em blocos está em acordo com a estratégia de *anytime algorithms* [Garvey 1994], de forma que o nível de QoS individual de cada serviço é um valor entre o nível de QoS máximo e mínimo do nodo em que o serviço foi executado.

A Tabela 6.1 descreve os nodos, serviços, níveis de QoS e tempos de execução utilizados na avaliação de desempenho das heurísticas.

Tabela 6.1. Descrição dos Nodos e Serviços do Segundo Experimento.

Nome do Nodo	Tipo de Serviço	QoS Máximo	QoS Mínimo	Tempo Máximo de Execução	Quantidade de Blocos
Nodo-A1	A	10	1	7.0	10
Nodo-A2	A	12	1	9.6	12
Nodo-B1	B	6	1	4.8	6
Nodo-B2	B	8	1	8.0	8
Nodo-B3	B	9	1	12.6	9
Nodo-C1	C	6	1	6.0	6
Nodo-C2	C	10	1	11.0	10
Nodo-D1	D	10	1	8.0	10
Nodo-D2	D	10	1	5.0	10
Nodo-E1	E	8	1	4.8	8
Nodo-E2	E	10	1	8.0	10
Nodo-E3	E	12	1	12.0	12
Nodo-F1	F	10	1	5.0	10
Nodo-F2	F	14	1	11.2	14
Nodo-G1	G	8	1	7.2	8
Nodo-G2	G	9	1	6.3	9
Nodo-G3	G	10	1	10.0	10
Nodo-G4	G	12	1	13.2	12

6.2.2 Resultados do Experimento

Para ilustrar o desempenho das heurísticas na bateria de testes realizados, são consideradas três diferentes missões (M1, M2 e M3) para três diferentes faixas de deadlines. Cada missão é composta de cinco tipos de serviço dentre os sete disponíveis. A missão também estabelece se deve ou não haver precedência entre a execução de cada tipo de

serviço. O objetivo dos testes com diferentes itinerários e diferentes deadlines é verificar se o comportamento das heurísticas permanece similar mesmo quando os AM executam diferentes missões. Como o AM não leva consigo um histórico de execuções anteriores faz-se necessário um período de ajuste (calibragem) até que se descubra o percentual do tempo de execução de cada serviço que compõe a missão, o que refletirá no alcance do maior nível de QoS possível para determinada missão e respectivo deadline.

As Tabelas 6.2, 6.3 e 6.4 apresentam os desempenhos das três heurísticas adaptativas propostas considerando deadlines apertados para as missões sorteadas M1, M2 e M3. Para facilitar a compreensão dos resultados encontrados, foi escolhido um único índice para representar esta faixa de deadline (D=15). Nas tabelas, NP representa o número de passos utilizados (cada passo representa uma viagem completa do AM), as colunas T e Q representam, respectivamente, o tempo de resposta e o nível de QoS adquiridos ao final da execução de cada missão, utilizando diferentes heurística.

Tabela 6.2. Desempenho das Heurísticas para M1

NP	M1					
	GLVP		PCVP		GFU	
	T	Q	T	Q	T	Q
1	58,4	55	58,4	55	3,8	5
2	54,4	51	29,1	28	7,6	10
3	49	46	16,2	15	9,6	13
4	40,7	39	10,8	10	13,4	18
5	29,9	28	13,7	13	14,8	20
6	24,1	23	14,8	14	18,6	25
7	19,1	18	14,8	14	**14,8**	**20**
8	13,7	13	14,8	14	14,8	20

As células com sombreamento apontam o momento em que os melhores índices são obtidos por cada heurística com sua respectiva

missão. Porém, são os valores em negrito na tabela que indicam os melhores desempenhos.

Por exemplo, para a primeira missão M1, o algoritmo que alcançou o melhor nível de QoS, cumprindo o deadline, foi o Algoritmo Guloso com Função de Utilidade (GFU). Para chegar a este índice foram necessárias sete viagens (NP) prévias utilizando diferentes percentuais de execução, conforme apresentado na Tabela 6.2.

Analisando quanto ao requisito de número de passos (número de viagens completas do agente), o algoritmo PCVP mostrou-se mais rápido para a missão M1, necessitando um menor número para descobrir o percentual ideal de cálculo para o tempo de computação de cada serviço.

Tabela 6.3. Desempenho das Heurísticas para M2

NP	M2					
	GLVP		PCVP		GFU	
	T	Q	T	Q	T	Q
1	54,4	53	54,4	53	3,7	5
2	47,9	50	27,9	27	7,4	10
3	43	44	15,6	15	9,3	13
4	38,1	39	10,4	10	13	18
5	33,2	34	12,3	12	**14,9**	**21**
6	25,7	27	14,2	14	14,9	21
7	21,5	22	14,2	14	14,9	21
8	15,6	17	14,2	14	14,9	21
9	11,7	12	14,2	14	14,9	21

Para as missões M2 e M3, a heurística que apresentou os melhores resultados foi a GFU, encontrando o melhor nível de QoS em um menor número de passos. Por ter seu comportamento inicial inspirado no Algoritmo Guloso, esperava-se que esta heurística encontrasse o percentual ideal mais rapidamente que as demais

heurísticas quando o deadline da missão fosse apertado. Os resultados dos testes comprovam esta expectativa.

Conforme esperado, a heurística GLVP, baseada no comportamento Guloso tradicional, não alcançou os melhores índices para esta faixa de deadline, necessitando um número maior de passos para ajustar o nível de QoS e atingir o ponto ótimo.

Tabela 6.4. Desempenho das Heurísticas para M3

NP	M3					
	GLVP		PCVP		GFU	
	T	Q	T	Q	T	Q
1	52,4	55	52,4	55	3,8	5
2	48,9	51	26,9	28	7,6	10
3	44	46	15,5	16	9,6	13
4	38,3	40	9,8	10	**13,4**	**18**
5	33,4	35	11,4	12	13,4	18
6	26,9	28	14,1	15	13,4	18
7	22	23	14,1	15	13,4	18
8	17,1	18	14,1	15	13,4	18
9	11,4	12	14,1	15	13,4	18

É importante lembrar que, com o objetivo de facilitar a compreensão dos resultados, nas Tabelas 6.2, 6.3 e 6.4 apenas ilustramos os resultados para deadline igual a 15, pois valores de deadline próximos apresentaram resultados similares.

As Figuras de 6.10 e 6.11 apresentam gráficos que contribuem para o entendimento do comportamento de cada heurística adaptativa. Observando essas Figuras, percebe-se que mesmo para diferentes missões, o comportamento das heurísticas permanece similar. A Figura 6.10 ilustra o comportamento para a primeira missão M1 (formada conforme descrito na seção anterior). A Figura 6.11 representa os índices alcançados pelas heurísticas adaptativas para a missão M2.

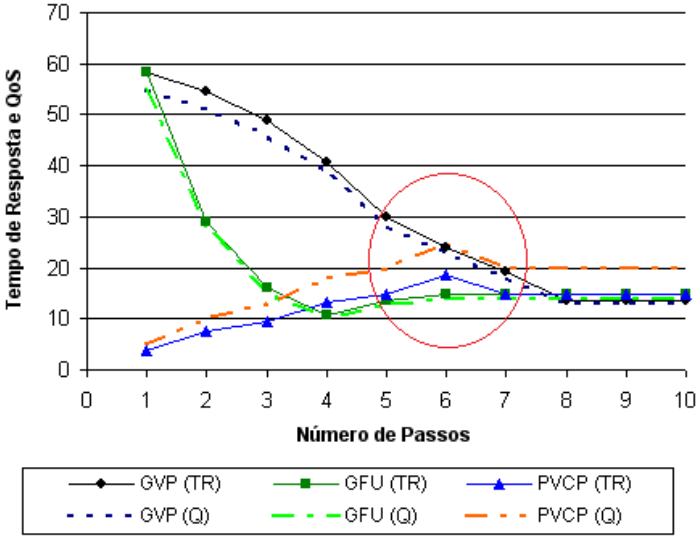

Figura 6.10. Desempenho das heurísticas considerando o Deadline 15 para a missão M1

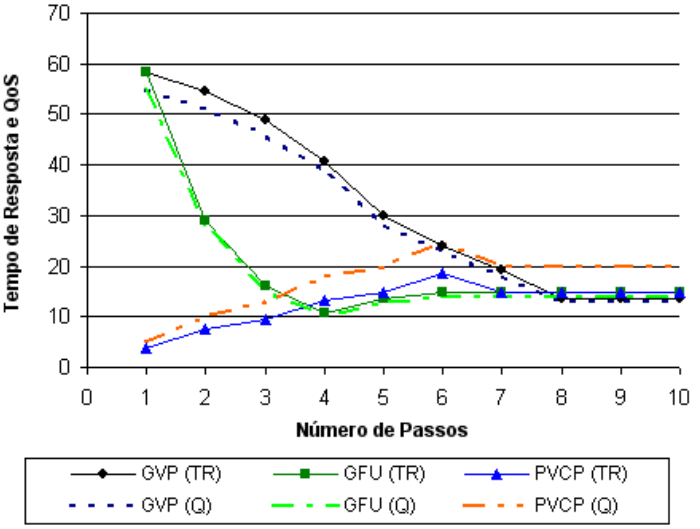

Figura 6.11. Desempenho das heurísticas considerando o Deadline 15 para a missão M2

Observando o eixo das ordenadas percebe-se que a Heurística GFU é a primeira a encontrar um tempo de resposta adequadamente próximo ao deadline da missão (passo seis). A elipse evidencia no gráfico o ponto ótimo encontrado por cada uma das heurísticas. É perceptível que após o ponto ótimo, os tempos de computação e níveis de QoS permanecem os mesmos para as futuras viagens dessas missões.

Também foi verificado o comportamento das heurísticas para deadlines justos e folgados. Analisando as Figuras 6.12 e 6.13, pode-se facilmente perceber o desempenho das heurísticas. Elas apresentam, respectivamente, os tempos de resposta obtidos considerando deadlines justos (índice escolhido para representação = 30) e deadlines folgados (D=45) para a Missão M3.

A heurística que exigiu menor número de passos para encontrar o ponto ótimo considerando missões com deadlines folgados foi a GLVP, esta rapidez na calibragem deve-se ao fato de que inicialmente esta heurística utiliza o comportamento tradicional do Algoritmo Guloso e sua variação é gradual. Nos testes, utilizando deadline justo (D=30), as heurísticas GLVP e GFU empataram no quesito número de passos – reflexo do comportamento adotado no ponto de partida.

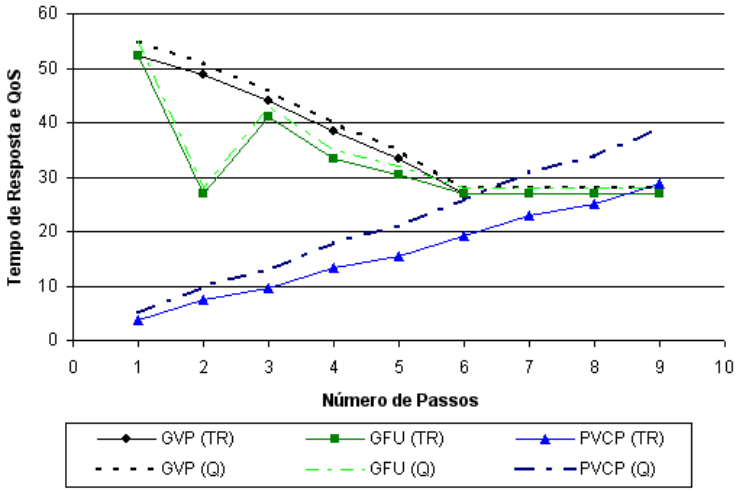

Figura 6.12. Desempenho das heurísticas considerando o Deadline Justo 30

Quanto ao nível de QoS alcançado, para deadlines folgados, GLVP superou as demais heurísticas. Como esperado, para estas faixas de deadline, a heurística PVCP exigiu um número bem maior de passos para atingir o "ponto ótimo", mas compensou este excedente no período de calibragem quando alcançou o melhor nível de QoS para D=30 (o mesmo não aconteceu com D=45).

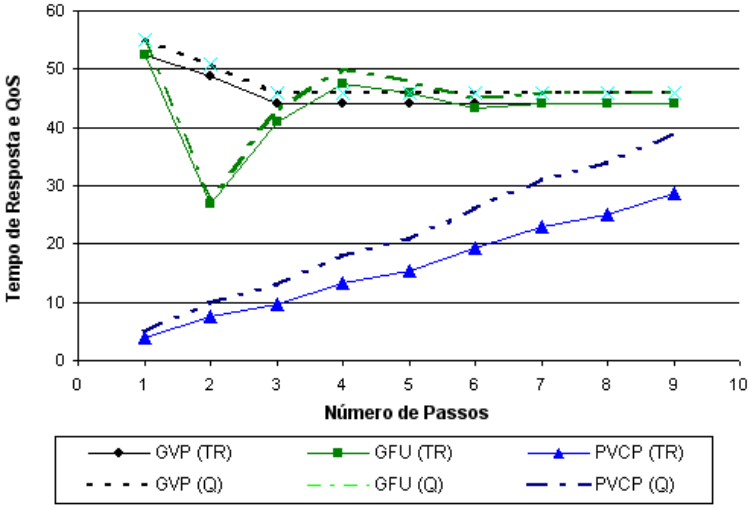

Figura 6.13. Desempenho das heurísticas considerando o Deadline Folgado 45

7 CONCLUSÃO

O objetivo central deste trabalho foi desenvolver novas heurísticas para determinação do itinerário de agentes móveis em missões com restrição temporal e análisá-las em experimentos que simulam cenários reais.

7.1 SOBRE HEURÍSTICAS ADAPTATIVAS

No contexto de aplicações distribuídas baseadas em agentes móveis, é possível encontrar agentes móveis que devam cumprir um deadline tendo um grau de flexibilidade na definição do itinerário, ou seja, precisam otimizar o nível de qualidade do serviço da missão.

No Capítulo 5 deste trabalho foi proposta e avaliada uma nova heurística adaptativa de definição de itinerário, a heurística Lazy-Adaptativo. Uma minuciosa análise comparativa realizada no experimento deste Capítulo permitiu confirmar o bom desempenho desta heurística. A adaptação no decorrer do percurso trouxe maior robustez ao algoritmo. A heurística utiliza um histórico de execuções anteriores para mudar o comportamento sugerido na partida da missão, conforme perceba alterações no ambiente.

Todas as heurísticas propostas neste trabalho oferecem um baixo custo computacional, algo necessário, pois alguns nodos a serem visitados pelo agente móvel podem possuir baixa capacidade de processamento.

O comportamento da heurística proposta foi analisado em uma arquitetura baseada nos padrões FIPA/IEEE e os resultados foram comparados com heurísticas existentes na literatura com o intuito de se eleger a heurística que melhor se adaptasse a diferentes situações em um sistema distribuído. Todas as heurísticas respeitam a premissa da miopia dos agentes móveis.

O fato das heurísticas simples existentes na literatura apresentarem um comportamento definido para cada faixa de deadline tornou possível sugerir o comportamento mais adequado para cada nova missão, uma vez que o deadline desta nova missão fosse conhecido. Portanto, levando-se em conta apenas o deadline da missão atual e a experiência adquirida nas missões anteriores, para cada nova missão foi possível escolher o comportamento mais adequado para aquela situação.

Os resultados obtidos com a execução do experimento usando abordagens adaptativas mostraram desempenho bastante satisfatório, fazendo com que fossem alcançados os melhores índices de benefício dentro das condições de cada missão. A possibilidade de mudar o comportamento em tempo de execução aliado à flexibilidade permitida pelas características de alguns dos recursos que formam a missão proporcionou bons resultados.

O objetivo do uso das abordagens adaptativas é criar uma heurística capaz de ajustar-se conforme a situação do sistema e as necessidades impostas pela missão a ser executada. Portanto, elas não têm por meta superar as heurísticas originais em todas as faixas de deadlines e sim manter bons resultados (próximos ao melhor resultado obtido por cada heurística em cada faixa de deadline) para todas as faixas de deadline. Este objetivo foi alcançado com a heurística Lazy-Adaptativo.

7.2 SOBRE ADAPTAÇÃO NA PARTIDA

Dentre os sistemas existentes em um ambiente corporativo, os sistemas de missão crítica (aqueles que não podem falhar, mas admitem flexibilidade no tempo de resposta) e os sistemas de tempo real (aqueles em que o tempo de resposta necessita ser previsível), em geral, podem sofrer com arquiteturas centralizadoras como a da orquestração de serviços. Neste contexto, várias soluções para descentralizar a orquestração já foram apresentadas pela literatura. No Capítulo 5 foi proposta uma modelagem utilizando agentes móveis para criar uma orquestração descentralizada.

Aproveitando a arquitetura padrão da orquestração descentralizada apresentada em diversos outros trabalhos, no Capítulo 6 apresentamos três novas heurísticas de definição de itinerário dos agentes móveis para auxiliar na orquestração. Realizamos uma análise comparativa entre essas heurísticas que nos permitiu confirmar o bom desempenho das mesmas para resolver problemas de desempenho da orquestração. São heurísticas que utilizam um fator de ajuste para mudar o nível de QoS a ser executado em cada missão e assim atender ao deadline esperado, ou seja, atender às expectativas de desempenho do cliente.

Todas as heurísticas aqui discutidas oferecem um baixo custo computacional, condição necessária para demonstrar a viabilidade dessa arquitetura de orquestração descentralizada utilizando agentes móveis,

pois uma vez que não é o objetivo deste trabalho detalhar esta nova arquitetura, precisamos pressupor que alguns nodos a serem visitados pelo agente móvel podem possuir baixa capacidade de processamento.

Os resultados da análise de desempenho das heurísticas foram comparados entre si com o intuito de se eleger a heurística que melhor se adaptasse a diferentes situações na orquestração descentralizada. A Heurística Gulosa com Função de Bipartição (GFU) foi a que apresentou as melhores respostas em diferentes cenários, às vezes com vantagem no tempo de resposta, as vezes na qualidade do serviço, mas na maioria dos casos era vantajosa em ambos os aspectos. Todas as heurísticas respeitam a premissa da miopia dos agentes móveis, ou seja, o agente não detém o conhecimento da lógica do fluxo dos serviços e depende de uma arquitetura híbrida entre a orquestração centralizada, para criar serviços compostos, e a orquestração descentralizada, onde os serviços detêm conhecimento do fluxo de serviços. O objetivo de se conseguir uma abordagem flexível, onde o agente móvel é capaz de adaptar seu comportamento para atender a diferentes níveis de QoS e cumprir os deadlines esperados, foi alcançado.

7.3 PRINCIPAIS CONTRIBUIÇÕES DO TRABALHO

Dentre as principais contribuições desta dissertação estão uma nova heurística adaptativa (Lazy-Adaptativo) para ser utilizadas por agentes móveis na determinação de itinerário de missões com deadline muito apertados e também 3 novas heurísticas capazes de se adaptarem em cenários com deadline fixos e assim atingir o ponto ótimo de QoS, sem estourar o prazo de sua restrição temporal.

Ao longo do desenvolvimento desta dissertação, foram publicados os seguintes artigos:

MAGALHÃES, Alex; Rech, Luciana; Lung, Lau; Oliveira, Rômulo (2009). **A Heurística Lazy Adaptativa para Determinar Itinerários de Agentes Móveis Imprecisos**, SBRC 2009, XXVII Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos. UFPE, Recife, Brasil.

MAGALHÃES, Alex; Rech, Luciana; Lung, Lau; Oliveira, Rômulo (2010). **Using Intelligent Mobile Agents to Dynamically Determine Itineraries with Time Constraints**, ETFA 2010, 15th IEEE

International Conference on Emerging Technologies and Factory Automation. Bilbao, Espanha.

MAGALHÃES, Alex; Rech, Luciana; Lung, Lau (2010). **Decentralized Services Orchestration Using Intelligent Mobile Agents with Deadline Restrictions**, AIAI 2010, 6th IFIP Conference on Artificial Intelligence Applications & Inovations. Larnaca, Chipre.

7.4 TRABALHOS FUTUROS

Após a conclusão desta dissertação, algumas possibilidades de continuidade do trabalho são apresentadas:

- Adaptar a plataforma JADE para suportar tempo real nativamente;
- Desenvolver heurísticas para a busca contínua de melhorias em deadlines folgados;
- Estudo de novos cenários de aplicação combinando com agentes inteligentes.

REFERÊNCIAS

Baek, J.W., Kim, G.T. and Yeom, H.Y. (2001). **Timed Mobile Agent Planning for Distributed Information Retrieval**. Computation and Engineering School, Seoul National University, Proceedings of AGENTS'01, Montreal, Canada.

Binder, W.; Constantinescu, I.; Faltings, B. (2006). **Decentralized Orchestration of Composite Web Services**. Proceedings of the IEEE International Conference on Web Services, pp.869-876.

Barker, A.; Weissman, J.; Hemert, J. (2008). **Eliminating The Middleman: Peer-to-Peer Dataflow**. Proceedings of the 17th international symposium on High performance distributed computing, pp.55-64.

Barker, A.; Weissman, J.; Hemert, J. (2009). **The Circulate architecture: avoiding workflow bottlenecks caused by centralized orchestration**. Cluster Computing, Vol. 12, Issue 2, pp.221-235.

Barker, A.; Weissman, J.; Hemert, J. (2009b). **The benefits of service choreography for data-intensive computing**. High Performance Distributed Computing. Proceedings of the 7th international workshop on Challenges of large applications in distributed environments. Garching, Germany.

Bellifemine F., Poggi A., Rimassa G. (2001). **Developing Multi-agent Systems with JADE**, Intelligent Agents VII Agent Theories Architectures and Languages, Ed. Springer.

FIPA Architecture Technical Committee (2002). **FIPA Abstract Architecture Specification**. Padrão FIPA, disponível em: <http://www.fipa.org/>.

FIPA Communication Technical Committee (2002b). **FIPA ACL Message Structure Specification**. Padrão FIPA, disponível em: <http://www.fipa.org/>.

FIPA Nomadic Application Technical Committee (2002c). **FIPA Quality of Service Ontology Specification**. Padrão FIPA, disponível em: <http://www.fipa.org/>

FIPA (2010). **The Foundation for Intelligent Physical Agents**. Disponível em: <http://www.fipa.org/>

Fokus Institute & IBM Corporation (1997). **Mobile Agent System Interoperability Facilities Specification**. Especificação conjunta padronizada pela OMG, disponível em: <http://www.omg.org/>.

Frolund, S. and Koistinen, J. (1998). **Quality-of-service specification in Distributed object systems**. Distributed System Engineering 5, pag. 179–202.

Garvey, A. and Lesser V. (1994). **A survey of research in deliberative real-time artificial intelligence**. The Journal of Real-Time Systems.

Golden, B. (1976). **Shortest-path algorithms - a comparison**. Operations Research, Vol. 24, No.9, pp. 1164-1168.

Gutin, G., and Punnen, A., P. (2002). **The Traveling Salesman Problem and Its Variations**, Kluwer Academic Publishers.

Ilarri, S., Mena, E., Illarramendi, A. (2008). **Using cooperative mobile agents to monitor distributed and dynamic environments**, Information Sciences: an International Journal, Volume 178, Issue 9.

Jacob, J.C., Katz, D.S., and et. al. (2004**). The Montage Architecture for Grid-Enabled Science Processing of Large, Distributed Datasets**. In Proceedings of the Earth Science Technology Conference, June 2004.

Kanoh, H., Hara, K. (2008). **Hybrid genetic algorithm for dynamic multi-objective route planning with predicted traffic in a real-world road network**, Proceedings of the 10th annual conference on Genetic and evolutionary computation, Atlanta, GA, USA.

Koenig, S. and Likhachev, M. (2006). **Real-time adaptive A***, International Conference on Autonomous Agents, pag 281-288. Japão.

Kyprianou, N. (2008). **Hybrid Web Service Orchestration**. MSc Thesis, The University of Edinburgh.

Liu, J. W. S.; Shih, W. K.; Lin, K. J. et al. (1994) **Imprecise Computations**. Proceedings of the IEEE. Volume 82. n°1. pp. 83-94. Janeiro. 1994.

Martins, P. and Silva, N. (2002). **Agentes Móveis: Redes Inteligentes e Aplicações**. ISEP – Instituto Superior de Engenharia do Porto. Departamento de Engenharia Eletrônica. Portugal, 2002.

OASIS UDDI Specification Technical Committee (2002). **UDDI Version 2.04 API Specification**. Padrão Oasis. Disponível em: <http://www.oasis-open.org/specs/>

Papazoglou, M. P. and Georgakopoulos, D. (2003). **Introduction: Service-oriented computing**. Communications of the ACM, 46(10): pag 24–28.

Qu, W.; Shen, H.; Jin, Y. (2005). **Theoretical Analysis on A Traffic-Based Routing Algorithm of Mobile Agents**. Proceedings of IEEE/WIC/ACM International Conference on Intelligent Agent Technology IAT. pp.520-526.

Rech, L., Oliveira, R. and Montez, C. (2005). **Dynamic Determination of the Itinerary of Mobile Agents with Timing Constraints**. IAT 2005 IEEE/WIC/ACM Int. Conf. on Intelligent Agent Technology. Compiègne. França. pag 45-50.

Rech, L., de Oliveira, R. and Montez, C. (2006). **A Clone-Pair for the Dynamic Determination of the Itinerary of Imprecise Mobile Agents with Firm Deadlines**, ETFA 2006 11th IEEE Int. Conf. on Emerging Technologies and Factory Automation, Praga, República Tcheca.

Rech, L., de Oliveira, R., Montez, C. et. al. (2008). **Determination of the Itinerary of Imprecise Mobile Agents using an Adaptive Approach**, ETFA 2008 13th IEEE International Conference on Emerging Technologies and Factory Automation, Hamburg, Alemanha.

Rech, L., de Oliveira, R. and Montez, C. (2008a). **Itinerary Determination of Imprecise Mobile Agents with Firm Deadlines**,

Web Intelligence and Agent Systems, An International Journal, Volume 6, n° 4, pages 421-439.

Russel, S., and Norvig, P. (1995). **Artificial Intelligence: A Modern Approach**. Prentice Hall Series in Artificial Intelligence. Prentice-Hall, Inc.

Silva, E.L., and Menezes, E. M. (2001). **Metodologia da pesquisa e elaboração de dissertação**. 3. ed. revisada. Laboratório de Ensino a Distância da UFSC, Florianópolis.

Stankovich, J.A. (1992). **Real Time Computing**. BYTE, Volume 17, Issue 8, August 1992. McGraw-Hill, Inc.

Tanenbaum, A.; Woodhull, A. (1997). **Operating Systems: Design and Implementation**. Prentice-Hall, Inc.

Zgaya, H., Hammadi, S., Ghédira, K., (2008). **A migration strategy of mobile agents for the transport network applications**, Mathematics and Computers in Simulation, Volume 76, Issue 5-6.