Ministério da
Ciência e Tecnologia

INPE-15729-TDI/1475

UMA ARQUITETURA MULTI-AGENTE DE BALANCEAMENTO DE CARGA PARA APLICAÇÕES DE OBJETOS DISTRIBUÍDOS

Andreia Carniello

Tese de Doutorado do Curso de Pós-Graduação em Computação Aplicada, orientada pelos Drs. Maurício Gonçalves Vieira Ferreira e José Demísio Simões da Silva, aprovada em 19 de fevereiro de 2009.

Registro do documento original:
<http://urlib.net/sid.inpe.br/mtc-m18@80/2009/04.20.17.15>

INPE
São José dos Campos
2009

PUBLICADO POR:

Instituto Nacional de Pesquisas Espaciais - INPE
Gabinete do Diretor (GB)
Serviço de Informação e Documentação (SID)
Caixa Postal 515 - CEP 12.245-970
São José dos Campos - SP - Brasil
Tel.:(012) 3945-6911/6923
Fax: (012) 3945-6919
E-mail: pubtc@sid.inpe.br

Ministério da
Ciência e Tecnologia

INPE-15729-TDI/1475

UMA ARQUITETURA MULTI-AGENTE DE BALANCEAMENTO DE CARGA PARA APLICAÇÕES DE OBJETOS DISTRIBUÍDOS

Andreia Carniello

Tese de Doutorado do Curso de Pós-Graduação em Computação Aplicada, orientada pelos Drs. Maurício Gonçalves Vieira Ferreira e José Demísio Simões da Silva, aprovada em 19 de fevereiro de 2009.

Registro do documento original:
<http://urlib.net/sid.inpe.br/mtc-m18@80/2009/04.20.17.15>

INPE
São José dos Campos
2009

Dados Internacionais de Catalogação na Publicação (CIP)

Carniello, Andreia.
C217a uma arquitetura multi-agente de balanceamento de carga para aplicações de objetos distribuídos / Andreia Carniello. – São José dos Campos : INPE, 2009.
176p. ; (INPE-15729-TDI/1475)

Tese (Computação Aplicada) – Instituto Nacional de Pesquisas Espaciais, São José dos Campos, 2009.
Orientadores : Drs. Maurício Gonçalves Vieira Ferreira e José Demísio Simões da Silva.

1. Sistemas distribuídos. 2. Sistemas multi-agentes. 3. Tecnologia de objetos distribuídos. 4. Balanceamento de carga. 5. Redes neurais artificiais. I.Título.

CDU 004.75:004.541.44

Aprovado (a) pela Banca Examinadora em cumprimento ao requisito exigido para obtenção do Título de Doutor(a) em

Computação Aplicada

Dr. José Carlos Becceneri

Presidente / INPE / SJCampos - SP

Dr. José Demisio Simões da Silva

Orientador(a) / INPE / SJCampos - SP

Dr. Mauricio Gonçalves Vieira Ferreira

Orientador(a) / INPE / SJCampos - SP

Dr. Lamartine Nogueira Frutuoso Guimarães

Membro da Banca / IEAv/CTA / SJCampos - SP

Dra. Ana Paula Gonçalves Serra

Convidado(a) / USJT / São Paulo - SP

Dr. Edgar Toshiro Yano

Convidado(a) / ITA / São José dos Campos - SP

Aluno (a): Andreia Carniello

São José dos Campos, 19 de fevereiro de 2009

A meus pais e avós,
que eu tanto amo.

AGRADECIMENTOS

Agradeço a Deus e aos Céus pela proteção e por orientar meus caminhos e pensamentos.

Aos meus orientadores prof. Mauricio G. V. Ferreira e prof. José Demisio S. da Silva agradeço pelo profissionalismo e orientação. Obrigada pela dedicação e, sobretudo, pela amizade.

A meus pais, a meus avós, a minhas irmãs, a minha sobrinha e a meu cunhado gostaria de agradecer pelo incentivo e carinho.

Agradeço aos amigos do INPE pelas contribuições ao trabalho e pela amizade. Agradeço a meus familiares e aos amigos do IFSP pela amizade.

Gostaria de agradecer à CAPES e ao IFSP pelo apoio financeiro.

RESUMO

Diante da necessidade de prover equilíbrio de carga aos sistemas distribuídos, este trabalho de pesquisa propõe um serviço de balanceamento de carga que atua em aplicações de objetos distribuídos denominado arquitetura MABal – arquitetura *M*ulti-*A*gente Distribuída de *Bal*anceamento de Carga para Aplicações de Objetos Distribuídos. A arquitetura MABal realiza o balanceamento de carga de um sistema com base nos níveis de carga dos nós e, diferentemente de outras abordagens de balanceamento de carga, preocupa-se com o tráfego de rede gerado durante a comunicação entre os objetos distribuídos de uma aplicação para a realização de um caso de uso, ou seja, para o provimento de uma determinada funcionalidade da aplicação ao usuário. Esta arquitetura é constituída por um grupo de agentes, modelados segundo a metodologia de Engenharia de Software Orientada a Agentes MESSAGE: (i) agentes gerenciadores; (ii) um agente neural, que utiliza uma rede neural Perceptron de Múltiplas Camadas como mecanismo de raciocínio; e (iii) agentes de balanceamento de carga, que realizam migrações e replicações de objetos servidores. Estes agentes atuam de forma cooperativa, a cada invocação de serviço de um objeto, para selecionar o nó mais apropriado para executar o objeto provedor do serviço requisitado. Esta seleção do nó mais apropriado é guiada por um conjunto de políticas que visam oferecer uma solução de balanceamento de carga orientada a caso de uso aos sistemas distribuídos. A arquitetura MABal teve o seu comportamento simulado pela ferramenta de simulação SimBal, desenvolvida neste trabalho de pesquisa. Os resultados desta simulação foram comparados aos resultados da simulação da execução de um mesmo caso de uso aplicando-se o Serviço de Nomes da especificação CORBA. Os resultados desta comparação mostraram-se favoráveis ao serviço de balanceamento de carga da arquitetura MABal, pois esta arquitetura apresentou menor tempo de execução e menor quantidade de acessos à rede durante a execução do caso de uso considerado.

A MULTI-AGENT LOAD BALANCING ARCHITECTURE FOR DISTRIBUTED OBJECT APPLICATIONS

ABSTRACT

Considering the need for balancing distributed systems, we propose a load balancing service that operates on distributed object applications named MABal. MABal architecture analyses the load levels of nodes and, differently from other load balancing approaches, considers the network transfer during the communication of distributed objects at a use case execution. This architecture is formed by a group of agents that are modeled by MESSAGE methodology: (i) coordinator agents; (ii) a neural agent that has a Multiple Layer Perceptron neural net as its inference mechanism; and (iii) load balancing agents that migrate and replicate server objects. This group of agents acts cooperatively to select the most appropriate node for executing the required service every time an object service is required. This selection is guided by a set of policies aiming at offering a use-case based load balancing solution to distributed systems. MABal architecture behavior has been simulated by a simulation tool named SimBal, developed by us. The simulation results have been compared to CORBA Naming Service ones for the same use case execution. The comparison results showed that MABal architecture obtained a lower execution time and also a lower number of network acesses during the use case execution.

SUMÁRIO

Pág.

LISTA DE FIGURAS

Pág.

LISTA DE TABELAS

Pág.

1 INTRODUÇÃO

1.1 Motivação

Uma aplicação de objetos distribuídos consiste em objetos que residem em nós diferentes de um sistema e se comunicam de forma remota e transparente para executarem suas funcionalidades. A palavra sistema refere-se ao conjunto de nós de uma rede de computadores.

As solicitações de serviço chegam aos objetos distribuídos de uma aplicação de forma aleatória. Isso pode gerar uma situação de balanceamento não-uniforme entre os nós do sistema, fazendo com que alguns nós apresentem uma alta carga de processamento enquanto outros fiquem levemente carregados ou totalmente ociosos. Esta situação é prejudicial em termos do tempo de resposta dos serviços oferecidos pelos objetos da aplicação e também em termos da utilização dos recursos computacionais. Assim, verifica-se a necessidade de oferecer uma solução de equilíbrio de carga aos sistemas distribuídos a fim de evitar prejuízos às aplicações e a estes sistemas.

Na literatura, existem abordagens propostas para solucionar o problema de balanceamento de carga em sistemas distribuídos, como as abordagens propostas por El-Abd e El-Bendary (1998), Schlemer (2002), Yang et al. (2003) e Suri et al. (2007). No entanto, identificou-se que estas abordagens não levam em consideração o tráfego de rede gerado pela comunicação entre os objetos de uma aplicação para efetuar as realocações de objetos no sistema com o objetivo de promover o balanceamento de carga.

Diferentemente destas abordagens de balanceamento de carga, este trabalho de pesquisa propõe uma arquitetura, chamada de arquitetura MABal – arquitetura *M*ulti-*A*gente Distribuída de *Bal*anceamento de Carga para Aplicações de Objetos Distribuídos, que além de realizar o balanceamento de

carga de um sistema com base nos níveis de carga dos nós, preocupa-se com o tráfego de rede gerado durante a comunicação entre os objetos distribuídos de um caso de uso. Um caso de uso representa um conjunto de objetos de uma aplicação que interagem entre si para o provimento de uma determinada funcionalidade da aplicação ao usuário.

Esta preocupação é pertinente, pois um serviço de balanceamento de carga pode otimizar o tempo de resposta do caso de uso requisitado pelo usuário por meio da redução da quantidade de acessos à rede para a execução do caso de uso. Assim, este trabalho de pesquisa apresenta um diferencial devido ao fato do serviço de balanceamento de carga ser orientado a caso de uso.

A idéia de prover um serviço de balanceamento de carga orientado a caso de uso consiste em buscar realocar objetos de nós sobrecarregados para nós que, além de apresentarem níveis de carga adequados, estabeleçam alguma comunicação com os objetos realocados. Em resumo, uma vez que se faz necessário realocar objetos para promover o balanceamento de carga do sistema, a idéia consiste em fazer com que objetos que se comunicam entre si fiquem fisicamente localizados no mesmo nó para reduzir acessos à rede nas invocações atuais e futuras.

1.2 Objetivos

Em um sistema distribuído, as solicitações de serviços chegam aos nós do sistema de forma aleatória, o que pode causar um desbalanceamento de carga no sistema, ou seja, pode haver sobrecarga em alguns nós enquanto outros se mantêm ociosos. Esta situação de desbalanceamento de carga pode comprometer os tempos de resposta dos serviços requisitados pelos objetos de uma aplicação, assim como comprometer a utilização dos recursos do sistema.

Em decorrência destes problemas, o objetivo principal da abordagem proposta neste trabalho de pesquisa consiste em evitar desbalanceamento de carga entre os nós de um sistema. Portanto, propõe-se uma arquitetura de software que atua, em aplicações distribuídas, para promover o balanceamento de carga por meio da realocação dos objetos da aplicação localizados em nós sobrecarregados para nós ociosos.

O segundo objetivo da arquitetura de balanceamento de carga proposta consiste em conduzir o balanceamento de carga do sistema de modo que a nova realocação dos objetos da aplicação, além de promover o balanceamento de carga do sistema, reduza a quantidade de acessos à rede, efetuados pelos objetos, durante a comunicação entre eles. Assim, a arquitetura proposta sugere que a realocação dos objetos da aplicação nos nós do sistema seja orientada a caso de uso, em prol do balanceamento de carga do sistema em solicitações de serviço atuais e futuras.

Conforme o mecanismo de balanceamento de carga orientado a caso de uso proposto neste trabalho, diante de uma situação de desbalanceamento de carga em um sistema, devem-se realocar os objetos para nós ociosos do sistema que possuam objetos que colaboram entre si para a realização de um mesmo caso de uso. O objetivo da realocação de objetos orientada a caso de uso consiste em induzir os objetos pertencentes a um mesmo caso de uso a estarem fisicamente localizados em um mesmo nó do sistema, que apresente nível de carga adequado, a fim de promover o balanceamento de carga do sistema e também reduzir a quantidade de acessos à rede durante a comunicação entre os objetos.

Outros objetivos deste trabalho de pesquisa consistem em conferir o seguinte conjunto de características à arquitetura de software de balanceamento de carga proposta:

- modelagem bem definida do problema de balanceamento de carga, por meio da: (i) utilização do conceito de agentes como elemento principal do projeto da arquitetura de software proposta; e (ii) aplicação de uma metodologia de Engenharia de Software Orientada a Agentes;
- atuação de forma preventiva, por meio da ativação do serviço de balanceamento de carga por demanda, ou seja, por meio da ativação do serviço de balanceamento de carga a cada invocação de serviço de um objeto da aplicação (invocação entre objeto cliente e objeto servidor). Esta forma de ativação do serviço de balanceamento de carga visa garantir que cada solicitação de serviço de um objeto seja atendida em um nó com estado de carga adequado, evitando sobrecargas ao sistema por parte da aplicação distribuída considerada;
- suporte à tomada de decisão, por meio da aplicação da técnica de Redes Neurais Artificiais para a classificação das cargas dos nós do sistema. A classificação em níveis de carga, gerada por esta técnica, define restrições que devem ser levadas em consideração durante o processo de balanceamento de carga;
- aplicabilidade em diferentes contextos, por meio da capacidade de promover o balanceamento de carga de sistemas heterogêneos que executem aplicações de objetos distribuídos quaisquer;
- tolerância a falhas, por meio da descentralização do processo de controle do serviço de balanceamento de carga;
- flexibilidade, por meio da replicação de objetos que provêem os serviços requisitados a fim de que as réplicas destes objetos passem a atender os serviços requisitados em nós ociosos, em caso de sobrecarga nos nós onde se localizam os objetos originais.

Por fim, além dos objetivos descritos anteriormente, este trabalho de pesquisa se propõe a: (i) implementar um protótipo que simule o funcionamento da arquitetura de balanceamento de carga proposta; (ii) desenvolver um estudo de caso aplicando a arquitetura proposta; e (iii) comparar a arquitetura proposta

com o serviço de nomes da especificação CORBA (OMG, 2008). A especificação CORBA consiste em um modelo de computação distribuída padrão que é independente de linguagem de programação. A escolha da especificação CORBA como parâmetro de comparação deve-se ao fato de que este modelo possui uma maior quantidade de características de apoio ao desenvolvimento de aplicações em ambientes distribuídos se comparado a outros modelos de suporte a aplicações distribuídas, como os modelos Java RMI (SUN, 2009) e DCOM (MICROSOFT, 2009).

1.3 Estrutura do Trabalho

No Capítulo 2 são introduzidos conceitos sobre sistemas distribuídos e descreve-se a estrutura da especificação CORBA. Este capítulo também apresenta as características do problema de balanceamento de carga em sistemas distribuídos e uma análise comparativa entre as abordagens de balanceamento de carga existentes na literatura para estes sistemas, destacando-se o aspecto diferencial deste trabalho de pesquisa.

O Capítulo 3 é dedicado à técnica de Redes Neurais Artificiais, em especial à descrição de uma rede neural de arquitetura Perceptron de Múltiplas Camadas. Esta arquitetura de redes neurais é utilizada pelo serviço de balanceamento de carga proposto neste trabalho de pesquisa para a classificação de dados essenciais à tomada de decisão do processo de balanceamento de carga.

O Capítulo 4 descreve o conceito de agentes e os motivos pelos quais se concebeu a arquitetura de balanceamento de carga proposta utilizando este conceito. Apresentam-se também as características gerais de um sistema multi-agente.

O Capítulo 5 está voltado para a área de Engenharia de Software Orientada a Agentes. Neste capítulo descreve-se um conjunto de *frameworks* de avaliação

de metodologias orientadas a agentes existentes na literatura e justifica-se a utilização de um destes *frameworks* para identificar uma metodologia adequada à modelagem da arquitetura de balanceamento de carga proposta neste trabalho. Em seqüência, este capítulo apresenta as características da metodologia MESSAGE, a metodologia orientada a agentes escolhida.

O Capítulo 6 apresenta a arquitetura multi-agente, proposta neste trabalho de pesquisa, que atua em aplicações de objetos distribuídos com o objetivo de promover balanceamento de carga ao sistema. Neste capítulo apresentam-se a estrutura e a organização de agentes da arquitetura proposta, e os modelos orientados a agentes da arquitetura.

O Capítulo 7 apresenta um estudo de caso que simula o comportamento da arquitetura proposta aplicando-a a um software simulador de satélites distribuído. Para realizar esta simulação foi desenvolvida uma ferramenta de simulação de balanceamento de carga denominada ferramenta SimBal. Este capítulo apresenta também uma simulação do comportamento da abordagem de balanceamento de carga da especificação CORBA para o mesmo cenário de execução e realiza uma análise comparativa entre as duas abordagens.

O Capítulo 8 apresenta as conclusões, incluindo as contribuições deste trabalho e os trabalhos futuros vislumbrados.

2 SISTEMAS DISTRIBUÍDOS E BALANCEAMENTO DE CARGA

2.1 Definição e Características de Sistemas Distribuídos

A definição de sistemas distribuídos é considerada consensual. Os pesquisadores envolvidos no estudo e desenvolvimento desta área de pesquisa apresentam definições com palavras próprias de forma a ressaltar um conjunto específico de propriedades de um sistema distribuído. A seguir, apresentam-se algumas destas definições.

“Um Sistema Distribuído é uma coleção de computadores independentes que parecem ao usuário ser um computador único” (TANENBAUM, 2002).

Um sistema distribuído consiste em uma coleção de computadores autônomos ligados por uma rede de computadores. Esta rede de computadores é equipada com um sistema de software distribuído, que confere as seguintes habilidades aos computadores: coordenar atividades, compartilhar os recursos (sejam eles *hardware*, *software* ou dados) e oferecer aos usuários a transparência de perceber somente um computador, embora seja uma rede de computadores que podem estar em diferentes localidades (TANENBAUM, 2002).

Coulouris et al. (2007) definem um sistema distribuído como sendo um sistema no qual os componentes de *hardware* e de *software*, localizados em computadores interligados por uma rede, se comunicam e coordenam suas ações através de troca de mensagens.

Tanenbaum (2002) descreve um conjunto de características dos sistemas distribuídos, conforme apresentado na Tabela 2.1.

Tabela 2.1 – Características de Sistemas Distribuídos

Item	**Descrição**
Relação Custo Benefício	Os microprocessadores oferecem uma melhor relação preço/desempenho do que a oferecida pelos *mainframes*.
Desempenho	Um sistema distribuído pode ter um poder de processamento maior que o de um *mainframe*.
Confiabilidade	Se uma máquina sair do ar, o sistema como um todo pode sobreviver.
Crescimento Incremental	O poder computacional pode ser expandido gradualmente, conforme a necessidade.
Compartilhamento	Permite que mais de um usuário acesse uma base de dados comum ou periféricos.
Flexibilidade	Espalha a carga de trabalho por todas as máquinas disponíveis na rede.
Restrições	Até o presente momento não há muita disponibilidade de *software* para os sistemas distribuídos. Pode-se ter problemas de excesso de tráfego na rede ou problemas com a segurança dos dados.

Fonte : Adaptada de Tanenbaum (2002)

Bernstein (1996) acrescenta a característica de heterogeneidade ao conjunto de características descrito na Tabela 2.1. Portanto, Bernstein (1996) ressalta a característica de heterogeneidade de um sistema distribuído, que significa que este tipo de sistema pode interagir com diferentes redes (protocolos) de computadores, com diferentes tipos de *hardware* computacional, sistemas operacionais diversos e diferentes linguagens de programação. Segundo Bernstein (1996), um sistema distribuído é capaz de integrar diferentes

tecnologias por meio de um consenso na interoperabilidade entre estas tecnologias.

Ferreira (2001) realiza uma análise interessante sobre sistemas distribuídos que evidencia conjuntos de vantagens e de desvantagens da utilização deste tipo de sistema. A Seção 2.2, a seguir, descreve esta análise.

2.2 Vantagens e Desvantagens dos Sistemas Distribuídos

Segundo Ferreira (2001), as vantagens de se distribuir o processamento consistem na(o):

- capacidade de distribuir a carga de uma determinada aplicação em computadores melhores adaptados para executar as tarefas;
- melhor aproveitamento dos recursos computacionais existentes;
- possibilidade de utilizar recursos de *hardware* e de *software* disponíveis em plataformas heterogêneas e;
- sistema ser visualizado como um todo ao invés de ser visto como uma coleção de componentes independentes (característica de transparência dos sistemas distribuídos). Dessa forma, objetos locais e remotos podem ser acessados usando operações idênticas, sem a necessidade de conhecimento de suas localizações.

Apesar das vantagens dos sistemas distribuídos, apresentadas por Ferreira (2001), existem algumas dificuldades. Segundo Ferreira (2001), desenvolver aplicações distribuídas pode ser uma tarefa difícil pelos seguintes motivos:

- Falta de experiência: a falta de experiência da equipe em projeto, implementação, teste e uso de *softwares* distribuídos. Além da dificuldade em testar e depurar aplicações distribuídas ser maior em relação a sistemas *stand-alone*;
- Questões de segurança: deve existir preocupação com a proteção dos canais de comunicação. Algumas políticas de segurança devem ser

planejadas e implementadas para aplicações distribuídas a fim de proteger e evitar acessos ilegais às informações;

- Uso de múltiplas tecnologias: uma aplicação distribuída pode requerer o uso de múltiplas tecnologias. Estas tecnologias podem ser de fabricantes diferentes e podem não ser compatíveis;
- Múltiplos modos de falha: aplicações distribuídas podem falhar em modos diferentes, pois são constituídas por um maior número de componentes em relação a uma aplicação *stand-alone*. Por exemplo, uma aplicação distribuída com dois componentes A e B, executando em máquinas diferentes, pode falhar de três modos: (i) uma falha no componente A; (ii) uma falha no componente B; ou (iii) uma falha na rede que conecta A e B.

2.3 Middleware

A característica de interoperabilidade de um sistema distribuído é implementada por uma solução conhecida como *middleware*, que é um *software* de conectividade que oferece um conjunto de serviços que permite a interação, através da rede, de múltiplos processos executando em uma ou mais máquinas (BERNSTEIN, 1996).

O *middleware* mascara a heterogeneidade da rede, do sistema operacional ou das linguagens de programação usadas e esconde do programador da aplicação toda a complexidade associada à passagem de valores e sincronização entre sistemas heterogêneos (FERREIRA, 2001).

Segundo Ferreira (2001), a adição de um *middleware* entre um cliente e um servidor gera um conjunto de benefícios, tais como:

- Redução da complexidade, uma vez que o *middleware* provê os serviços comuns de troca de mensagens entre os objetos da aplicação distribuída;
- Transparência de localização;

- Isolamento dos objetos da aplicação de sua implementação, que pode estar: escrita em outra linguagem de programação e residente em outra plataforma;
- O cliente e o servidor não precisam se conhecer diretamente.

Alguns exemplos de *middleware são*: arquitetura CORBA – *Commom Object Request Broker Architecture* (OMG, 2008), JAVA RMI – JAVA *Remote Method Invocation* (SUN, 2009) e DCOM – *Distributed Component Object Model* (MICROSOFT, 2009). Neste trabalho de pesquisa descreve-se a arquitetura CORBA devido às seguintes características deste *middleware*: consiste em um modelo de computação distribuída padrão, independente de linguagem de programação, e possui a maior quantidade de características de apoio ao desenvolvimento de aplicações em um ambiente distribuído. A Seção 2.4 descreve este *middleware* em detalhes.

2.4 Arquitetura CORBA

A arquitetura CORBA – *Common Object Request Broker Architecture* (OMG, 2008) define uma especificação que permite aos objetos de sistemas distribuídos comunicarem entre si de forma transparente, não importando onde se localizam fisicamente, em que plataforma ou sistema operacional rodam, em que linguagem de programação eles foram implementados e até mesmo qual protocolo de comunicação eles utilizam.

A arquitetura CORBA 1.1 foi inicialmente implementada em 1991 como sendo um produto intelectual do *Object Management Group* (OMG), um consórcio de mais de 800 companhias de diferentes áreas (IBM, Canon, DEC, Philips, Sun, Apple e etc., assim como grandes usuários como Citicorp, British Telecom, American Airlines e outros) interessadas em prover uma estrutura comum para

o desenvolvimento, independente de aplicações, usando técnicas de orientação a objeto em redes de computadores heterogêneas.

A arquitetura CORBA, que é um modelo baseado em objetos, permite que métodos de objetos sejam ativados remotamente através de um elemento intermediário chamado ORB (*Object Request Broker*). O ORB situa-se entre o objeto propriamente dito e o sistema operacional.

Em dezembro de 1994, a OMG introduziu à arquitetura CORBA o *Internet Inter-ORB Protocol* (IIOP). Antes do IIOP, a arquitetura CORBA definia somente interação entre objetos distribuídos criados pelo mesmo fornecedor. Os objetos tinham que ser desenvolvidos por uma implementação específica. Usando-se o IIOP, a arquitetura CORBA torna-se de fato uma solução para a interoperabilidade entre objetos.

A arquitetura CORBA representa uma parte de um grande quadro chamado *Object Management Architecture* (OMA). Seus principais componentes são:

- Núcleo CORBA e ORB – manipulam requisições entre objetos;
- Serviços CORBA – definem serviços em nível de sistema que ajudam a gerenciar e manter objetos;
- Facilidades CORBA – definem facilidades e interfaces em nível de aplicação: manipulação de dados e armazenamento;
- Objetos de Aplicação - são os objetos propriamente ditos em nível visível de aplicação.

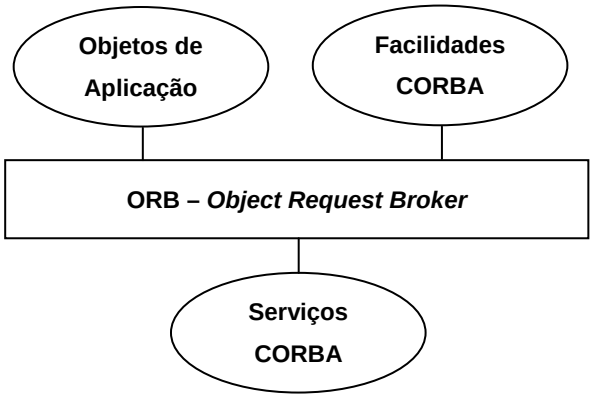


Figura 2.1 – Componentes da OMA
Fonte: Adaptada de OMG (2008)

O ORB é o componente mais importante da arquitetura OMA. Ele permite que objetos façam e recebam requisições de métodos transparentemente em um ambiente distribuído e heterogêneo.

2.4.1 Object Request Broker

O ORB consiste na estrutura mais importante da arquitetura CORBA (OMG, 2008), sendo responsável por intermediar todas as transferências entre cliente e servidor, e fazer com que a transação seja transparente para cada uma das partes durante todo o processo.

A Figura 2.2 ilustra uma requisição oriunda de um cliente sendo enviada por intermédio do ORB a uma implementação de objeto. O ORB é responsável: (i) pela localização do objeto ao qual se destina a requisição; (ii) pelo envio dos parâmetros da requisição no formato aceito por este objeto; e (iii) pelo retorno de parâmetros de saída da requisição para o cliente, se houver.

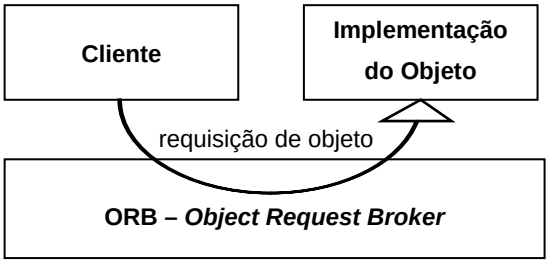


Figura 2.2 – Requisição enviada através do ORB
Fonte: Adaptada de OMG (2008)

A Figura 2.3 apresenta a estrutura de um ORB e as suas interfaces, através das quais ocorre a interação com os clientes e com as implementações de objetos.

Os objetos clientes fazem requisições de serviços de objetos através de um ORB. Considera-se um cliente qualquer objeto capaz de solicitar um serviço, sendo a implementação do objeto a resposta para esta solicitação. O ORB é responsável por todos os mecanismos necessários para:

- Localizar a implementação de objeto de uma requisição de serviço;
- Preparar a implementação de objeto para atender a requisição;
- Realizar toda a comunicação.

Este processo é totalmente transparente ao cliente, não importando onde o objeto está localizado, em qual linguagem de programação foi implementado ou qualquer outro aspecto no que diz respeito à interface do objeto e/ou a sua implementação.

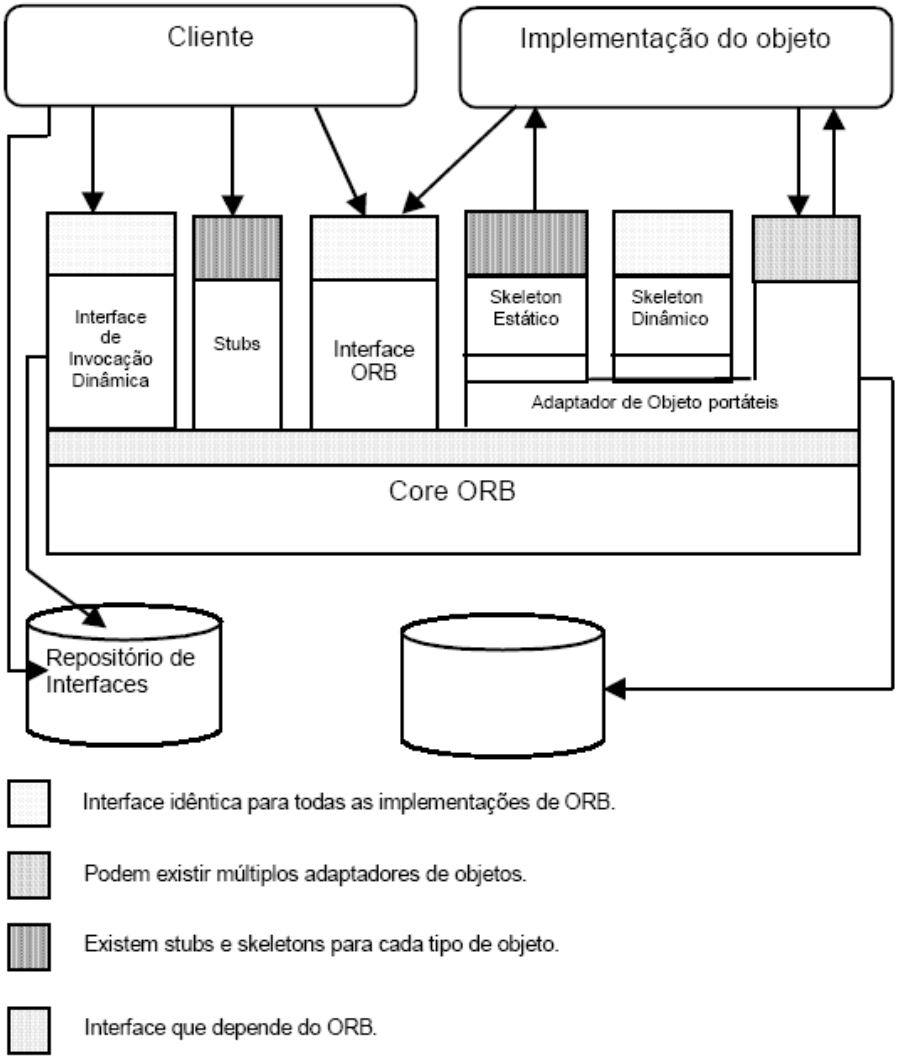


Figura 2.3 – Estrutura de um ORB

Fonte: Retirada de Ferreira (2001)

Para realizar uma requisição de serviço, o cliente pode efetuar a invocação do objeto servidor, ou seja, do objeto que provê o serviço requisitado, de dois modos: (i) através de uma rotina *stub*, que depende do objeto servidor; ou (ii) através de uma Interface de Invocação Dinâmica, que é independente da interface do objeto servidor.

A rotina *stub* é gerada na compilação da descrição de interface do objeto servidor, ao qual se destina a requisição de serviço. A interface do objeto

servidor deve ser escrita em uma linguagem chamada *Interface Definition Language* (IDL). A IDL é uma linguagem puramente declarativa, que permite declarar os nomes dos métodos do objeto servidor e os seus respectivos parâmetros. Portanto, especificando-se as interfaces dos objetos de uma aplicação na linguagem IDL torna-se possível a um cliente obter as informações necessárias para a requisição de serviço de um objeto servidor, independente de qual linguagem de programação o cliente e o objeto servidor tiverem sido escritos. A linguagem IDL representa uma linguagem de definição que ambos os objetos possuem em comum para serem capazes de se comunicar.

Alternativamente, pode-se utilizar uma Interface de Invocação Dinâmica (*Dynamic Invocation Interface* – DII) para realizar a invocação do objeto servidor. Para tanto, faz-se necessário a inclusão da interface do objeto servidor em um depósito de interfaces chamado de Repositório de Interfaces (*Interface Repository*). O Repositório de Interfaces armazena as interfaces dos objetos registrados, sendo que neste Repositório os componentes das interfaces passam a ser considerados objetos, permitindo que estes componentes sejam acessados em tempo de execução.

Assim, para a requisição de um serviço o cliente pode utilizar rotinas *stubs* correspondentes ou formular uma requisição dinâmica através da DII. O método utilizado na requisição é transparente à implementação do objeto servidor, uma vez que os fatores que os diferenciam são tratados internamente pelo ORB.

Uma vez efetuada a requisição de serviço, o ORB localiza o código da implementação do objeto servidor, transmite os parâmetros e transfere o controle para esta implementação do objeto através de um Esqueleto Estático (IDL *Skeleton*) ou de um Esqueleto Dinâmico (*Dynamic Skeleton*), específico à Interface e ao Adaptador de Objeto.

Na realidade podem existir vários objetos servidores, ou seja, vários objetos que provêem o serviço requisitado pelo cliente. Em decorrência disto, quando um objeto cliente envia uma requisição para o ORB, este seleciona a implementação que satisfaz a requisição e utiliza o Adaptador de Objeto (*Object Adapter*) para direcionar a requisição para a implementação correspondente. O Adaptador de Objetos utiliza os Esqueletos (*Skeletons*) para chamar os métodos na implementação (FERREIRA, 2001).

Ferreira (2001) descreve em seu trabalho de pesquisa definições, apresentadas a seguir, para os componentes: Adaptador de Objeto, Esqueletos Estáticos e Esqueletos Dinâmicos.

Um Adaptador de Objeto é responsável por gerar e interpretar referências de objetos, invocar métodos, ativar e desativar objetos, mapear as referências dos objetos às suas correspondentes implementações e registrar implementações.

Esqueletos Estáticos são responsáveis por prover uma conexão entre os Adaptadores de Objeto e os métodos que executam as operações em um objeto. São similares aos *stubs* do cliente, sendo gerados, automaticamente, no momento da compilação do arquivo que contém a descrição em IDL de cada método do objeto.

Quando os objetos não possuem os correspondentes Esqueletos Estáticos, os Esqueletos Dinâmicos se tornam responsáveis por oferecer um mecanismo de acesso a estes objetos. O Esqueleto Dinâmico inspeciona os valores dos parâmetros da mensagem recebida para determinar o objeto destinatário e o correspondente método, podendo utilizar o Repositório de Interfaces para obter informações.

O núcleo ORB oferece os mecanismos de representação de objetos e de comunicação para que se torne possível o processamento e a execução das requisições.

2.4.2 Serviços CORBA

O ORB (OMG, 2008) fornece os mecanismos básicos necessários para os objetos interoperarem. No entanto, outros serviços são necessários para prover a interoperabilidade entre objetos. Assim, a arquitetura OMA, além de ser constituída pelo ORB, oferece um conjunto de serviços que são realizados por objetos com interface IDL, que são padronizados pela OMG e disponibilizados para os objetos de aplicação poderem utilizar. Desta forma, os serviços CORBA garantem a consistência das aplicações e auxiliam no aumento da produtividade da atividade de programação distribuída.

Os serviços CORBA existentes são: Serviço de Nomes, Serviço de Eventos, Serviço de Ciclo de Vida, Serviço de Transação, Serviço de Relacionamento, Serviço de Negócio, Serviço de Segurança, Serviço de Controle de Concorrência, Serviço de Tempo, Serviço de Persistência de Objetos, Serviço de Externalização, Serviço de Comando, Serviço de Licença, Serviço de Propriedade, Serviço de Coleção de Objetos e Serviço de Negociação de Objetos.

Neste trabalho de pesquisa descreve-se, a seguir, apenas o Serviço de Nomes devido a este Serviço ser um dos principais Serviços CORBA e constituir o Serviço para o qual a arquitetura de balanceamento de carga proposta sugere uma extensão.

2.4.2.1 Serviço de Nomes

Conforme apresentado por Ferreira (2001), o Serviço de Nomes (*Name Service*) é um serviço genérico utilizado para localizar um objeto, isto é, se o cliente possui o nome do objeto, ele pode, através deste serviço, recuperar a referência ou o endereço de memória deste objeto.

As operações chaves do serviço de nomeação são de “ligar” (*bind*) e “resolver” (*resolve*). A operação de “ligar” é utilizada para adicionar um nome de objeto e sua referência no diretório do Serviço. Na operação de “resolver”, o cliente entra com um nome de objeto particular e recebe a referência ou endereço deste objeto como valor de retorno ou uma exceção, caso o nome não se encontre no diretório.

A associação nome/objeto é chamada nome de ligação (*binding name*). O nome de ligação é sempre definido em relação a um nome de contexto (*context name*), que é um objeto contendo um conjunto de nomes de ligação. “Resolver” um nome é determinar o objeto associado a esse nome, em um dado contexto, referenciado por um nome de contexto. “Ligar” um nome é criar um nome de ligação neste contexto. O nome é sempre resolvido em relação a um contexto; não existem nomes absolutos.

Uma questão importante em sistemas distribuídos é o gerenciamento de carga dos nós da rede. É interessante que haja um equilíbrio de carga entre os nós para que não haja gargalos no sistema. Uma boa política para balancear a carga do sistema é fundamental para se tirar o melhor proveito possível das capacidades computacionais da rede, para se alocar nós mais adequados a execução de determinadas tarefas e para se obter um maior desempenho de execução do sistema. A próxima seção aborda o problema de Balanceamento de Carga em sistemas distribuídos.

2.5 Balanceamento de Carga em Sistemas Distribuídos

A disponibilidade de microprocessadores de baixo custo e os avanços na tecnologia de comunicação têm estimulado o interesse na utilização de sistemas distribuídos. As principais vantagens destes sistemas são o alto desempenho, a disponibilidade de recursos e a extensibilidade a um baixo custo.

Entretanto é preciso garantir que todos os recursos sejam utilizados da melhor forma possível e que haja trabalho suficiente e compatível com a capacidade do sistema. Para que todos os recursos tenham uma carga uniforme deve ser feito um balanceamento de carga no sistema a fim de evitar que alguns recursos fiquem ociosos enquanto outros fiquem sobrecarregados. O problema reside em como distribuir (ou escalonar) os processos entre os recursos a fim de minimizar o tempo de execução e os atrasos causados pelo sistema de comunicação e/ou maximizar a utilização dos recursos do sistema (SHIVARATRI et al., 1992).

O balanceamento de carga tem sido um dos principais temas de discussão no desenvolvimento de sistemas distribuídos. Existem dois tipos de escalonamentos: o escalonamento estático e o escalonamento dinâmico. No escalonamento estático, a distribuição de tarefas pelos processadores é feita antes que os processos comecem sua execução, ou seja, a tomada de decisão de escalonamento é realizada em tempo de compilação. Isto é possível quando há conhecimento em tempo de compilação sobre os tempos de execução das tarefas e os recursos de processamento necessários para executá-las.

Em contrapartida, no escalonamento dinâmico ocorre uma redistribuição de processos (carga) entre os diversos processadores do sistema em tempo de execução. A redistribuição é realizada transferindo-se as tarefas dos processadores com maior carga para aqueles com menor carga, visando

aumentar o desempenho das aplicações. A desvantagem deste método é o custo adicional de processamento em função da transferência de informações sobre a carga de trabalho entre os processadores, do processo de decisão para escolher os processos e processadores para transferência de tarefas e dos atrasos de comunicação para realocação das tarefas.

Com o fato dos processadores cada vez mais rápidos e das redes locais com altas velocidades, tem-se mostrado viável promover o deslocamento da carga de equipamentos sobrecarregados para outros com pouca carga. Este contexto vem estimulando estudos voltados à otimização do balanceamento de carga para aplicações distribuídas em redes de computadores (ELÄSSER et al., 2000).

O escalonamento dinâmico subdivide-se em fisicamente distribuído e fisicamente centralizado. No escalonamento fisicamente centralizado, um único processador é responsável pelas funções de balanceamento de carga para todo o sistema. Os outros processadores enviam informações atualizadas para o processador central que, baseado em suas informações, decide quando e para onde alocar as tarefas. A vantagem deste esquema, se comparado ao esquema distribuído, está na redução do tráfego de informações na rede. Sua desvantagem está na existência de apenas um único ponto de distribuição de informação na rede, o qual pode tornar-se um gargalo para o sistema.

Já no escalonamento fisicamente distribuído, o processo de decisão é distribuído entre os diversos processadores do sistema, sendo que cada processador possui autonomia para decidir a respeito do escalonamento. A vantagem deste método é que todos os processadores possuem uma imagem da situação do sistema, aumentando desse modo a tolerância a falhas. A desvantagem está relacionada à sobrecarga causada devido à quantidade de comunicação entre os processadores.

Segundo Ferreira (2001), quando se deseja otimizar os recursos computacionais existentes é interessante que o balanceamento seja feito de forma dinâmica, ou seja, que os objetos possam migrar de um nó para outro de acordo com a demanda por serviços. Além disso, é interessante também que os objetos possam ser replicados. As réplicas aumentam a disponibilidade do sistema e garantem a continuidade das operações em caso de falhas. As características de flexibilidade e dinamismo da arquitetura definida por Ferreira (2001) estão fundamentadas na capacidade de migrar e replicar os objetos da aplicação.

2.5.1 Ativação do Serviço de Balanceamento de Carga

O serviço de balanceamento de carga pode ser acionado diante da ocorrência de alguns eventos. A literatura enumera, como básicos, os seguintes eventos (NOGUEIRA et al., 2001):

- a chegada de uma carga ao nó;
- término da execução (partida) de uma carga;
- início do período de ativação, quando a ativação ocorrer periodicamente.

Estes eventos expressam circunstâncias nas quais o serviço de balanceamento pode ser ativado. Todavia, nem sempre a ocorrência de todos estes eventos precisa ser considerada pelo serviço de balanceamento.

Alguns serviços de balanceamento são ativados pela chegada de cargas, como por exemplo, os serviços que transferem uma carga que chega para um nó aleatoriamente escolhido, condicionados apenas ao estado de sobrecarga local. Em contrapartida, outros serviços de balanceamento desconsideram a chegada e a partida de cargas, mas procuram manter, dentro de um patamar aceitável, a diferença entre a carga local e as cargas dos vizinhos imediatos, periodicamente efetuando, se necessário, o balanceamento das cargas.

2.5.2 Abordagens de Balanceamento de Carga Existentes

O Instituto Nacional de Pesquisas Espaciais (INPE) concentra esforços no desenvolvimento tecnológico da área espacial. Dentre as atuações do INPE, destacam-se as ações de inovação tecnológica espacial, que buscam garantir o resultado sócio-econômico da atividade de pesquisa e do desenvolvimento espacial.

Atualmente, o INPE controla em órbita os satélites SCD-1, SCD-2, CBERS-2 e CBERS-2B. Novos acordos foram estabelecidos para o desenvolvimento de satélites tecnologicamente mais avançados, como recentemente o acordo estabelecido com a China, que contempla a construção dos satélites CBERS-3 e CBERS-4. Em virtude dos novos programas de satélites propostos, faz-se necessária a atualização da infra-estrutura computacional de rastreio e controle de satélites do INPE.

Em relação à infra-estrutura de rastreio, os satélites criados pelo INPE são controlados pelo Centro de Controle de Satélites (CCS), que é constituído por um complexo computacional que assegura o funcionamento nominal do satélite desde sua injeção em órbita até o fim de sua vida útil. O CCS utiliza um software aplicativo, denominado Software de Controle de Satélites, para realizar o controle dos satélites em órbita.

Ferreira (2001) identificou a necessidade de atualização do Software de Controle de Satélites devido à complexidade e aos avanços tecnológicos presentes nos projetos de novos satélites. Segundo este autor, os projetos atuais de construção de novos satélites incorporam novas funcionalidades a fim de propiciar um maior conhecimento do espaço e da Terra. Estas novas funcionalidades conferem maior complexidade aos satélites e ao software utilizado para o seu controle.

Ferreira (2001) considera que a evolução da área espacial requer um software de controle de satélites que seja flexível, disponível e capaz de gerenciar os recursos computacionais de forma otimizada para atender à demanda de controle de novos satélites sem comprometer os resultados destas missões espaciais. Este autor sugere que estas características sejam agregadas ao Software de Controle de Satélites do CCS-INPE adotando-se a tecnologia de Objetos Distribuídos, que divide aplicações cliente-servidor em objetos autogerenciáveis capazes de interoperar mesmo com sistemas operacionais e redes heterogêneas.

Assim, Ferreira (2001) propõe uma arquitetura flexível e dinâmica para objetos distribuídos aplicada ao Software de Controle de Satélites do INPE, denominada arquitetura SICSD (*SI*stema de *C*ontrole de *S*atélites *D*istribuído). Esta arquitetura propõe que seja realizado o balanceamento de carga em um sistema no qual atua o Software de Controle de Satélites a fim de otimizar a utilização dos recursos computacionais existentes.

A arquitetura SICSD (FERREIRA, 2001) confere autonomia a cada nó do sistema para efetuar o balanceamento de carga em um tempo aleatório e propõe um conjunto de mecanismos para realizar o balanceamento de carga, dentre os quais encontram-se dois importantes mecanismos: a migração de objetos de máquinas saturadas para máquinas ociosas e a replicação de objetos cujos serviços são freqüentemente requisitados.

Diante da previsão de aumento do consumo de recursos computacionais devido ao controle de novos satélites, este trabalho de pesquisa propõe no contexto da área espacial um serviço de balanceamento de carga complementar ao desenvolvido por Ferreira (2001). A abordagem de balanceamento de carga proposta neste trabalho de pesquisa visa otimizar ainda mais a utilização dos recursos computacionais de um sistema, em

especial, do sistema dedicado ao software de controle dos satélites atuais e futuros do INPE.

Considera-se a abordagem proposta neste trabalho de pesquisa complementar à arquitetura SICSD no sentido de que ambas podem atuar em um mesmo sistema, de forma independente, pois apresentam procedimentos distintos de ativação do serviço de balanceamento de carga: a arquitetura SICSD ativa o balanceamento de carga de forma descentralizada a um tempo aleatório de cada nó, enquanto o serviço de balanceamento de carga proposto neste trabalho de pesquisa ativa o balanceamento de carga a cada invocação de serviço entre dois objetos (cliente e servidor) de uma aplicação distribuída.

Outros autores propõem trabalhos com o objetivo de otimizar os recursos computacionais de sistemas. No entanto, os serviços de balanceamento de carga propostos por estes autores, a saber, El-Abd e El-Bendary (1998), Schlemer (2002), Yang et al. (2003) e Suri et al. (2007), não contemplam algumas características abordadas na proposta deste trabalho de pesquisa.

Os trabalhos de Schlemer (2002) e Yang et al. (2003) propõem serviços de balanceamento de carga centralizados, ou seja, nestes serviços existem nós centrais únicos gerenciadores de todo o processo de balanceamento de carga. Em contrapartida, o serviço de balanceamento de carga proposto neste trabalho de pesquisa permite que as decisões de balanceamento de carga sejam fisicamente distribuídas entre os nós do sistema. Dessa forma, o serviço de balanceamento de carga proposto fica menos propenso a falhas devido a não existência de um ponto único de falha no sistema.

Outra característica não contemplada pelos autores El-Abd e El-Bendary (1998), Yang et al. (2003) e Suri et al. (2007) refere-se ao diferencial deste trabalho de pesquisa, o qual consiste em realizar um balanceamento de carga orientado a caso de uso. Um caso de uso representa um conjunto de objetos

que interagem entre si para o provimento de uma determinada funcionalidade da aplicação ao usuário.

A idéia de prover um serviço de balanceamento de carga orientado a caso de uso consiste em buscar realocar objetos de nós sobrecarregados para nós que, além de apresentarem níveis de carga adequados, estabeleçam alguma comunicação com os objetos realocados. Em resumo, uma vez que se faz necessário realocar objetos para promover o balanceamento de carga do sistema, a idéia consiste em fazer com que objetos que se comunicam entre si fiquem fisicamente localizados no mesmo nó para reduzir acessos à rede nas invocações atuais e futuras.

2.5.3 Índices de Carga

Qualquer que seja a política de escalonamento a ser utilizada com o objetivo de promover o balanceamento de carga do sistema é preciso utilizar informações sobre a carga de trabalho de cada um dos elementos envolvidos no processo. Estas informações são chamadas de índices de carga e são utilizadas para quantificar o conceito de carga (BRANCO, 2002). Segundo Ferrari e Zhou (1987), um índice de carga, preferencialmente, é uma variável não negativa que assume o valor zero se o recurso estiver ocioso, aumentando gradativamente à medida que a carga aumenta.

A eficiência do esquema de balanceamento de carga pode ser altamente afetada pelo modo através do qual a informação de carga é escolhida e quando esta informação é atualizada. Atualizações periódicas do índice de carga são necessárias, uma vez que as cargas dos nós variam constantemente, fazendo com que os índices de carga tornem-se obsoletos (desatualizados). Uma atenção especial deve ser dada à escolha do intervalo de atualização: a sobrecarga causada por intervalos pequenos pode reduzir as vantagens de

manter índices de carga atualizados; por outro lado, grandes intervalos de atualização podem tornar os índices de carga obsoletos.

Não existe, entretanto, um consenso quanto à eficácia e à eficiência dos índices de carga que podem ser adotados para promover o balanceamento de carga de um sistema. Na realidade, geralmente combinam-se mais de um índice de carga. Uma lista parcial dos possíveis índices de carga inclui os seguintes índices:

- Tamanho da fila da CPU: o índice de carga é uma função do tamanho da fila de “pronto”. Para se determinar o índice de carga pode-se utilizar o tamanho instantâneo da fila da CPU ou o tamanho médio da fila nos últimos *t* segundos. Alguns índices de carga são definidos em função dos tamanhos de várias filas, como por exemplo, tamanhos médios das filas de CPU e de memória;
- Utilização da CPU: alguns pesquisadores consideram a utilização da CPU um bom indicador de carga. Novamente, valores instantâneos ou médios de utilização do processador podem ser utilizados como índices de carga;
- Tempo de resposta ou tempo de processamento: sugerem-se diversas variações ou combinações do tempo de resposta de uma tarefa como índices de carga;
- Funções agregadas: um índice de carga pode ser definido como uma função dos índices mencionados acima.

Há diferenças na eficiência de cada índice de carga, mas pesquisas realizadas (FERRARI e ZHOU, 1987; KUNZ, 1991) relatam que os índices de carga mais simples são particularmente os mais eficientes, refletindo melhor o estado do sistema. No entanto, índices de carga são métricas que sofrem alterações muito rápidas devido às flutuações da carga de trabalho do sistema (BRANCO, 2002). Assim, é preciso utilizar técnicas apropriadas para analisar os índices de

carga a fim de que se tenha uma indicação atualizada e correta sobre a quantidade de carga presente em uma máquina.

A utilização de índices de carga tem como objetivo indicar os seguintes estados de uma máquina: (i) se a máquina está ociosa (condição de inexistência de carga ou um valor pouco expressivo); (ii) se a máquina está sobrecarregada (condição em que a política de escalonamento deve evitar a alocação de processos para serem executados nesta máquina e/ou considerar a possibilidade de remover alguns processos para aliviar a carga da máquina); ou (iii) se a máquina está em uma situação normal de processamento com disponibilidade para receber outros processos.

Como escalonar os processos visando o balanceamento de carga de um sistema é um problema que precisa do apoio de técnicas apropriadas. A área de Inteligência Artificial (IA) oferece suporte às aplicações que envolvem processos de tomada de decisão. Em decorrência desta característica da área de IA, buscou-se identificar durante o desenvolvimento deste trabalho de pesquisa uma técnica de IA que pudesse auxiliar na geração de soluções para o problema de balanceamento de carga de sistemas distribuídos.

A técnica de Redes Neurais Artificiais foi considerada adequada para prover este auxílio, uma vez que o processo de balanceamento de carga em sistemas distribuídos efetua tomadas de decisão com base em conjuntos de dados. As diversas aplicações da técnica de Redes Neurais existentes na literatura mostram que esta técnica é capaz de gerar soluções aos problemas de classificação de dados. Portanto, a arquitetura de balanceamento de carga proposta adota a técnica de Redes Neurais para auxiliar na classificação de dados essenciais às tomadas de decisão do processo de balanceamento de carga.

A arquitetura de balanceamento de carga proposta contempla a utilização de dois índices de carga para aferir o estado de carga de um nó: o índice de utilização de CPU e o índice de utilização de memória. As informações destes índices de carga são fornecidas para uma Rede Neural Artificial, que é responsável por classificar as informações destes índices de carga em um único nível de carga para o nó.

Esta classificação do nó em um nível de carga varia de “muito ocioso” a “muito carregado”, sendo posteriormente utilizada pela arquitetura de balanceamento de carga proposta no processo de tomada de decisão do balanceamento de carga do sistema.

Assim, o próximo capítulo apresenta conceitos sobre Redes Neurais Artificiais, detalhando a arquitetura de redes neurais utilizada na abordagem de balanceamento de carga proposta neste trabalho de pesquisa.

3 REDES NEURAIS ARTIFICIAIS

As redes neurais artificiais podem ser utilizadas para contribuir na solução de diversos tipos de problemas. Esta técnica inspira-se em uma unidade elementar do cérebro humano, o neurônio, e a sua eficiência deve-se à simulação de um conjunto de características dos neurônios biológicos. Dentre estas características encontram-se a adaptabilidade, a aprendizagem por meio de exemplos, o processamento paralelo e distribuído, a capacidade de generalização e a tolerância a falhas. Estas características tornaram esta técnica atrativa para ser aplicada no contexto deste trabalho de pesquisa.

3.1 Motivação para a Utilização de Redes Neurais Artificiais

A partir do momento em que as máquinas começaram a evoluir, um grande desejo do homem tem sido a criação de uma máquina que possa operar independentemente do controle humano, sendo capaz de aprender e interagir com ambientes incertos (desconhecidos por ela). Estas máquinas são muito úteis onde a interação humana é perigosa, tediosa ou impossível, como em reatores nucleares, no combate ao fogo, em operações militares e na exploração do espaço.

Os organismos humanos são uma fonte de motivação para o desenvolvimento destas máquinas, pois proporcionam diversas dicas para o desenvolvimento de algoritmos de aprendizado e de adaptação. Enquanto os computadores funcionam de modo seqüencial, proporcionando maior eficiência na resolução de tarefas constituídas por etapas, o cérebro humano funciona de modo paralelo e é mais eficiente na resolução de tarefas que exigem muitas variáveis. Assim, as redes neurais são projetadas de forma a apresentarem algumas características exclusivas de sistemas biológicos.

Os sistemas de computação baseados em redes neurais têm a capacidade de receber ao mesmo tempo várias entradas e distribuí-las de maneira organizada. Geralmente, as informações armazenadas por uma rede neural são compartilhadas por todas as suas unidades de processamento.

Em um sistema de rede neural, a informação pode parecer ter representação redundante, porém, o fato da informação se encontrar distribuída por todos os elementos da rede significa que mesmo que parte da rede seja destruída, a informação contida nesta parte ainda estará presente na rede e poderá ser recuperada. Portanto, a redundância na representação da informação em uma rede neural torna o sistema tolerante a falhas. Devido ao paralelismo, as possíveis ocorrências de falhas nos neurônios não causam efeitos significantes para o desempenho do sistema.

A principal força na estrutura de uma rede neural reside em sua habilidade de adaptação e aprendizagem. A habilidade de adaptação e aprendizagem significa que modelos de redes neurais podem lidar com dados imprecisos e situações não totalmente definidas. Uma rede neural treinada de maneira razoável tem a habilidade de generalizar quando lhe são apresentadas entradas que não estão presentes em dados já conhecidos pela rede. As redes neurais podem ter várias entradas e várias saídas, e são facilmente aplicáveis a sistemas com muitas variáveis.

Segundo Braga et al. (2000), aplicam-se redes neurais para gerar soluções a diferentes tipos de problemas. Este autor lista algumas aplicações de redes neurais, como: análise do mercado financeiro por grupos de investimento, reconhecimento ótico de caracteres, controle de processos industriais, aplicações climáticas, identificação de fraude de cartão de crédito, diagnóstico médico, análise e processamento de sinais, robótica, classificação de dados, reconhecimento de padrões em linhas de montagem, filtros contra ruídos eletrônicos, análise de imagens, análise de voz e avaliação de crédito.

3.2 O Neurônio Artificial

O elemento básico que forma uma rede neural artificial é o neurônio artificial, conhecido também por nó ou elemento processador. Ele foi projetado baseado no funcionamento de um neurônio natural.

McCulloch e Pitts interpretaram o funcionamento do neurônio biológico como sendo um circuito de entradas binárias combinadas por uma soma ponderada (com pesos) produzindo uma entrada efetiva. A Figura 3.1 ilustra o modelo de um neurônio artificial, interpretado por McCulloch e Pitts a partir da estrutura e funcionamento de um neurônio biológico.

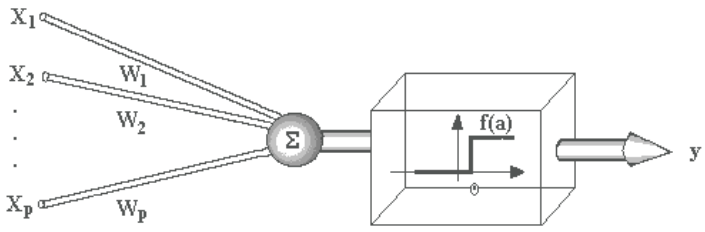


Figura 3.1 – Modelo de um neurônio de McCulloch e Pitts

A função utilizada para o cálculo de ativação geralmente é algum tipo de função não-linear, a qual garante a plena funcionalidade das redes neurais com múltiplas camadas de neurônios. Redes neurais que processam dados analógicos devem utilizar funções com formato sigmoidal, como tangente hiperbólica, seno, etc. Em redes utilizadas para processar valores discretos a função deve ser do tipo degrau.

De acordo com uma ponderação dos sinais de entrada, realizada pela função de ativação, o neurônio pode ser ativado, enviando um sinal de saída. Este sinal de saída será propagado de acordo com a topologia de interconexão da rede de neurônios.

Uma rede neural artificial pode ser comparada a um grafo orientado composto por nós ou elementos processadores interconectados que operam em paralelo. Cada elemento processador (ou neurônio artificial) possui um determinado número de entradas e somente um único sinal de saída, que se propaga através das conexões com os outros elementos processadores. A cada entrada de um elemento processador está associado um peso sináptico que pode ser excitatório (positivo) ou inibitório (negativo), normalmente variando de -1 à +1. O sinal de entrada pode assumir uma variação contínua (desde -1 à +1) ou discreta (restrito aos valores binários, 0 ou 1) (LAWRENCE, 1992).

O elemento processador avalia seus sinais de entrada realizando um somatório ponderado das suas entradas (através dos pesos sinápticos associados a cada entrada), seguindo a equação:

$$u_j = \sum_{i=1}^{N} w_{ij} o_i \qquad (3.1)$$

onde u_j representa a soma ponderada dos N sinais de entrada do neurônio j, w_{ij} representa o valor do peso sináptico associado à conexão entre o neurônio i e j, e o_i representa a saída do i-ésimo neurônio.

De uma maneira mais simplificada, isto significa somar todos os sinais de entrada que chegam a um neurônio levando em consideração o peso das conexões envolvido em cada sinal de entrada.

O sinal de saída do elemento processador é encontrado aplicando-se o somatório ponderado das suas entradas em uma função ativação que determinará seu valor de saída (nível de ativação):

$$o_j = f(u_j) \tag{3.2}$$

onde *f* = função ativação do neurônio *j*.

Os neurônios artificiais organizados e conectados de várias formas podem resultar em diferentes arquiteturas neurais, com características e aplicações distintas. No entanto, as arquiteturas neurais existentes com diferenças significativas entre si são chamadas indistintamente de redes neurais, pelo fato do bloco básico de construção destas arquiteturas ser o neurônio artificial.

A arquitetura mais simples de uma rede neural é o Perceptron de Única Camada construído em torno do modelo de neurônio de McCulloch-Pitts. O Perceptron é usado para a classificação de padrões ditos linearmente separáveis, ou seja, padrões que se encontram em lados opostos de um hiperplano (HAYKIN, 2001).

O Perceptron de Única Camada não atende a todas as necessidades de classificação de padrões. Quando se faz necessário realizar cômputos mais complexos do que a classificação com separabilidade linear, emprega-se uma estrutura com diversos neurônios encadeados chamada Perceptron de Múltiplas Camadas (*Multi Layer Perceptron* – MLP).

A arquitetura de balanceamento de carga proposta neste trabalho de pesquisa utiliza uma rede neural de arquitetura Perceptron de Múltiplas Camadas para conseguir realizar classificações dos níveis de carga dos nós de um sistema. Esta rede realiza o treinamento de forma supervisionada pelo algoritmo de retropropagação de erro. A seção a seguir descreve a rede neural de

arquitetura Perceptron de Múltiplas Camadas e o algoritmo de treinamento utilizado por esta rede.

3.3 Perceptron de Múltiplas Camadas

Segundo Haykin (2001), as principais características da rede neural Perceptron de Múltiplas Camadas são as seguintes: possui múltiplas camadas, as entradas e saídas podem ser tanto analógicas quanto digitais e não existe a restrição de separabilidade linear entre classes.

Em uma rede neural Perceptron de Múltiplas Camadas, o fluxo de informação ocorre em camadas de neurônios, sem que haja realimentação (retorno da informação da saída para a entrada de algum dos neurônios da rede). Esta organização em camadas e a ausência de realimentação é própria desta arquitetura.

A Figura 3.2 ilustra a arquitetura de uma rede neural Perceptron com duas Camadas Intermediárias.

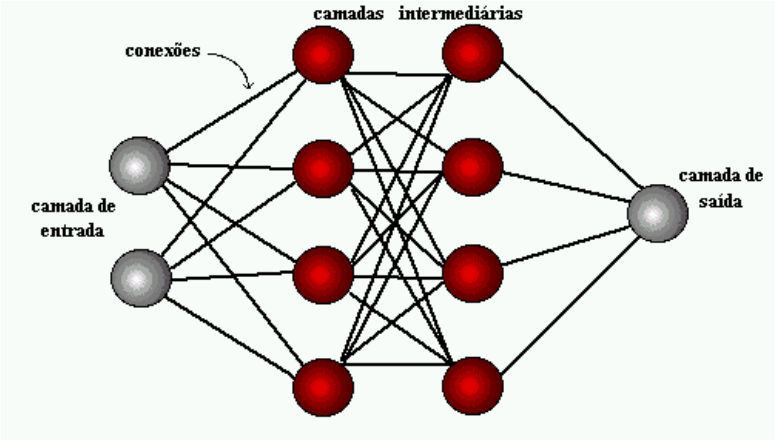


Figura 3.2 – Rede neural Perceptron de Múltiplas Camadas
Fonte: Retirada de Hernandez (2003)

As redes neurais Perceptrons de Múltiplas Camadas têm sido aplicadas com sucesso para resolver diversos problemas difíceis através do seu treinamento de forma supervisionada com um algoritmo popular, conhecido como algoritmo de retropropagação de erro (*error back-propagation*), baseado na regra de aprendizagem por correção de erro (BRAGA et al., 2000; HAYKIN, 2001).

Segundo Haykin (2001), esta aprendizagem é constituída por dois passos: (i) um para frente, a propagação; e (ii) um para trás, a retropropagação. No primeiro passo, um padrão de atividade (vetor de entrada) é aplicado aos nós sensoriais da rede e seu efeito se propaga através da rede, camada por camada. Finalmente, um conjunto de saídas é produzido como a resposta real da rede.

No segundo passo, os pesos sinápticos são todos ajustados de acordo com uma regra de correção de erro. Especificamente, a resposta real da rede é subtraída de uma resposta desejada (alvo) para produzir um sinal de erro. Este sinal de erro é então propagado para trás através da rede, contra a direção das conexões sinápticas. Os pesos sinápticos são ajustados para fazer com que a resposta real da rede se mova para mais perto da resposta desejada.

Assim, sempre que ocorre erro no reconhecimento de uma entrada, o algoritmo de aprendizado supervisionado de retropropagação de erro (*error backpropagation*) conduz um ajuste sináptico. O ajuste sináptico procura corrigir os pesos de modo que se produza a saída desejada diante da respectiva entrada, representando assim o aprendizado em cada neurônio do fato apresentado. Pode-se dizer que ao final do processo de aprendizado, o conhecimento dos neurônios da rede neural reside nos pesos sinápticos (HERNANDEZ, 2003).

4 AGENTES DE SOFTWARE

Com o propósito de fornecer subsídios para a compreensão da arquitetura de balanceamento de carga proposta, este capítulo apresenta uma fundamentação teórica sobre o conceito de agentes, uma vez que a arquitetura proposta é constituída por um grupo de agentes que interagem entre si para o provimento do serviço de balanceamento de carga.

Assim, este capítulo descreve os motivos pelos quais se concebeu a arquitetura multi-agente de balanceamento de carga proposta utilizando o conceito de agentes e as características gerais de um sistema multi-agente.

4.1 Definição de Agente

A definição de agente não é consensual (LIEBERMAN, 1997). O maior problema na definição do termo agente é que se trata de um vocábulo extensamente usado por diversos pesquisadores que trabalham em áreas estreitamente relacionadas (SYCARA, 1998). Assim, existem várias definições vinculadas a pesquisadores empenhados no estudo e no desenvolvimento de agentes. A seguir, apresentam-se algumas destas definições.

Para Franklin e Graesser (1996), um agente é um sistema situado dentro de um ambiente e como parte deste ambiente, sente e age sobre o mesmo, ao longo do tempo, em busca de cumprir sua própria agenda e de maneira que afete o seu próprio comportamento no futuro.

Para Ferber (1999), um agente é uma entidade física ou virtual com as seguintes características: é capaz de agir em um ambiente, pode comunicar-se diretamente com outros agentes, é dirigido por um conjunto de tendências, possui recursos próprios, é capaz de perceber o ambiente com uma extensão

limitada, tem somente uma representação parcial do seu ambiente, oferece serviços, seu comportamento tende a satisfazer os seus objetivos, e depende de suas percepções, de suas representações e da comunicação que ele estabelece com o ambiente.

Por entidade física, entende-se algo que possui uma existência física e age no mundo real, como um robô, uma espaçonave ou um carro. Por entidade virtual entende-se algo que não existe na forma física, mas existe na forma lógica, como um componente de software ou um modelo computacional.

Para Russell e Norvig (2004), um agente pode ser visto como algo que observa o ambiente por meio de sensores e age neste ambiente por intermédio de atuadores. Para exemplificar este conceito de agente, pode-se fazer um comparativo entre agentes humanos, agentes robôs e agentes de software.

Um agente humano possui olhos, ouvidos e outros órgãos que funcionam como sensores no ambiente, e possui mãos, pernas, boca e outras partes do corpo que agem como atuadores. Um agente robô, por sua vez, pode possuir câmeras de vídeo e outros sensores para observar o ambiente, e pode ter vários motores como atuadores. Um agente de software recebe seqüências de digitações, conteúdo de arquivos e/ou pacotes de rede como entradas sensórias e atua sobre o ambiente exibindo algo na tela, gravando arquivos e/ou enviando pacotes de rede.

Em termos matemáticos, Russell e Norvig afirmam que o comportamento de um agente é descrito por uma função que mapeia qualquer seqüência de percepções específica para uma ação. Para um agente de software, esta função é implementada por um programa de agente, que consiste em uma implementação concreta.

Caire et al. (2002) definem um agente como sendo uma entidade autônoma que é capaz de desempenhar alguma função específica. Eles consideram esta capacidade como sendo os serviços do agente. Um serviço possui o nível de conhecimento análogo a uma operação (método) de um objeto.

A característica de autonomia significa que as ações de um agente não são somente ditadas por interações ou eventos externos, mas também pela própria motivação do agente. Os autores capturam a motivação de um agente em um atributo denominado propósito. O propósito de um agente determina se um agente concordará ou não em desempenhar um serviço requisitado e também em como este serviço será oferecido. Um agente pode ser tanto um agente humano quanto um agente de software, sendo ambos considerados especializações de agente.

Neste trabalho de pesquisa adota-se a definição de agente proposta por Caire et al. (2002), uma vez que esta definição representa um dos conceitos fundamentais de uma metodologia de engenharia de software orientada a agentes, proposta pelos mesmos autores, denominada MESSAGE. A metodologia MESSAGE encontra-se descrita em detalhes no Capítulo 5 e a sua adoção oferece o benefício de se obter primitivas para descrever algumas estruturas presentes em agentes.

A seção a seguir descreve os motivos pelos quais modelou-se a arquitetura proposta utilizando o conceito de agentes.

4.2 Diretrizes para o Uso do Conceito de Agentes

Na arquitetura proposta adotou-se o conceito de agentes, um conceito que originou-se dentro da comunidade de Inteligência Artificial (IA) nos anos 70 e infiltrou-se na comunidade de Engenharia de Software no começo dos anos 90. A adoção deste conceito embasou-se em um conjunto de diretrizes, proposto

pelos participantes do projeto P907 do EURESCOM – European Institute for Research and Strategic Studies in Telecommunications (MILGROM, 2001), para identificar tipos de problemas para os quais os autores consideram adequada a concepção de um sistema de software de forma orientada a agentes.

Estas diretrizes identificam propriedades existentes em softwares complexos, que caracterizam situações nas quais a adoção do conceito de agentes é promissora.

O que de fato motivou a adoção do conceito de agentes foi a constatação de que, não apenas uma diretriz, mas um conjunto de diretrizes propostas pelo projeto P907 do EURESCOM (MILGROM, 2001) são simultaneamente válidas para o problema de balanceamento de carga em análise. O conjunto de diretrizes considerado aplicável a este problema encontra-se listado a seguir.

4.2.1 Diretriz 1 – Necessidade de Controle Distribuído

Para obter soluções a problemas essencialmente complexos, estes são geralmente decompostos em subproblemas, sendo que cada subproblema pode ser tratado de forma independente. As soluções para os subproblemas identificados podem ser implementadas de forma distribuída, constituindo um sistema distribuído.

Assim, em um sistema distribuído pode-se utilizar agentes de software para controlar cada subproblema (subsistema). O controle do sistema pode ser efetuado por meio de agentes de software a fim de que não somente o sistema seja distribuído, mas também o seu controle.

4.2.2 Diretriz 2 – Necessidade de Comportamento Autônomo

Os sistemas de software podem ser projetados de acordo com uma abordagem puramente reativa. Segundo esta abordagem, o sistema toma ações somente quando estas são solicitadas por meio de pedidos explícitos. Uma maneira mais flexível de projeto de sistemas consiste na abordagem direcionada a objetivos, que considera que os sistemas devem tomar ações para tentar satisfazer os seus objetivos mesmo na ausência de pedidos explícitos.

Os sistemas baseados em agentes podem se comportar mais facilmente de forma orientada a objetivos, o que lhes confere um comportamento autônomo.

4.2.3 Diretriz 3 – Necessidade de Alta Flexibilidade e Adaptação

Muitas vezes um sistema de software terá que ser expandido ou modificado durante a sua vida de operação ou até mesmo o propósito do sistema pode evoluir. Quando esta questão for relevante, a utilização do conceito de agente de software é aplicável devido a sua característica de modularidade. Os agentes podem ser facilmente adicionados ou substituídos, o que reduz o custo de manutenção de um sistema.

4.2.4 Diretriz 4 – Necessidade de Interoperabilidade

O desenvolvimento de um sistema de software orientado a agente pode ser considerado principalmente quando um problema requer que um software interaja, possivelmente no futuro, com outro software que seja ainda desconhecido durante o projeto do primeiro.

Um sistema multi-agente pode interconectar agentes que foram desenvolvidos separadamente e que são independentes uns dos outros, permitindo que juntos ofereçam um serviço cuja abrangência supera as capacidades individuais dos agentes.

Por exemplo, o uso de agentes de software é recomendável para integrar sistemas que executam em plataformas computacionais incompatíveis. No ambiente de Tecnologia de Informação (TI) freqüentemente ocorre a fusão de duas ou mais empresas e desta fusão surge a necessidade de unificar os sistemas computacionais das empresas envolvidas. Neste caso, pode-se encapsular os sistemas computacionais, que são incompatíveis, em agentes de software para integrar estes sistemas por meio de uma interface de comunicação uniforme.

A seção a seguir aborda as características relacionadas a sistemas multi-agentes, visto que a arquitetura proposta não é constituída unicamente por um agente, mas sim por um grupo de agentes.

4.3 Definição de Sistemas Multi-Agentes

Um sistema multi-agente (SMA) consiste em um sistema constituído por um grupo de agentes que atuam em conjunto no sentido de resolver problemas que estão além de suas capacidades individuais. Para prover soluções aos problemas, os agentes de um SMA interagem entre si de modo cooperativo ou competitivo para atingir objetivos comuns ao grupo ou objetivos individuais.

O termo cooperação pode ser utilizado na literatura de SMA com significados distintos. Neste trabalho de pesquisa, assim como em Rezende (2003), adota-se este termo com o seguinte significado: dois ou mais agentes que almejam atingir um mesmo objetivo em conjunto e são conscientes deste fato.

Segundo Henderson-Sellers e Giorgini (2005), para o projeto de sistemas multi-agentes utiliza-se uma metáfora da organização humana. Nesta metáfora, estruturas e modelos da organização humana são empregados para o projeto

de SMAs e define-se que estes sistemas sejam constituídos por agentes que desempenham um ou mais papéis e interagem entre si.

Assim, para satisfazer a idéia de metáfora da organização humana, utilizam-se os conceitos de papel, de dependência social e de regras organizacionais para a modelagem de SMAs. As definições destes conceitos encontram-se especificadas, de forma particular, em cada metodologia de engenharia de software orientada a agentes existente. O Capítulo 5 introduz o conceito de metodologia de engenharia de software orientada a agentes e detalha os conceitos da metodologia MESSAGE (CAIRE et al., 2002), em especial.

5 ENGENHARIA DE SOFTWARE ORIENTADA A AGENTES

Os agentes de software requerem um tratamento adequado para o seu projeto. As metodologias de engenharia de software convencionais não são suficientes para capturar algumas características importantes e inerentes aos agentes. Este fato motivou o estudo de Metodologias de Engenharia de Software Orientadas a Agentes.

Assim, este capítulo apresenta a genealogia de um conjunto de metodologias orientadas a agentes e descreve alguns *frameworks* de avaliação de metodologias orientadas a agentes existentes na literatura. Estes *frameworks* possuem a finalidade de comparar as características das metodologias entre si a fim de identificar as metodologias com maior abrangência de recursos e de fácil aplicação.

Por fim, neste capítulo justificam-se: (i) a utilização do framework de avaliação proposto por Henderson-Sellers e Giorgini (2005) para a escolha de uma metodologia orientada a agentes adequada para a modelagem da arquitetura multi-agente de balanceamento de carga proposta; e (ii) a escolha da metodologia orientada a agentes MESSAGE por meio da aplicação deste framework.

5.1 Definições de Engenharia de Software

Uma metodologia consiste em um conjunto de métodos empregados de forma sistemática. Um método consiste em um procedimento para se obter algo desejado.

Segundo Henderson-Sellers e Giorgini (2005), existem muitos sistemas orientados a agentes desenvolvidos por meio da aplicação de métodos, mas

que carecem de uma metodologia. Estes sistemas são desenvolvidos segundo uma abordagem *ad-hoc*, que apresenta escassez de diretrizes e de uma terminologia uniforme.

A abordagem *ad-hoc* oferece flexibilidade, mas em contrapartida, torna a qualidade do sistema de software questionável, uma vez que o conhecimento e a experiência adquiridos no desenvolvimento de um sistema de software não podem ser facilmente transferidos a outros projetos. A combinação das características – presença de flexibilidade e ausência de controle – faz com que dificilmente seja possível definir uma metodologia, pois falta uma abordagem coordenada e sistemática para estabelecer métodos de trabalho.

Diante da aleatoriedade, algumas metodologias de engenharia de software orientadas a agentes foram propostas na literatura com os objetivos de: (i) prover auxílio aos desenvolvedores de software por meio da sugestão de diretrizes, de técnicas, de notações e de uma terminologia uniforme; e (ii) assegurar qualidade aos sistemas desenvolvidos por meio da garantia da aplicação coordenada de métodos.

5.2 Projetos de Sistemas Multi-Agentes

De acordo com os participantes do projeto P907 do EURESCOM (MILGROM, 2001), os projetos de engenharia de software de sistemas multi-agentes existentes se dividem em duas categorias:

- projetos que aplicam metodologias de engenharia de software convencionais a sistemas multi-agentes, modificando-as quando necessário;
- projetos que apresentam como conceito central o conceito de agentes. Estes projetos aplicam novas metodologias de engenharia de software orientadas a agentes para cobrir as fases de análise e projeto de software.

Os projetos que se encaixam na primeira categoria buscam meios adicionais para suprir a ausência de recursos das metodologias de engenharia de software convencionais para a modelagem de agentes.

Odell et al. (2000) propõem um recurso útil a esta primeira categoria de projetos de sistemas multi-agentes. Este recurso consiste em uma notação, denominada AUML – Agent UML, que provê extensões à linguagem de modelagem UML (*Unified Modeling Language*).

A UML, *Unified Modeling Language*, é uma linguagem gráfica para visualização, especificação, construção e documentação de artefatos de sistemas de software e representa a coletânea de práticas de engenharia de software que foram bem sucedidas na modelagem de sistemas complexos (Fowler et al., 2005). A UML é composta por um conjunto de diagramas utilizados para a modelagem de software, tais como: o diagrama de casos de uso, o diagrama de atividades, o diagrama de classes e o diagrama de seqüência, entre outros.

Neste trabalho de pesquisa não será descrita a linguagem UML, em detalhes, por se tratar de uma linguagem que se tornou um padrão para a modelagem de sistemas orientados a objeto. As notações e o detalhamento dos diagramas propostos pela linguagem UML podem ser consultados em Rumbaugh et al. (2004).

As extensões à linguagem UML propostas por Odell et al. (2000) são capazes de representar as características essenciais aos agentes e têm como principal objetivo propor um meio de representar protocolos de interação de agentes.

A Seção 5.3 contempla os projetos de sistemas multi-agentes pertencentes à segunda categoria, ou seja, projetos que apresentam como conceito central o

conceito de agentes por meio da aplicação de metodologias de engenharia de software inteiramente orientadas a agentes.

5.3 Metodologias de Engenharia de Software Orientadas a Agentes

As metodologias orientadas a agentes consistem em uma ramificação da área de Engenharia de Software voltada especialmente para o desenvolvimento de sistemas multi-agentes, que, portanto, contemplam as características inerentes a estes tipos de sistemas, como o comportamento autônomo, as interações inter-agentes e as estruturas organizacionais complexas dos sistemas de agentes.

5.3.1 Genealogia das Metodologias Orientadas a Agentes

As metodologias orientadas a agentes possuem diversas origens. Algumas baseiam-se em idéias da área de Inteligência Artificial, outras descendem diretamente de metodologias orientadas a objetos, outras ainda baseiam-se em uma combinação destas abordagens, e outras não se fundamentam em metodologias divulgadas na literatura.

Algumas metodologias orientadas a objeto de renome serviram de base para projetos de metodologias orientadas a agentes. A Figura 5.1 ilustra as influências diretas e indiretas de um grupo de metodologias orientadas a objetos em um grupo representativo de dez metodologias orientadas a agentes existentes na literatura, citadas em Henderson-Sellers e Giorgini (2005).

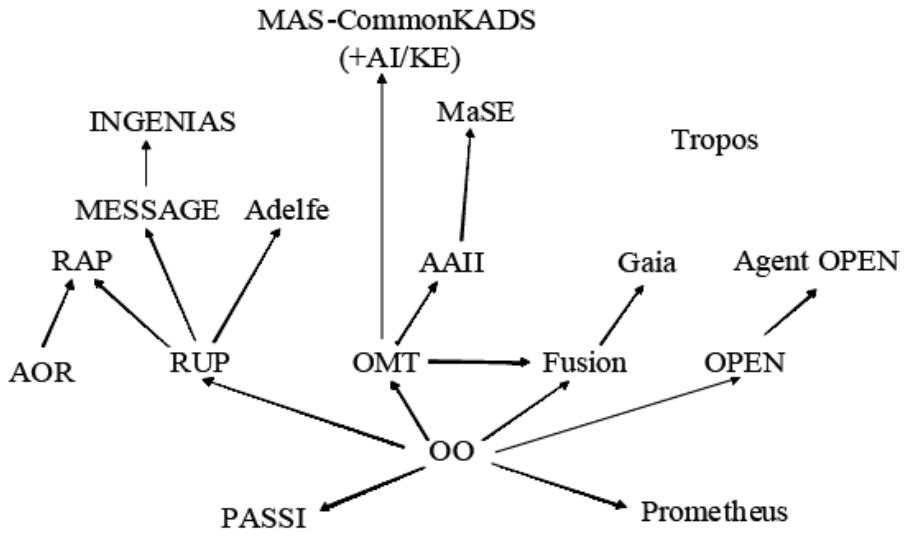


Figura 5.1 – Genealogia de um grupo de metodologias orientadas a agentes
Fonte: Retirada de Henderson-Sellers e Giorgini (2005)

A seguir, apresenta-se um resumo do histórico das influências das metodologias orientadas a objeto apresentadas na Figura 5.1:

- A metodologia orientada a agentes MaSE (WOOD e DELOACH, 2001) recebe influências da metodologia AAII de Kinny et al. (1996), que por sua vez recebeu fortes influências da metodologia orientada a objeto de Rumbaugh e amigos, a metodologia OMT (RUMBAUGH et al., 1991);
- De forma similar, a metodologia orientada a objeto Fusion (COLEMAN et al., 1994) teve forte influência no projeto da metodologia orientada a agentes Gaia (WOOLDRIDGE et al., 2000) (ZAMBONELLI et al., 2003);
- A metodologia orientada a objeto RUP formou as bases para as metodologias orientadas a agentes ADELFE (BERNON et al, 2002) e MESSAGE (CAIRE et al., 2002), que por sua vez é a base da metodologia INGENIAS (PAVON et al., 2005). Posteriormente, a metodologia RUP juntamente com a metodologia AOR de Wagner (2003) constituíram a base para a metodologia orientada a agentes RAP (TRAVETER e WAGNER, 2005);

- A metodologia orientada a objeto OPEN foi estendida para oferecer suporte ao desenvolvimento de sistemas multi-agentes, sendo denominada a partir desta extensão de Agent OPEN (DEBENHAM e HENDERSON-SELLERS, 2003);
- As metodologias PASSI (COSSENTINO, 2005) e Prometheus (PADGHAM e WINIKOFF, 2003) não descendem de uma metodologia orientada a objetos específica, mas recebem influências do paradigma de orientação a objetos, sugerindo inclusive a utilização de diagramas deste paradigma na notação da linguagem UML;
- A metodologia MAS-CommonKADS (IGLESIAS et al., 1998) é uma metodologia baseada na área de IA, mas também é influenciada por metodologias orientadas a objeto, notavelmente a metodologia OMT;
- Existem metodologias que não possuem nenhuma ligação genealógica direta com outras metodologias, sejam estas orientadas a objeto ou orientadas a agentes. Este é o caso da metodologia Tropos (PERINI et al., 2002), (BRESCIANI et al., 2004), que pelo fato de utilizar a linguagem de modelagem i* (YU, 1995) apresenta uma aparência que lhe é peculiar, diferente das metodologias que utilizam a linguagem de modelagem AUML (ODELL et al., 2000).

De forma geral, todas as metodologias orientadas a agentes citadas acima oferecem contribuições valiosas que auxiliam o projeto de sistemas baseados em agentes. No entanto, cada metodologia possui uma perspectiva única para o desenvolvimento deste tipo de sistema, enquanto ao mesmo tempo possui características que se sobrepõem às características de outras metodologias. Na verdade, nenhuma metodologia em especial pode ser considerada ideal para ser aplicada em toda situação de desenvolvimento de sistemas.

Na literatura sobre Engenharia de Software Orientada a Agentes existem modelos, chamados de frameworks de avaliação, que oferecem suporte à avaliação de metodologias orientadas a agentes. Um framework de avaliação

define critérios que permitem: (i) analisar as características das metodologias; e (ii) compará-las entre si por meio de métricas de satisfação ou não-satisfação dos requisitos definidos pelos critérios.

A seção a seguir apresenta uma análise sobre alguns frameworks de avaliação de metodologias orientadas a agentes existentes na literatura e detalha o framework proposto por Tran et al. (2003). O detalhamento das características deste framework de avaliação deve-se ao fato dos pesquisadores Henderson-Sellers e Giorgini (2005) terem adotado este framework para a realização de uma análise comparativa de um grupo de metodologias orientadas a agentes existentes. Os resultados desta análise comparativa foram utilizados neste trabalho de pesquisa para a escolha da metodologia orientada a agentes adotada para a modelagem da arquitetura multi-agente de balanceamento de carga proposta.

5.3.2 Frameworks de Avaliação de Metodologias Orientadas a Agentes

Os seguintes autores propõem frameworks de avaliação de metodologias orientadas a agentes:

- Sturm e Shehory (2003) definem um framework de avaliação quantitativa e qualitativa, e o aplica na avaliação das metodologias Gaia, Adept e Desire por meio da modelagem de um estudo de caso de Leilão;
- Dam e Winikoff (2003) utilizam um framework de avaliação baseado em atributos e em um questionário respondido por pesquisadores da área, especificamente pelos autores das metodologias analisadas e por alunos que modelaram um estudo de caso de Planejamento de Itinerário Pessoal. A avaliação comparativa utilizando este framework foi realizada para as metodologias Gaia, MaSE, MESSAGE, Prometheus e Tropos;

- Tran et al. (2003), propõem um framework de avaliação que contém critérios que analisam categorias de características presentes nas metodologias de forma a possibilitar a análise dos três componentes de uma metodologia – processo, modelos e técnicas.

Este último framework de avaliação de metodologias, proposto por Tran et al. (2003), foi constituído a partir da integração de critérios de avaliação de vários frameworks existentes. Em especial, este framework inclui os critérios de avaliação propostos por Sturm e Shehory (2003), além de adicionar novos critérios, como por exemplo, o critério: “a abordagem de desenvolvimento de sistemas multi-agentes que a metodologia oferece suporte”.

Os critérios propostos pelo framework de Tran e co-autores dividem-se em quatro categorias:

- critérios relacionados ao processo. Analisam os seguintes aspectos de uma metodologia: sua aplicabilidade, os passos existentes para o seu processo de desenvolvimento e a abordagem de desenvolvimento adotada;
- critérios relacionados às técnicas. Avaliam a usabilidade das técnicas oferecidas por uma metodologia para a execução dos passos do seu processo de desenvolvimento e/ou para o desenvolvimento dos seus modelos;
- critérios relacionados aos modelos. Avaliam os seguintes aspectos dos modelos de uma metodologia: os conceitos representados pelos modelos, a qualidade das notações e as características de agentes que os modelos oferecem suporte;
- critérios relacionados às características de suporte. Avaliam características suplementares de alto nível de uma metodologia orientada a agentes, como por exemplo, as características de disponibilidade de ferramentas de software e de suporte à ontologia.

Como este framework propõe um vasto conjunto de critérios para cada categoria, estes não serão listados, mas encontram-se disponíveis para consulta em Tran et al. (2003).

Em virtude do framework de Tran et al. (2003) oferecer uma vasta abrangência de critérios, os autores Henderson-Sellers e Giorgini (2005) escolheram este framework para utilizar como suporte à avaliação de um conjunto de dez metodologias orientadas a agentes. O resultado desta análise comparativa deu origem a um framework voltado a auxiliar projetistas de sistemas multi-agentes na seleção, entre o grupo de metodologias consideradas pelos autores, da metodologia orientada a agentes mais apropriada para sistemas computacionais de qualquer tipo, ou seja, para sistemas computacionais que resolvam qualquer tipo de problema.

5.3.3 Resultado da Aplicação do Framework de Henderson-Sellers e Giorgini

O framework de Henderson-Sellers e Giorgini (2005) constitui uma análise comparativa interessante, realizada por meio da aplicação do framework de Tran et al. (2003), do seguinte grupo de metodologias orientadas a agentes: Gaia, Tropos, MAS-CommonKADS, Prometheus, Passi, Adelfe, MaSE, RAP, MESSAGE e INGENIAS.

Para a concepção da arquitetura proposta neste trabalho de pesquisa, utilizaram-se os resultados obtidos desta análise comparativa, que permitiram identificar uma metodologia mais adequada ao projeto da arquitetura multi-agente de balanceamento de carga proposta.

O resultado da aplicação do framework de Henderson-Sellers e Giorgini (2005) para a escolha de uma metodologia adequada à modelagem da arquitetura

multi-agente de balanceamento de carga proposta indicou a metodologia MESSAGE (CAIRE et al., 2002) como a metodologia mais adequada.

Dentre o vasto conjunto de critérios, utilizados pelos pesquisadores Henderson-Sellers e Giorgini para efetuar as comparações entre o conjunto de metodologias considerado, observou-se que a metodologia MESSAGE satisfaz aos critérios listados na Tabela 5.1 enquanto as outras metodologias não os satisfazem ou satisfazem-nos em níveis de intensidade inferiores.

Tabela 5.1 – Critérios de avaliação satisfeitos pela metodologia MESSAGE

Critério	**Descrição do Critério**
Suporte ao conceito de papel	A metodologia emprega o conceito de “papel” na análise e projeto de sistemas multi-agentes?
Suporte às atitudes mentais dos agentes	A metodologia oferece suporte à definição das atitudes mentais dos agentes, ou seja, seus objetivos, compromissos, etc.?
Suporte à arquitetura do sistema multi-agente	A metodologia oferece suporte à especificação da arquitetura do sistema, permitindo esboçar uma visão geral dos componentes do sistema e suas conexões?
Suporte à estrutura organizacional	A metodologia contempla a especificação da estrutura organizacional do sistema ou dos relacionamentos sociais entre agentes?
Suporte a objetos	A metodologia permite a utilização ou a integração de objetos convencionais em um sistema multi-agente?
Suporte à habilidade de comunicação	A metodologia oferece suporte à modelagem do comportamento cooperativo ou competitivo de um grupo de agentes, realizado por meio da comunicação entre eles, para alcançar objetivos de planejamento?
Suporte à característica de autonomia	A metodologia oferece suporte à característica de autonomia de um agente?
Fornecimento de exemplos	A metodologia oferece exemplos de aplicação das técnicas que propõe?
Suporte de software	Existem softwares que oferecem suporte à construção dos modelos que a metodologia propõe?

Assim, a verificação da satisfação do conjunto de critérios descrito na Tabela 5.1 conduziu a escolha da metodologia MESSAGE para o projeto da arquitetura multi-agente de balanceamento de carga proposta neste trabalho de pesquisa. A seção a seguir descreve os principais conceitos desta metodologia.

5.4 Metodologia MESSAGE

O instituto europeu de pesquisas na área de telecomunicações EURESCOM – European Institute for Research and Strategic Studies in Telecommunications – direcionou um dos seus projetos, o P907 (CAIRE et al., 2002; MILGROM, 2001), para a definição de uma metodologia de análise e projeto baseada em agentes com o objetivo de aplicar esta metodologia no ambiente de telecomunicações. A metodologia definida recebeu o nome de MESSAGE – Methodology for Engineering Systems of Software Agents.

MESSAGE (CAIRE et al., 2002) utiliza a linguagem UML – Unified Modeling Language (RUMBAUGH et al., 2004) como ponto de partida, adicionando conceitos e relacionamentos necessários à modelagem orientada a agentes. Esta modelagem busca descrever a forma como um sistema multi-agente funciona para a realização de um objetivo coletivo, similarmente às organizações humanas e sociedades, e o comportamento cognitivo dos agentes, com o auxílio da área de Inteligência Artificial.

Na realidade, a metodologia MESSAGE propõe extensões ao MOF, a linguagem de meta-modelo da UML que lhe confere extensibilidade. MESSAGE estende o meta-modelo da UML com novos conceitos de entidades orientadas a agentes. A maioria destes conceitos de entidades podem ser classificados em três categorias: categoria de Entidade Concreta, categoria de Atividade e categoria de Estado Mental. A primeira categoria inclui os conceitos de Agente, Organização, Papel e Recurso. A segunda categoria contém os

conceitos de Tarefa, Interação e Protocolo de Interação. A terceira tem um foco maior no conceito de Objetivo.

5.4.1 Conceitos da Metodologia MESSAGE

Descrição dos Conceitos da Categoria Entidade Concreta

Conceito de Agente – consiste em uma entidade autônoma atômica, capaz de realizar alguma função útil. Esta competência funcional é representada pelos serviços do agente, que são análogos às operações de um objeto. A autonomia significa que as ações do agente não são somente determinadas por eventos externos ou interações, mas também por sua própria motivação. Esta motivação é referenciada como um atributo chamado propósito, que influencia em como um agente reagirá a um pedido para a execução de um determinado serviço e em como este será provido. Neste âmbito, os agentes de software e os agentes humanos são especializações do conceito de Agente.

Conceito de Organização – consiste em um grupo de agentes que trabalham em conjunto para o alcance de um propósito comum. Trata-se de uma entidade virtual no sentido de que não possui uma entidade computacional individual que a codifique. Os agentes que a constitui provêem os serviços da Organização e alcançam os seus objetivos de forma coletiva. Estes agentes conectam-se entre si por meio de relacionamentos organizacionais, como os existentes em uma organização humana, tal como o de superior-subordinado, e por meio de mecanismos de coordenação, expressos pelas interações entre agentes.

Conceito de Papel – um Papel descreve as características externas de um agente em um contexto particular. A distinção entre os conceitos de Papel e Agente é análoga a existente entre a Classe de um objeto e sua Interface. Um

agente pode desempenhar diferentes papéis e vários agentes podem desempenhar um mesmo papel, visto que este descreve as características externas de um agente em um contexto particular.

Conceito de Recurso – representa entidades que não são autônomas, como bases de dados e programas externos utilizados pelos agentes. A metodologia MESSAGE considera adequados os conceitos padronizados da orientação a objetos para a modelagem de recursos.

Descrição dos Conceitos da Categoria Atividade

Conceito de Tarefa – consiste em uma unidade de atividade, que possui um único executor primário e estados definidos por pré-condições e pós-condições. Para uma tarefa ser realizada, suas pré-condições devem ser válidas e pode-se esperar, após o término de mesma, que as pós-condições sejam satisfeitas.

Conceitos de Interação e Protocolo de Interação – o conceito de Interação está fortemente relacionado ao introduzido pela metodologia Gaia (ZAMBONELLI et al., 2003). Uma interação possui um ou mais participantes e um propósito que estes pretendem atingir em conjunto. Este propósito geralmente consiste na obtenção de uma visão consistente de algum aspecto do domínio do problema, no estabelecimento dos termos de um serviço ou na troca de resultados de um ou mais serviços. Um protocolo de interação define um padrão para a troca de mensagens existente em uma interação.

Descrição do Conceito da Categoria Estado Mental

Conceito de Objetivo – este conceito associa o agente a um estado de forma que este pretende causar o estado relacionado ao objetivo. Alguns objetivos são derivados do propósito do agente e são intrínsecos a sua identidade, sendo persistentes por toda a vida útil deste.

Descrição de Outros Conceitos Importantes da MESSAGE

Conceito de Mensagem – representa um objeto comunicado de um agente a outro. A transmissão de uma mensagem tem um tempo finito de duração e requer que uma ação seja executada entre um agente emissor e um agente receptor. Uma Mensagem é composta por atributos que especificam o emissor, o receptor, a categorização conforme o plano do emissor (que categoriza a Mensagem em termos da intenção do emissor) e o conteúdo (uma Entidade de Informação).

Conceito de Entidade de Informação – consiste simplesmente em um objeto que encapsula um conjunto de informações.

5.4.2 Visões do Modelo de Análise

O modelo de análise consiste em uma complexa rede de classes e instâncias inter-relacionadas, derivadas dos conceitos da metodologia. A metodologia MESSAGE (CAIRE et al., 2002) propõe que este modelo seja estruturado em visões a fim de oferecer uma técnica que possibilite ao analista focalizar nos conceitos da metodologia e, conseqüentemente, descrever os vários aspectos de um sistema.

A metodologia propõe a utilização de um conjunto de visões, dentre as quais se destacam: a visão Organizacional, a visão de Objetivo e a visão de Agente. Os autores da MESSAGE recomendam que se assegure a consistência entre as diferentes visões para que a técnica de estruturação em visões seja bem sucedida.

A Visão Organizacional representa as entidades concretas no sistema e no seu ambiente, e os relacionamentos entre estas entidades, como os relacionamentos de agregação e de conhecimento. O relacionamento de conhecimento indica que há pelo menos uma interação entre as entidades. As entidades podem ser agentes, organizações, papéis ou recursos.

A Visão de Objetivo representa um objetivo e as dependências entre seus sub-objetivos. É possível representar um objetivo como a composição de um conjunto de sub-objetivos, em um nível inferior do diagrama de representação desta visão.

A Visão de Agente descreve, por meio de um esquema de agente, as seguintes características de cada agente: os objetivos, os eventos percebidos, os recursos controlados, as regras de comportamento e as tarefas que o agente sabe realizar.

Na metodologia MESSAGE considera-se uma abordagem top-down para a definição da estrutura do sistema multi-agente. Para verificar se os comportamentos destes agentes não apresentam desvios prejudiciais ao sistema como um todo, deve-se fazer uma análise bottom-up. Assim, o comportamento do sistema será avaliado em relação aos seus requisitos, de forma que, caso aspectos indesejáveis sejam encontrados, a análise top-down deva ser revisada.

O Capítulo 6, a seguir, apresenta a arquitetura multi-agente de balanceamento de carga proposta neste trabalho de pesquisa.

6 ARQUITETURA MABal

Este capítulo apresenta a arquitetura multi-agente distribuída proposta neste trabalho de pesquisa para o problema de balanceamento de carga. A arquitetura de software proposta é denominada arquitetura *M*ulti-*A*gente Distribuída de *Bal*anceamento de Carga para Aplicações de Objetos Distribuídos, referenciada pela sigla arquitetura MABal.

Evidencia-se que a arquitetura MABal apresenta as seguintes características: (i) atua em sistemas que possuem aplicações de objetos distribuídos, efetuando um processo de realocação dos objetos destas aplicações entre os nós do sistema a fim de evitar sobrecargas nos nós; (ii) provê um serviço de balanceamento de carga distribuído e de controle descentralizado que atua de modo preventivo, ou seja, este serviço impede a execução de objetos em nós sobrecarregados ao invés de posteriormente balancear a carga do sistema por meio da realocação de objetos que já encontram-se em execução; e (iii) utiliza o conceito de agente como o elemento principal de seu projeto arquitetural.

Neste capítulo apresentam-se a estrutura e a organização de agentes da arquitetura MABal e, em seqüência, apresentam-se os modelos da metodologia MESSAGE (CAIRE et al., 2002) construídos para o projeto orientado a agentes da arquitetura.

6.1 Estrutura e Organização de Agentes da Arquitetura MABal

A arquitetura MABal considera o conceito de agente como o elemento principal de seu projeto arquitetural. A adoção do conceito de agente embasou-se no conjunto de diretrizes proposto pelos participantes do projeto P907 do EURESCOM detalhados no Capítulo 4. Em síntese, as principais razões para se conceber a arquitetura MABal utilizando o conceito de agente são as

capacidades que os agentes possuem de atuação autônoma e distribuída, e de se adaptarem flexivelmente a mudanças de requisitos ou no ambiente (MILGROM, 2001).

Os agentes da arquitetura são considerados agentes especialistas, pois têm como objetivo o provimento de serviços específicos. Além disso, apresentam relacionamentos de dependência funcional entre si. Um relacionamento de dependência funcional caracteriza-se pela necessidade de um agente em se beneficiar do provimento de uma funcionalidade (serviço) de outro agente. A arquitetura MABal apresenta uma estrutura hierárquica distribuída baseada nos relacionamentos de dependência funcional dos agentes que a constitui.

A estrutura hierárquica distribuída da arquitetura MABal é constituída pelo grupo de agentes a seguir, que atuam em cada nó do sistema:

- agentes responsáveis pela migração e pela replicação de objetos de aplicações distribuídas a fim de prover o serviço de balanceamento de carga distribuído do sistema;

- agentes que cooperam com os agentes responsáveis pela migração e pela replicação de objetos no sentido de gerar informações essenciais ao processo de balanceamento de carga, como: (i) a classificação dos níveis (estados) de carga dos nós do sistema a partir da coleta dos índices de uso de CPU e uso de memória destes nós; e (ii) a decisão de para quais nós do sistema migrar os objetos servidores, ou seja, os objetos que atendem os serviços requisitados, a fim de evitar sobrecargas no sistema;

- agentes gerenciadores, que coordenam as interações entre agentes da arquitetura MABal. Existem agentes gerenciadores de nós servidores e agentes gerenciadores de nós clientes. Estes dois tipos de agentes encontram-se presentes em cada nó do sistema, uma vez que qualquer

nó do sistema pode exercer os dois papéis, ou seja, atuar como servidor e como cliente.

O nó servidor consiste no nó do sistema que, em um instante de tempo considerado, contém o objeto servidor da aplicação distribuída. O objeto servidor consiste no objeto que atende a uma invocação de serviço realizada por outro objeto da aplicação. Em outras palavras, o objeto servidor é aquele que provê o serviço requisitado e o nó servidor é o nó onde este objeto se localiza.

Por sua vez, o nó cliente consiste no nó do sistema que, em um instante de tempo considerado, contém o objeto cliente. O objeto cliente consiste no objeto da aplicação que efetua a invocação de um serviço do objeto servidor.

6.2 Modelos da Arquitetura MABal

A arquitetura MABal foi modelada adotando-se o processo de desenvolvimento de sistemas orientado a agentes proposto pela metodologia MESSAGE (CAIRE et al., 2002). Esta metodologia sugere que os modelos de análise de um sistema sejam produzidos por meio de refinamentos sucessivos. Segundo a metodologia MESSAGE, o processo de modelagem inicia-se com a construção das visões Organizacional e de Objetivos, sendo que ambas provêem a base para a construção da Visão de Agentes. As seções subseqüentes apresentam os modelos de análise construídos para a arquitetura MABal.

6.2.1 Visão Organizacional

A Visão Organizacional considera o sistema a ser desenvolvido como uma caixa preta, focalizando os relacionamentos de interação do sistema com as entidades do ambiente no qual o sistema está inserido.

A Figura 6.1 apresenta o diagrama de Relacionamentos de Interação da Visão Organizacional para o serviço de balanceamento de carga distribuído provido pela arquitetura MABal.

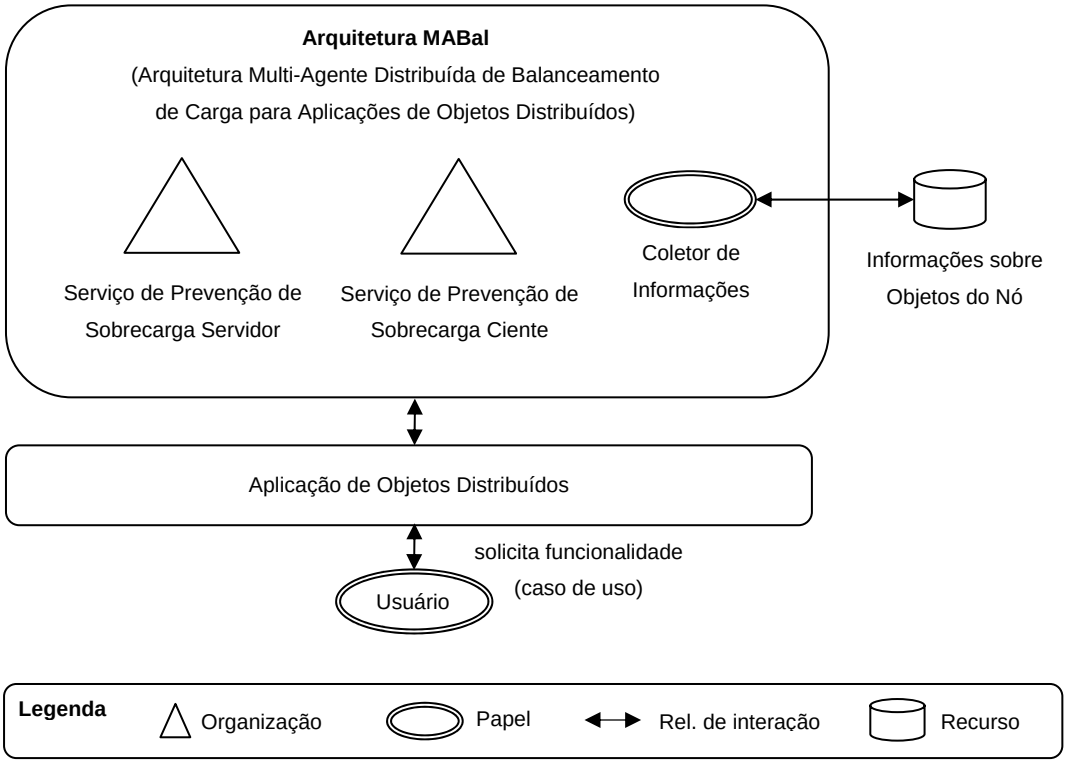


Figura 6.1 – Diagrama de relacionamentos de interação da visão organizacional

Como ilustra o diagrama da Figura 6.1, o Usuário solicita à Aplicação de Objetos Distribuídos o provimento de uma funcionalidade. A funcionalidade requisitada é atendida pela invocação de uma seqüência de objetos distribuídos, que provêem serviços específicos e, em conjunto, provêem a funcionalidade.

A arquitetura MABal realiza o balanceamento de carga do sistema a cada invocação de serviço de um objeto da aplicação distribuída que colabora no provimento da funcionalidade requisitada pelo Usuário. Assim, a arquitetura MABal atua a cada invocação de serviço de um objeto (objeto servidor)

selecionando o nó mais apropriado para executar este objeto a fim de evitar sobrecarga no nó servidor.

Ressalta-se que um mesmo nó do sistema pode atuar como nó servidor ou como nó cliente, dependendo se este possui um objeto da aplicação que atende a um serviço requisitado ou se possui um objeto que invoca um serviço. Em decorrência desta possibilidade de atuação de um mesmo nó como nó servidor e como nó cliente, distribui-se a arquitetura MABal, com sua estrutura completa, em cada nó do sistema. Assim, todos os nós da rede possuem autonomia para realizar o processo de busca por um nó mais adequado para prover o serviço requisitado. Esta autonomia confere à arquitetura MABal a característica de descentralização do controle do processo de balanceamento de carga.

A estrutura da arquitetura MABal é constituída por dois serviços e por um papel, conforme ilustra o diagrama da Figura 6.1.

O Serviço de Prevenção de Sobrecarga Servidor e o Serviço de Prevenção de Sobrecarga Cliente atuam para o caso de um nó exercer a função de nó servidor ou de nó cliente, respectivamente:

- Serviço de Prevenção de Sobrecarga Servidor, responsável por garantir que o nó servidor, que contém o objeto provedor do serviço requisitado (objeto servidor), atenda à invocação de serviço somente quando possuir nível de carga ocioso. Assim, este Serviço atua de forma preventiva contra sobrecargas nos nós servidores, impedindo que estes atendam às invocações de serviços quando apresentarem níveis de carga elevados;

- Serviço de Prevenção de Sobrecarga Cliente, responsável por atender uma invocação de serviço no próprio nó cliente ou em outro nó do sistema ao invés de ter a invocação atendida no nó servidor. Este Serviço atua após a verificação, pelo Serviço de Prevenção de Sobrecarga Servidor, de

sobrecarga no nó servidor, o que inviabiliza que o nó servidor atenda ao serviço requisitado. Assim, o Serviço de Prevenção de Sobrecarga Cliente atua migrando ou replicando o objeto servidor para o nó cliente, caso este apresente um nível de carga apropriado, a fim de que o nó cliente atenda ao serviço requisitado. No entanto, caso o nó cliente esteja sobrecarregado, o Serviço de Prevenção de Sobrecarga Cliente: (i) seleciona um nó do sistema que seja adequado para atender ao serviço requisitado, conforme uma política de Seleção de Terceiro Nó; e (ii) ativa o Serviço de Prevenção de Sobrecarga Cliente do nó selecionado a fim de que o objeto servidor seja migrado ou replicado para o nó selecionado, que passará a atender o serviço requisitado.

Por sua vez, o papel Coletor de Informações efetua buscas por informações específicas em uma base de dados, existente em cada nó do sistema, que contém duas tabelas de dados: Tabela de Objetos e Tabela de Relacionamentos.

A Tabela de Objetos contém todos os objetos fisicamente localizados em um nó e os estados destes objetos. O estado de um objeto pode ser: (i) “em uso”, significa que o objeto está atendendo solicitações de serviços realizadas por outros objetos da aplicação; ou (ii) “não em uso”, significa que o objeto não está atendendo nenhuma solicitação de serviço.

A Tabela de Relacionamentos registra as invocações entre objetos para a realização de cada caso de uso da aplicação. Assim, nesta tabela consta cada relacionamento existente entre dois objetos da aplicação, ou seja, um objeto (objeto cliente) que invoca um serviço de outro objeto (objeto servidor) para a realização de um determinado caso de uso.

6.2.2 Visão de Objetivos

O objetivo principal da arquitetura MABal consiste em evitar que a execução de aplicações de objetos distribuídos sobrecarregue os nós de um sistema considerado. Em outras palavras, o objetivo principal da arquitetura MABal consiste em balancear a carga do sistema no momento prévio a um objeto de uma aplicação distribuída (objeto servidor) atender a uma nova invocação de serviço a fim de evitar sobrecarga no nó onde este objeto se localiza (nó servidor). Este objetivo macro é decomposto em subobjetivos. O diagrama da Figura 6.2, um diagrama da Visão de Objetivos chamado de diagrama de Decomposição de Objetivos, ilustra a decomposição de objetivos para o objetivo principal da arquitetura multi-agente distribuída de balanceamento de carga proposta.

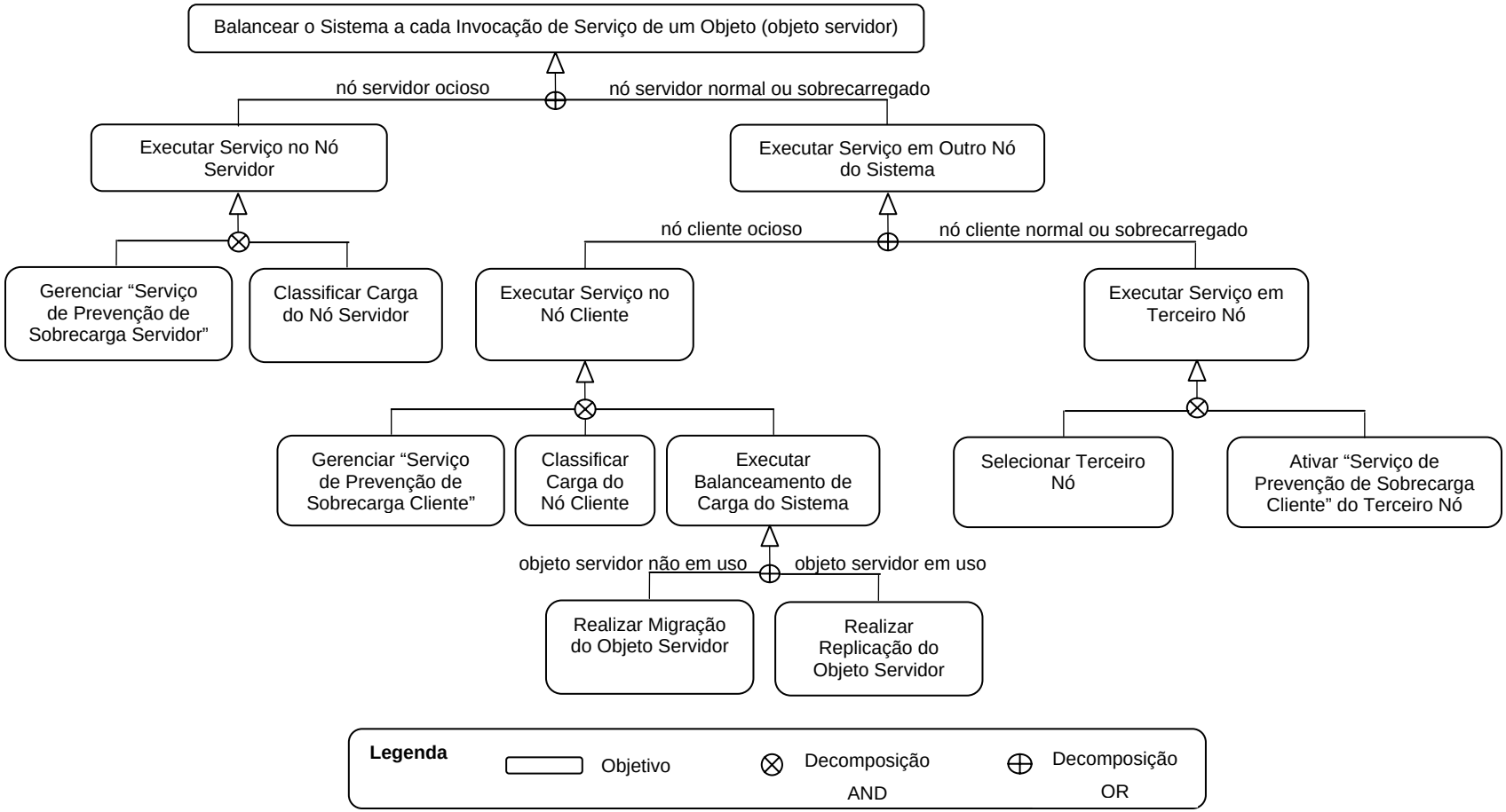


Figura 6.2 – Diagrama de decomposição de objetivos da Visão de Objetivos

Conforme ilustra o diagrama da Figura 6.2, o objetivo principal da arquitetura MABal “Balancear o Sistema a cada Invocação de Serviço de um Objeto” é decomposto nos subobjetivos: “Executar Serviço no Nó Servidor” ou “Executar Serviço em Outro Nó do Sistema”, sendo satisfeito quando um destes subobjetivos for satisfeito.

Por sua vez, estes subobjetivos são satisfeitos quando:

- Subobjetivo “Executar Serviço no Nó Servidor”: o serviço requisitado pelo objeto cliente é atendido pelo objeto servidor, que provê o serviço requisitado, no nó em que o objeto servidor se encontra fisicamente localizado (nó servidor);

- Subobjetivo “Executar Serviço em Outro Nó do Sistema”: o serviço requisitado pelo objeto cliente não é atendido pelo objeto servidor no nó em que o objeto servidor se encontra localizado, mas é atendido pelo objeto servidor em outro nó do sistema. Para isso o objeto servidor deve ser realocado para um nó do sistema, previamente escolhido, onde o serviço requisitado será executado pelo objeto servidor e provido ao objeto cliente.

Objetivo “Executar Serviço no Nó Servidor”

O alcance do objetivo “Executar Serviço no Nó Servidor” é de responsabilidade do Serviço de Prevenção de Sobrecarga Servidor da arquitetura MABal, que atua em um determinado nó quando este desempenha o papel de nó servidor. Como ilustra a Figura 6.3, este objetivo é decomposto nos subobjetivos “Gerenciar Serviço de Prevenção de Sobrecarga Servidor” e “Classificar Carga do Nó Servidor”, sendo satisfeito quando ambos os subobjetivos forem satisfeitos.

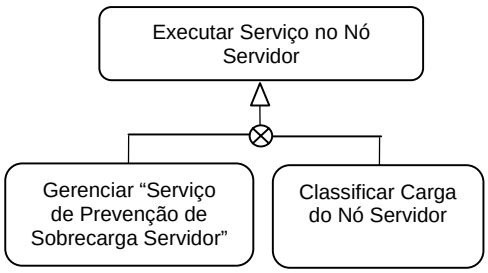


Figura 6.3 – Decomposição de objetivos para o objetivo “Executar Serviço no Nó Servidor”

Por sua vez, os subobjetivos que compõem o objetivo “Executar Serviço no Nó Servidor” são satisfeitos quando:

- Subobjetivo “Gerenciar Serviço de Prevenção de Sobrecarga Servidor”: o nó servidor, que contém o objeto provedor do serviço requisitado (objeto servidor) pelo objeto cliente, apresentar-se em constante monitoração por pedidos de atuação do Serviço de Prevenção de Sobrecarga Servidor e, ao receber um pedido, solicitar que o nível de carga do nó servidor seja classificado e proceder conforme a classificação do nível de carga do nó: (i) caso o nó servidor (que contém o objeto servidor) esteja ocioso, conclui-se que o nó servidor possui nível de carga adequado para executar o serviço requisitado; ou (ii) caso o nó servidor esteja sobrecarregado, ativar o Serviço de Prevenção de Sobrecarga Cliente do nó cliente com o objetivo de que o nó cliente passe a executar o serviço requisitado;

- Subobjetivo “Classificar Carga do Nó Servidor”: o nível de carga do nó servidor for definido após a coleta dos índices de uso de CPU e de memória deste nó e da classificação dos valores coletados.

Objetivo “Executar Serviço em Outro Nó do Sistema”

O alcance do objetivo “Executar Serviço em Outro Nó do Sistema” é de responsabilidade do Serviço de Prevenção de Sobrecarga Cliente da arquitetura MABal, que atua em um determinado nó quando: (i) este desempenha o papel de nó cliente; ou (ii) este não desempenha o papel de nó cliente e nem de nó servidor, ou seja, consiste em um nó do sistema que não realizou a invocação de serviço considerada e não contém o objeto capaz de prover o serviço requisitado.

Conforme ilustra a Figura 6.4, o objetivo “Executar Serviço em Outro Nó do Sistema” é decomposto nos subobjetivos “Executar Serviço no Nó Cliente” ou “Executar Serviço em Terceiro Nó”, sendo satisfeito quando um destes subobjetivos for satisfeito.

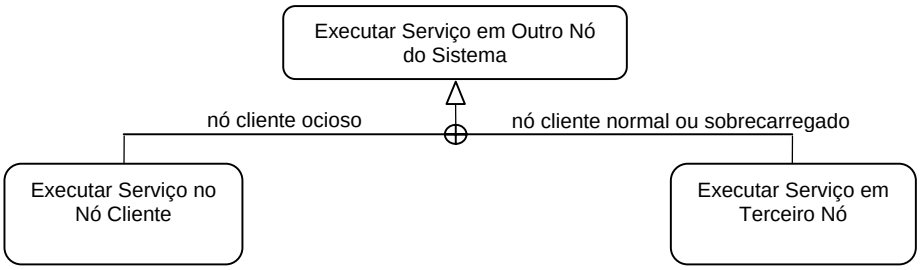


Figura 6.4 – Decomposição de objetivos para o objetivo “Executar Serviço em Outro Nó do Sistema”

Por sua vez, os subobjetivos que compõem o objetivo “Executar Serviço em Outro Nó do Sistema” são satisfeitos quando:

- Subobjetivo “Executar Serviço no Nó Cliente”: o objeto servidor atende ao serviço requisitado no nó cliente, ou seja, o objeto servidor executa o serviço requisitado no mesmo nó que invocou o serviço. Isto implica no objeto

servidor, localizado fisicamente no nó servidor, em ser realocado para o nó cliente a fim de prover (executar) o serviço requisitado neste nó;

- Subobjetivo “Executar Serviço em Terceiro Nó”: o objeto servidor atende ao serviço requisitado em outro nó do sistema, previamente selecionado, que não sejam os nós cliente e servidor. Isto implica no objeto servidor, localizado fisicamente no nó servidor, em ser realocado para este terceiro nó selecionado a fim de prover (executar) o serviço requisitado neste nó. A seleção deste terceiro nó é guiada por uma política que promove a seleção do nó do sistema mais apropriado para receber o objeto servidor e prover o serviço requisitado.

Objetivo “Executar Serviço no Nó Cliente”

Conforme ilustra a Figura 6.5, o objetivo “Executar Serviço no Nó Cliente” é decomposto nos subobjetivos: “Gerenciar Serviço de Prevenção de Sobrecarga Cliente”, “Classificar Carga do Nó Cliente” e “Executar Balanceamento de Carga do Sistema”, sendo satisfeito quando estes três subobjetivos forem satisfeitos.

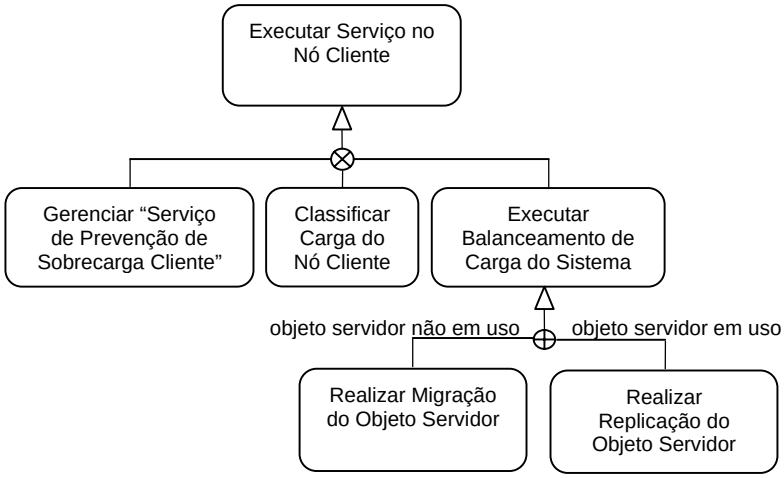


Figura 6.5 – Decomposição de objetivos para o objetivo “Executar Serviço no Nó Cliente”

Por sua vez, estes subobjetivos são satisfeitos quando:

- Subobjetivo “Gerenciar Serviço de Prevenção de Sobrecarga Cliente”: o nó cliente ou qualquer outro nó do sistema, exceto o nó servidor, apresentar-se em constante monitoração por pedidos de atuação do Serviço de Prevenção de Sobrecarga Cliente e, ao receber um pedido, solicitar que o nível de carga do nó seja classificado e proceder conforme a classificação do nível de carga do nó: (i) caso o nó esteja ocioso, ativar o processo de balanceamento de carga a fim de realocar o objeto servidor para este nó; ou (ii) caso o nó esteja sobrecarregado, selecionar outro nó do sistema e ativar o Serviço de Prevenção de Sobrecarga Cliente do nó selecionado com o objetivo de que o nó selecionado efetue o processo de balanceamento de carga e, portanto, passe a conter o objeto servidor e possa executar o serviço requisitado;

- Subobjetivo “Classificar Carga do Nó Cliente”: o nível de carga do nó cliente ou de qualquer outro nó do sistema, exceto o nó servidor, for definido após a coleta dos índices de uso de CPU e de memória do nó considerado e da classificação dos valores coletados;

- Subobjetivo “Executar Balanceamento de Carga do Sistema”: o objeto servidor for realocado para o nó cliente ou para outro nó do sistema a fim de executar o serviço requisitado neste nó, que apresenta nível de carga mais adequado para atender à invocação de serviço em relação ao nó servidor. Ressalta-se que o processo de balanceamento de carga somente se faz necessário quando o nó servidor apresentar nível de carga normal ou sobrecarregado e, portanto, não ser recomendável para o equilíbrio de carga do sistema que este nó atenda a uma nova invocação de serviço. A realocação do objeto servidor, objeto que provê o serviço requisitado, consiste em migrá-lo ou replicá-lo para um nó do sistema com nível de carga

ocioso a fim de evitar sobrecarga no nó sevidor e, com isso, obter um maior equilíbrio de carga no sistema.

Objetivo “Executar Serviço em Terceiro Nó”

Conforme ilustra a Figura 6.6, o objetivo “Executar Serviço em Terceiro Nó” é decomposto nos subobjetivos: “Classificar Cargas dos Nós do Sistema”, “Selecionar Terceiro Nó” e “Ativar Serviço de Prevenção de Sobrecarga Cliente do Terceiro Nó”, sendo satisfeito quando estes três subobjetivos forem satisfeitos.

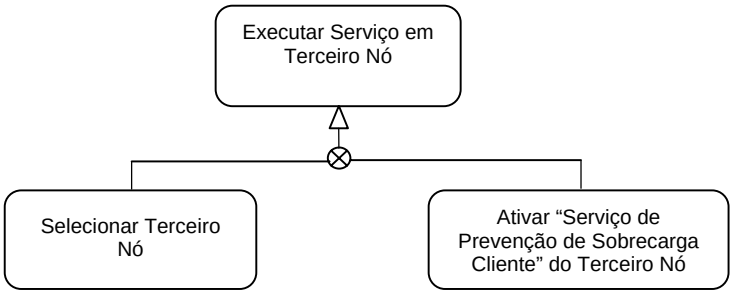


Figura 6.6 – Decomposição de objetivos para o objetivo “Executar Serviço em Terceiro Nó”

Por sua vez, estes subobjetivos são satisfeitos quando:

- Subobjetivo “Selecionar Terceiro Nó”: um terceiro nó for escolhido, aplicando-se uma política para esta seleção, a fim de que o objeto servidor (que provê o serviço requisitado) seja realocado para o nó escolhido, considerado o nó mais adequado para receber o objeto servidor;

- Subobjetivo “Ativar Serviço de Prevenção de Sobrecarga Cliente do Terceiro Nó”: o “Serviço de Prevenção de Sobrecarga Cliente” do nó previamente escolhido para receber o objeto servidor for ativado. O objetivo de ativar este Serviço consiste em encaminhar ao nó considerado o papel de efetuar

o balanceamento de carga, ou seja, de migrar ou replicar o objeto servidor para o nó, a fim de que o serviço invocado seja executado neste nó do sistema.

6.2.3 Visão de Agentes

A Visão de Agentes relaciona-se diretamente com as Visões Organizacional e de Objetivos. Esta visão considera as organizações, definidas na Visão Organizacional, e a decomposição de objetivos para estas organizações, definida na Visão de Objetivos.

A arquitetura MABal é constituída por duas organizações, a organização Serviço de Prevenção de Sobrecarga Servidor e a organização Serviço de Prevenção de Sobrecarga Cliente. Estas duas organizações, em conjunto, são responsáveis por atingir o objetivo principal da arquitetura MABal.

Em particular, a organização Serviço de Prevenção de Sobrecarga Servidor é responsável por atingir o subobjetivo “Executar Serviço no Nó Servidor”, enquanto a organização Serviço de Prevenção de Sobrecarga Cliente é responsável por atingir o subobjetivo “Executar Serviço em Outro Nó do Sistema”. A Visão de Agentes da arquitetura MABal define como estes subobjetivos podem ser alcançados por meio da delegação de agentes responsáveis pelo alcance destes subobjetivos.

A Figura 6.7 apresenta o diagrama da Visão de Agentes para a organização Serviço de Prevenção de Sobrecarga Servidor da arquitetura MABal, no qual os subobjetivos obtidos da decomposição do objetivo principal desta organização são atribuídos para agentes que constituem a arquitetura MABal. Por sua vez, a Figura 6.8 apresenta o diagrama da Visão de Agentes para a organização Serviço de

Prevenção de Sobrecarga Cliente da arquitetura MABal, no qual os subobjetivos obtidos da decomposição do objetivo principal desta organização também são atribuídos para agentes que constituem a arquitetura MABal.

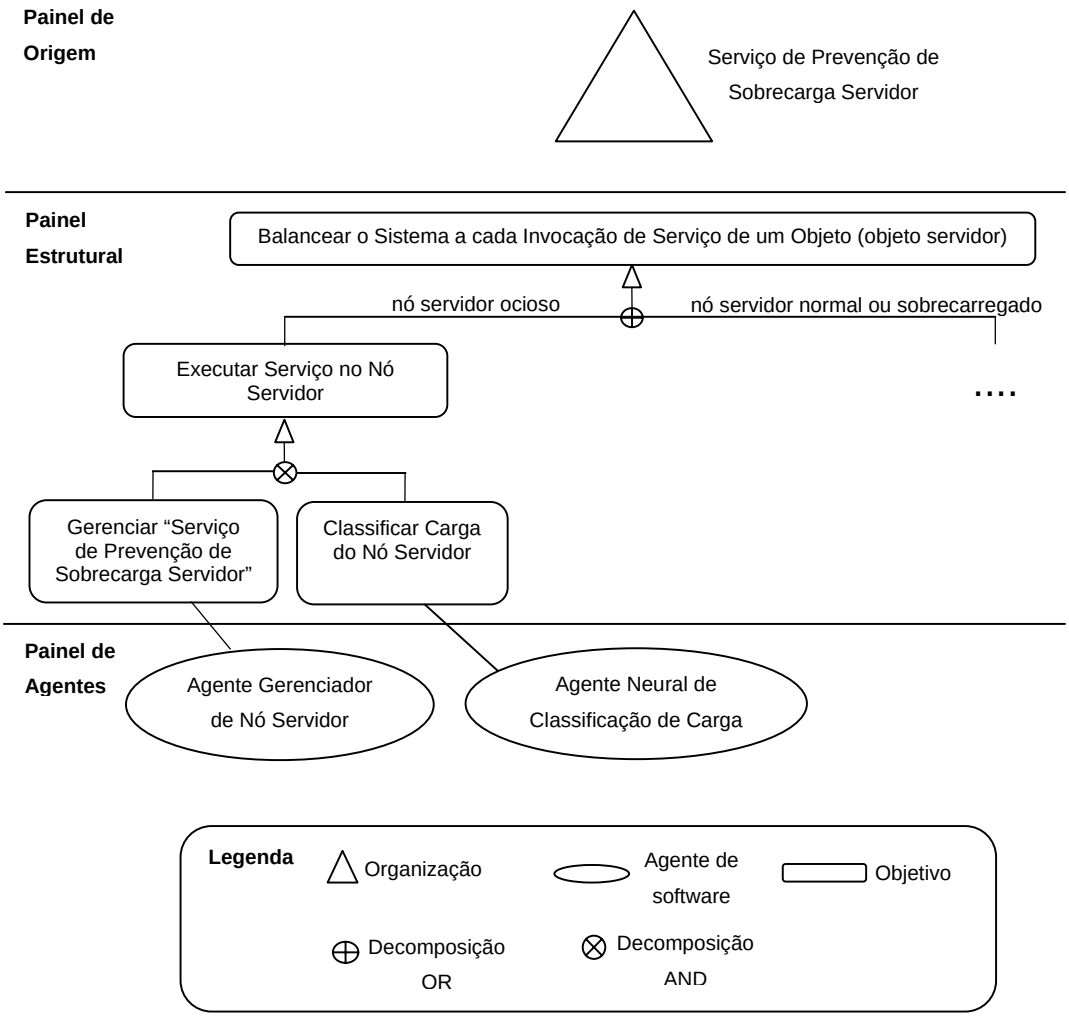


Figura 6.7 – Diagrama da Visão de Agentes para a organização "Serviço de Prevenção de Sobrecarga Servidor"

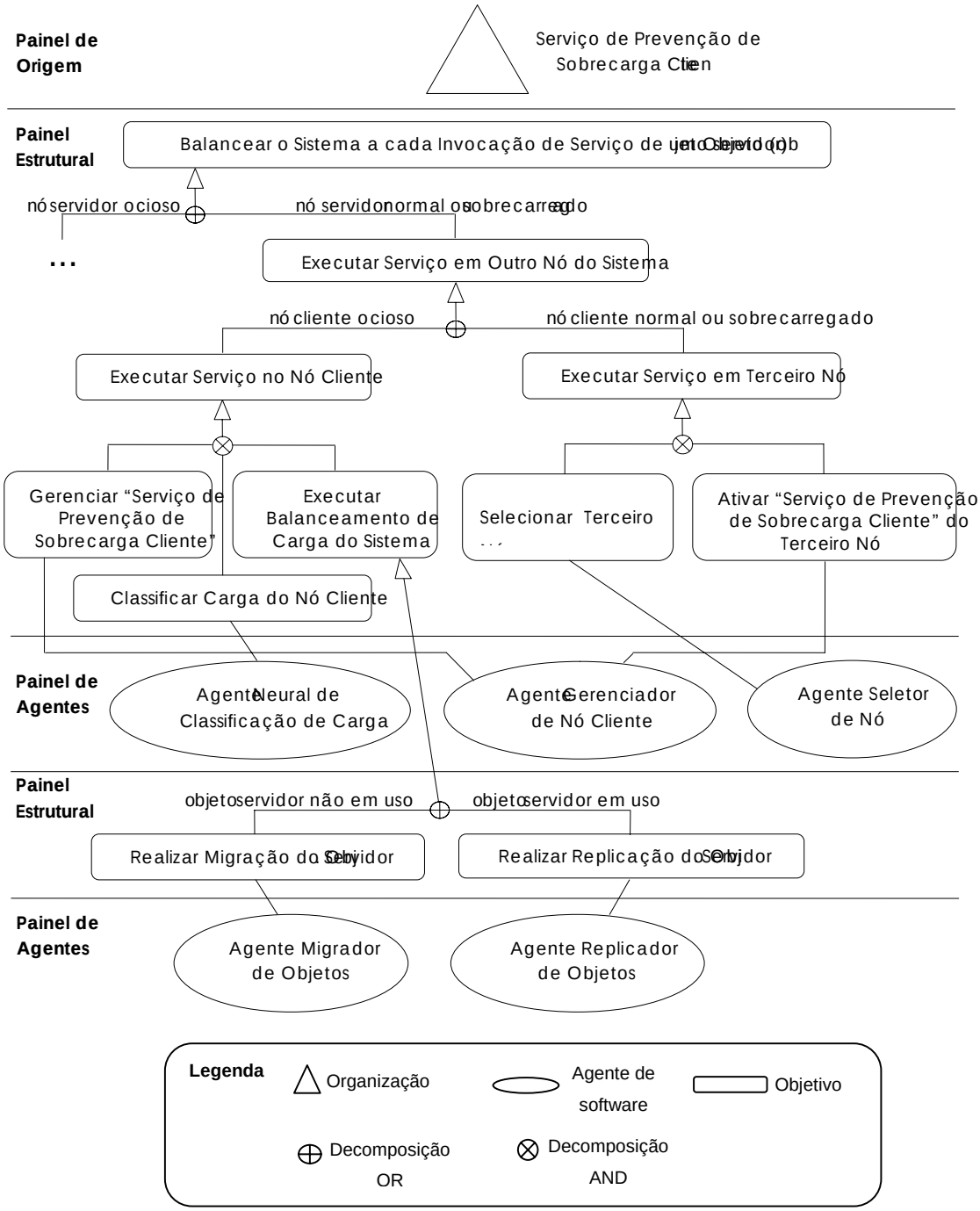


Figura 6.8 – Diagrama da Visão de Agentes para a organização "Serviço de Prevenção de Sobrecarga Cliente"

Como pode-se observar nos diagramas das Figuras 6.7 e 6.8, identificou-se os agentes que constituem a arquitetura MABal por meio da necessidade de atingir os objetivos previamente identificados. Assim, foram levantados agentes para a arquitetura MABal que realizam tarefas para o alcance de cada um dos objetivos.

Nas próximas seções detalham-se os comportamentos dos agentes identificados, que constituem a arquitetura MABal, descrevendo as suas características principais.

6.2.4 Agente Neural de Classificação de Carga

O agente Neural de Classificação de Carga atua no Serviço de Prevenção de Sobrecarga Servidor assim como no Serviço de Prevenção de Sobrecarga Cliente da arquitetura MABal. Estes dois Serviços encontram-se presentes em cada nó do sistema devido ao fato de um mesmo nó do sistema poder exercer os papéis de nó servidor ou de nó cliente, dependendo se o nó contém um objeto capaz de prover um serviço requisitado (objeto servidor) ou se contém um objeto que invoca um serviço de outro objeto (objeto cliente), respectivamente.

Este agente é responsável por determinar o estado de carga dos nós do sistema por meio da coleta de dois índices de carga, uso de CPU e uso de memória, e da classificação de um determinado nó do sistema em cinco estados possíveis:

- Nível 1: muito ocioso;
- Nível 2: pouco ocioso;
- Nível 3: normal;
- Nível 4: pouco carregado;
- Nível 5: muito carregado.

Assim, o conjunto {1, 2, 3, 4, 5} representa os níveis de carga possíveis para cada nó do sistema. Para classificar o nível de carga de um nó, este agente utiliza uma rede neural MLP (*Multiple Layer Perceptron*) (HAYKIN, 2001).

A Figura 6.9 ilustra a definição do agente Neural de Classificação de Carga, que inclui a descrição dos eventos que este agente sente, do serviço que oferece (retorno do agente) e do seu comportamento (fluxo de tarefas do agente).

AGENTE-NEURAL-DE-CLASSIFICAÇÃO-DE-CARGA

(**eventos:** recebimento de mensagem do agente Gerenciador de Nó Servidor, do agente Gerenciador de Nó Cliente ou do agente Seletor de Nó efetuando um pedido de atuação do agente Neural de Classificação de Carga

)

retorna: informação de nível de carga do nó considerado

- Coleta os índices de CPU e de memória do respectivo nó, que representam as medidas da carga deste nó;
- Ativa a rede neural MLP (*Multiple Layer Perceptron*), fornecendo os índices de CPU e de memória como entrada para a rede;
- A rede neural MLP classifica os dados de entrada em um dos seguintes níveis de carga: nível 1 (muito ocioso), nível 2 (pouco ocioso), nível 3 (normal), nível 4 (pouco carregado) e nível 5 (muito carregado). Um nível de carga representa o estado de carga do respectivo nó no instante de tempo considerado.

Figura 6.9 – Definição do agente Neural de Classificação de Carga

Para implementar o agente Neural de Classificação de Carga, primeiramente utilizou-se engenharia de conhecimento para associar o uso de CPU e o uso de memória aos cinco níveis de carga definidos. A Tabela 6.1 apresenta estas associações.

Tabela 6.1 – Associações de uso de CPU e uso de memória com os níveis de carga

	MEMÓRIA				
CPU	0 – 0.19	0.2 – 0.39	0.4 – 0.59	0.6 – 0.79	0.8 – 1.0
0 – 0.19	Nível 1	Nível 2	Nível 3	Nível 4	Nível 5
0.2 – 0.39	Nível 2	Nível 2	Nível 3	Nível 4	Nível 5
0.4 – 0.59	Nível 3	Nível 3	Nível 3	Nível 4	Nível 5
0.6 – 0.79	Nível 4	Nível 4	Nível 4	Nível 4	Nível 5
0.8 – 1.0	Nível 5	Nível 5	Nível 5	Nível 5	Nível 5

O agente Neural de Classificação de Carga utiliza uma rede neural MLP (*Multiple Layer Perceptron*), que foi treinada de forma supervisionada pelo algoritmo de retropropagação do erro (HAYKIN, 2001). A arquitetura da rede é formada por 02 entradas, 37 neurônios na camada escondida, sendo a função de ativação a tangente hiperbólica, e 03 neurônios na camada de saída com a função de ativação logística sigmoidal.

O treinamento foi realizado com 10201 vetores de entrada, a rede atingiu um erro de 0.000008 após 30000 épocas e na fase de ativação a rede generalizou.

6.2.5 Agente Gerenciador de Nó Servidor

O agente Gerenciador de Nó Servidor constitui o Serviço de Prevenção de Sobrecarga Servidor da arquitetura MABal. Este agente é responsável por gerenciar este Serviço no nó servidor, ou seja, no nó do sistema que contém o objeto servidor no instante de tempo considerado.

A Figura 6.10 ilustra a definição do agente Gerenciador de Nó Servidor, que inclui a descrição dos eventos que este agente sente, do serviço que oferece (retorno do agente) e do seu comportamento (fluxo de tarefas do agente).

AGENTE-GERENCIADOR-DE-NÓ-SERVIDOR

(**eventos:** recebimento de mensagem do objeto cliente da aplicação efetuando um pedido de atuação do agente Gerenciador de Nó Servidor

)

retorna: mensagem de ativação do Serviço de Prevenção de Sobrecarga Cliente do nó cliente, contendo o estado do objeto servidor

- Monitora novo pedido de atuação;
- Ao receber um pedido de atuação:
  - envia mensagem ao agente Neural de Classificação de Carga solicitando que classifique o nível de carga do respectivo nó servidor;
  - analisa a informação do nível de carga do nó servidor:
    - caso o nó servidor esteja ocioso, atualiza o estado do objeto servidor na Tabela de Objetos do nó servidor para “em uso” e encerra a atuação do agente, pois o nível de carga do nó servidor está adequado para a futura execução do serviço requisitado;
    - caso contrário (nó servidor normal ou sobrecarregado), verifica o estado do objeto servidor na Tabela de Objetos do nó servidor e envia uma mensagem ao Serviço de Prevenção de Sobrecarga Cliente do nó cliente para ativar este Serviço da arquitetura MABal, contendo o estado do objeto servidor. O estado de um objeto servidor, em um determinado instante de tempo, pode ser: (i) “em uso”, o objeto sevidor está atendendo solicitações de serviços realizadas por outros objetos da aplicação; ou (ii) “não em uso”, o objeto servidor não está atendendo nenhuma solicitação de serviço.

Figura 6.10 – Definição do agente Gerenciador de Nó Servidor

Ressalta-se que o agente Gerenciador de Nó Servidor atua como gerenciador do Serviço de Prevenção de Sobrecarga Servidor e, portanto, tem como objetivo evitar sobrecarga ao nó servidor.

Ao detectar que o nó servidor possui nível de carga ocioso e, portanto, possui estado de carga adequado para a execução do serviço requisitado, o agente

Gerenciador de Nó Servidor altera o estado do objeto servidor para “em uso”. O estado “em uso” sinaliza que este objeto estará executando o serviço requisitado. O estado de carga de um objeto influencia diretamente na escolha da decisão de balanceamento de carga a ser realizada em um sistema: objetos “em uso” são replicados e objetos “não em uso” são migrados. Portanto, a alteração do estado de carga do objeto servidor para “em uso” direciona a arquitetura MABal, em uma invocação futura de um serviço deste mesmo objeto, a criar uma réplica deste objeto em outro nó do sistema para atender à nova invocação, caso o nó servidor não apresente nível de carga adequado para executar a nova requisição de serviço.

O estado do objeto servidor é armazenado na Tabela de Objetos, que contém todos os objetos fisicamente localizados no nó servidor e os estados destes objetos. Ressalta-se que uma Tabela de Objetos está presente em cada nó do sistema para armazenar os objetos que encontram-se localizados em um nó e os estados destes objetos.

Em contrapartida, ao detectar que o nó servidor não possui nível de carga adequado para executar o serviço requisitado, o agente Gerenciador de Nó Servidor ativa o Serviço de Prevenção de Sobrecarga Cliente do nó cliente com a intenção de que o serviço requisitado do objeto servidor passe a ser executado no nó cliente. Durante esta ativação, o agente Gerenciador de Nó Servidor fornece ao Serviço de Prevenção de Sobrecarga Cliente do nó cliente o estado do objeto servidor para que este Serviço seja capaz de decidir a ação de balanceamento de carga a ser efetuada: (i) estado “não em uso”, migra-se o objeto servidor; e (ii) estado “em uso”, replica-se o objeto servidor.

6.2.6 Agente Migrador de Objetos

O agente Migrador de Objetos constitui o Serviço de Prevenção de Sobrecarga Cliente da arquitetura MABal, presente em cada nó do sistema. Este agente é responsável por migrar um objeto servidor para o nó cliente (nó que contém o objeto cliente) ou para outro nó do sistema, previamente escolhido por meio da aplicação de uma política de Seleção de Terceiro Nó que conduz à seleção de um nó adequado, em relação ao balanceamento de carga do sistema, para o recebimento do objeto servidor.

Na realidade, quando um agente Migrador de Objetos recebe um pedido de atuação, este realiza a migração do objeto servidor para o nó onde reside. Ressalta-se que migra-se um objeto servidor caso ele não esteja executando serviços no instante de tempo considerado, ou seja, o objeto servidor apresenta o estado “não em uso”.

A Figura 6.11 ilustra a definição do agente Migrador de Objetos, que inclui a descrição dos eventos que este agente sente, do serviço que oferece (retorno do agente) e do seu comportamento (fluxo de tarefas do agente).

AGENTE-MIGRADOR-DE-OBJETOS

(**eventos:** recebimento de mensagem do agente Gerenciador de Nó Cliente efetuando um pedido de atuação do agente Migrador de Objetos e recebimento de mensagem de um agente Migrador de Objetos de um nó remoto efetuando um pedido de remoção do objeto servidor

)

retorna: informação de objeto servidor migrado e informação de objeto servidor removido

- Fluxo de Tarefas A: ao receber uma mensagem que solicita o início da atuação deste agente:
  - Envia mensagem ao agente Migrador de Objetos do nó servidor efetuando um pedido de remoção do objeto servidor;
  - Cria no nó onde este agente reside uma nova instância da classe do objeto servidor: novoObjServidor ← new ClasseObjServidor();
  - Atualiza a Tabela de Objetos do nó onde este agente reside, inserindo o objeto servidor nesta tabela;
  - Atualiza o estado do objeto servidor, na Tabela de Objetos, para “em uso”.

- Fluxo de Tarefas B: Ao receber uma mensagem, oriunda de um nó remoto, que solicita a remoção do objeto servidor:
  - Elimina o objeto servidor do nó servidor;
  - Atualiza a Tabela de Objetos do nó servidor, eliminando o objeto servidor desta tabela.

Figura 6.11 – Definição do agente Migrador de Objetos

Como descrito na Figura 6.11, o agente Migrador de Objetos possui dois fluxos de tarefas alternativos, que são ativados por mensagens de conteúdos distintos: (i) Fluxo de Tarefas A – um fluxo de tarefas ativado por uma mensagem, oriunda do mesmo nó onde reside o agente, que solicita o início da atuação do agente; e (ii) Fluxo de Tarefas B – um fluxo de tarefas ativado por uma mensagem, oriunda de um nó remoto em relação ao nó onde o agente reside, que solicita a remoção do objeto servidor.

O Fluxo de Tarefas A visa controlar o processo de migração do objeto servidor para o nó onde o agente reside. O processo de migração em si consiste em duas tarefas: primeiramente, eliminar o objeto servidor do nó servidor; e, em segundo, criar o objeto servidor no nó onde o agente reside. No entanto, o agente Migrador de Objetos não é capaz de eliminar um objeto que se localiza em um nó remoto, ou seja, no nó servidor. Portanto, este agente envia uma mensagem ao agente Migrador de Objetos do nó servidor remoto para que o agente Migrador de Objetos do nó servidor realize a tarefa de eliminação do objeto servidor, pois este agente e o objeto servidor a ser removido encontram-se ambos localizados no mesmo nó.

O Fluxo de Tarefas B consiste no fluxo de tarefas executado por um agente Migrador de Objetos residente em um nó servidor para a eliminação do objeto servidor neste nó.

Como qualquer nó do sistema pode exercer o papel de nó servidor e de nó cliente, faz-se necessário que o agente Migrador de Objetos da arquitetura MABal contenha ambos os Fluxos de Tarefas A e B, para que ele seja capaz de: (i) controlar o processo de migração, caso ele resida em um nó cliente ou em outro nó do sistema, escolhido para receber o objeto servidor a ser migrado; ou (ii) eliminar o objeto servidor, caso ele resida no nó servidor.

6.2.7 Agente Replicador de Objetos

O agente Replicador de Objetos constitui o Serviço de Prevenção de Sobrecarga Cliente da arquitetura MABal. Este agente é responsável por replicar um objeto servidor para o nó cliente (nó que contém o objeto cliente que invocou o serviço do objeto servidor) ou para outro nó do sistema, previamente escolhido por meio da aplicação de uma política de Seleção de Terceiro Nó.

Quando um agente Replicador de Objetos recebe um pedido de atuação, este cria uma réplica do objeto servidor no nó onde reside. Na arquitetura MABal cria-se uma réplica de um objeto servidor somente quando este objeto está executando algum serviço no instante de tempo considerado, ou seja, quando o objeto servidor apresentar o estado “em uso”.

A criação de uma réplica de um objeto servidor permite que um nó servidor sobrecarregado não atenda à invocação de serviço, que será atendida pela réplica do objeto servidor, localizada em um nó ocioso do sistema. Portanto, a criação de réplicas de objetos servidores promove o balanceamento de carga do sistema.

A Figura 6.12 ilustra a definição do agente Replicador de Objetos, que inclui a descrição dos eventos que este agente sente, do serviço que oferece (retorno do agente) e do seu comportamento (fluxo de tarefas do agente).

AGENTE-REPLICADOR-DE-OBJETOS

(**eventos:** recebimento de mensagem do agente Gerenciador de Nó Cliente efetuando um pedido de atuação do agente Replicador de Objetos

)

retorna: informação de objeto servidor replicado

- Cria no nó onde este agente reside uma nova instância da classe do objeto servidor: novoObjServidor ← new ClasseObjServidor();
- Atualiza a Tabela de Objetos do nó onde este agente reside, inserindo o objeto servidor nesta tabela;
- Atualiza o estado do objeto servidor, inserido na Tabela de Objetos, para “em uso”.

Figura 6.12 – Definição do agente Replicador de Objetos

6.2.8 Agente Seletor de Nó

O agente Seletor de Nó constitui o Serviço de Prevenção de Sobrecarga Cliente da arquitetura MABal. Este agente é responsável por selecionar um nó do sistema, exceto os nós servidor e cliente, que seja o nó mais adequado para receber o objeto servidor.

Para selecionar o nó receptor do objeto servidor, o agente Seletor de Nó aplica a seguinte política de Seleção de Terceiro Nó: (i) escolhe-se um nó ocioso do sistema que possua objeto(s) com a maior quantidade de relacionamentos com o objeto servidor, a ser migrado ou replicado; e (ii) caso não exista um nó no sistema que possua objeto(s) com relacionamento(s) com o objeto servidor e esteja ocioso, seleciona-se o nó de menor carga do sistema com base nos níveis de carga dos nós.

Considera-se que um objeto remoto *x* apresenta um relacionamento com um objeto *y* quando um serviço do objeto remoto *x* é invocado pelo objeto *y* para o provimento de parte de uma funcionalidade (caso de uso) da aplicação.

Em cada nó do sistema existe a Tabela de Relacionamentos, que registra as invocações entre objetos para a realização de cada caso de uso da aplicação. Assim, nesta tabela consta cada relacionamento existente entre dois objetos da aplicação, ou seja, um objeto (objeto cliente) que invoca um serviço de outro objeto (objeto servidor) para a realização de um determinado caso de uso.

A Figura 6.13 ilustra a definição do agente Seletor de Nó, que inclui a descrição dos eventos que este agente sente, do serviço que oferece (retorno do agente) e do seu comportamento (fluxo de tarefas do agente).

AGENTE-SELETOR-DE-NÓ

(**eventos:** recebimento de mensagem do agente Gerenciador de Nó Cliente efetuando um pedido de atuação do agente Seletor de Nó, contendo o estado do objeto servidor

)

retorna: informação do nó selecionado para receber o objeto servidor

- Obtém da Tabela de Relacionamentos os objetos que apresentam relacionamentos com o objeto servidor, ou seja, os objetos que recebem invocações oriundas do objeto servidor. Estes objetos são chamados de Objetos Invocados;
- Envia uma mensagem ao agente Coletor de Informações de cada nó do sistema solicitando que verifiquem a existência do conjunto de Objetos Invocados nas Tabelas de Objetos dos seus respectivos nós. O objetivo consiste em localizar em quais nós se encontram fisicamente localizados os Objetos Invocados. Após a verificação, cada agente Coletor de Informações retorna a quantidade de Objetos Invocados localizados em seu respectivo nó;
- Seleciona o nó do sistema que contém a maior quantidade de Objetos Invocados;
- Envia mensagem ao agente Neural de Classificação de Carga do nó selecionado solicitando que classifique o nível de carga do seu respectivo nó;
- Analisa a informação do nível de carga do nó selecionado;
  - caso o nível de carga seja ocioso, retorna a informação do nó como sendo o nó selecionado para futuramente receber o objeto servidor, a ser migrado ou replicado;
  - caso contrário, seleciona outro nó do sistema que contém a maior quantidade de Objetos Invocados, envia mensagem ao agente Neural solicitando que classifique o nível de carga deste nó selecionado e analisa a informação do nível de carga do nó. Repetem-se estes passos até obter-se um nó que apresente Objetos Invocados e nível de carga ocioso;
- Caso não exista um nó no sistema que apresente Objetos Invocados e esteja ocioso, envia mensagens aos agentes Neurais de Classificação de Carga de todos os nós do sistema, exceto dos nós servidor e cliente, solicitando que classifiquem os níveis de carga dos seus respectivos nós. Seleciona o nó de menor nível de carga e retorna a informação deste nó selecionado como sendo o nó selecionado para futuramente receber o objeto servidor, a ser migrado ou replicado.

Figura 6.13 – Definição do agente Seletor de Nó

6.2.9 Agente Gerenciador de Nó Cliente

O agente Gerenciador de Nó Cliente constitui o Serviço de Prevenção de Sobrecarga Cliente da arquitetura MABal. Este agente é responsável por gerenciar este Serviço no nó cliente ou em outro nó do sistema, exceto no nó servidor.

A Figura 6.14 ilustra a definição do agente Gerenciador de Nó Cliente, que inclui a descrição dos eventos que este agente sente, do serviço que oferece (retorno do agente) e do seu comportamento (fluxo de tarefas do agente).

AGENTE-GERENCIADOR-DE-NÓ-CLIENTE

(**eventos:** recebimento de mensagem do agente Gerenciador de Nó Servidor do nó servidor ou do agente Gerenciador de Nó Cliente de outro nó do sistema efetuando um pedido de atuação do agente Gerenciador de Nó Cliente do nó considerado. Esta mensagem contém o estado do objeto servidor

)

retorna: mensagem de ativação do Serviço de Prevenção de Sobrecarga Cliente de um nó do sistema (exceto os nós cliente e servidor) previamente selecionado por meio da aplicação de um conjunto de políticas de seleção de nó. Esta mensagem, além de ativar o Serviço de Prevenção de Sobrecarga Cliente do nó selecionado, também informa o estado do objeto servidor

- Monitora novo pedido de atuação;
- Ao receber um pedido de atuação:
  - envia mensagem ao agente Neural de Classificação de Carga solicitando que classifique o nível de carga do nó considerado;
  - analisa a informação do nível de carga do nó:
    - caso o nível de carga do nó seja ocioso, verifica o estado do objeto servidor:
      - caso o estado do objeto servidor seja “em uso”, envia mensagem ao agente Replicador de Objetos para que este agente crie uma réplica do objeto servidor no nó considerado com o objetivo de que a nova solicitação de serviço seja atendida pela réplica no nó considerado. Encerra a atuação do agente.
      - caso o estado do objeto servidor seja “não em uso”, envia mensagem ao agente Migrador de Objetos para que este agente migre o objeto servidor para o nó considerado com o objetivo de que a nova solicitação de serviço não seja atendida pelo objeto servidor no nó servidor, mas seja atendida pelo objeto servidor no nó considerado. Encerra a atuação do agente.
    - caso o nível de carga do nó seja normal ou sobrecarregado, envia mensagem ao agente Seletor de Nó solicitando que este agente selecione o nó mais adequado do sistema para receber o objeto servidor e envia uma mensagem de ativação do Serviço de Prevenção de Sobrecarga Cliente do nó selecionado para que este Serviço gerencie o processo de balanceamento de carga, migrando ou replicando o objeto servidor para o nó selecionado.

Figura 6.14 – Definição do agente Gerenciador de Nó Cliente

Nota-se que o agente Gerenciador de Nó Cliente de um respectivo nó do sistema pode ser ativado ao receber mensagens de dois agentes: uma mensagem do agente Gerenciador de Nó Servidor, enviada a partir do nó servidor, ou uma mensagem do agente Gerenciador de Nó Cliente, enviada a partir de outro nó do sistema.

Quando o agente Gerenciador de Nó Cliente é ativado pelo agente Gerenciador de Nó Servidor significa que o Serviço de Prevenção de Sobrecarga Servidor da arquitetura MABal identificou que o nó servidor não possui nível de carga adequado para executar o serviço requisitado. Portanto, o agente Gerenciador de Nó Servidor ativa o Serviço de Prevenção de Sobrecarga Cliente do nó cliente com a intenção de que o serviço requisitado passe a ser executado no nó cliente. No entanto, o nó cliente não contém o objeto servidor. Assim, o objeto servidor deverá ser migrado ou replicado para o nó cliente para que o serviço requisitado possa ser executado neste nó. A escolha das ações de migrar ou de replicar o objeto servidor será realizada com base na informação do estado do objeto servidor. Assim, a informação do estado do objeto servidor conduzirá o agente Gerenciador de Nó Cliente a solicitar as seguintes ações: (i) estado “não em uso”, uma migração do objeto servidor; e (ii) estado “em uso”, uma replicação do objeto servidor.

Quando o agente Gerenciador de Nó Cliente de um respectivo nó é ativado pelo agente Gerenciador de Nó Cliente de outro nó do sistema significa que a arquitetura MABal identificou a seguinte seqüência de situações:

- Primeira situação, o nó servidor não possui nível de carga adequado para executar o serviço requisitado (situação identificada pelo Serviço de Prevenção de Sobrecarga Servidor). A identificação desta situação conduz

o Serviço de Prevenção de Sobrecarga Servidor a solicitar a atuação do Serviço de Prevenção de Sobrecarga Cliente do nó cliente;

- Segunda situação, o nó cliente também não possui nível de carga adequado para executar o serviço requisitado (situação identificada pelo Serviço de Prevenção de Sobrecarga Cliente no nó cliente).

Em decorrência da inviabilidade (em relação ao equilíbrio de carga do sistema) dos nós servidor e cliente executarem o serviço requisitado, o agente Gerenciador de Nó Cliente do nó cliente expande a busca por um nó adequado para executar o serviço requisitado a outro nó do sistema. Portanto, este é o contexto de situações que conduzem o agente Gerenciador de Nó Cliente do nó cliente a enviar uma mensagem ao agente Gerenciador de Nó Cliente de outro nó do sistema solicitando a sua atuação.

6.3 Considerações Finais

As atuações dos agentes que constituem a arquitetura MABal mostram que esta arquitetura possui prioridades durante a busca por um nó adequado para a execução de um determinado serviço requisitado, a fim de prover balanceamento de carga ao sistema.

A ordem com que a arquitetura MABal analisa os níveis de carga dos nós do sistema, a fim de identificar o nó adequado a receber o objeto servidor e, portanto, a prover o serviço requisitado, favorece a aglutinação dos objetos que colaboram para a realização de um mesmo caso de uso da aplicação em um único nó do sistema.

Este benefício de promover a aglutinação de objetos que realizam um caso de uso em um único nó do sistema é obtido pela seguinte ordem de prioridade com que a arquitetura analisa os níveis de carga dos nós do sistema:

- A primeira prioridade da arquitetura MABal consiste em executar o serviço requisitado no próprio nó servidor, que já contém o objeto servidor, quando este possui nível de carga adequado para atender à nova invocação de serviço. Esta prioridade evita as ações de migração ou de replicação do objeto servidor para outro nó do sistema;

- A segunda prioridade da arquitetura MABal consiste em executar o serviço requisitado no nó cliente, que contém o objeto que invocou o serviço requisitado (objeto cliente), quando este possui nível de carga adequado para atender à nova invocação de serviço. Apesar do nó cliente não conter o objeto servidor e, portanto, ser necessário realizar uma migração ou replicação do objeto servidor para o nó cliente, este nó contém um objeto que se relaciona com o objeto servidor. Conseqüentemente, a migração ou replicação do objeto servidor para o nó cliente promoverá a aglutinação de ambos os objetos, ou seja, do objeto cliente e do objeto servidor, em um único nó do sistema. A localização dos objetos cliente e servidor em um único nó do sistema, neste caso no nó cliente, evita acessos à rede do sistema para efetuar a invocação do serviço requisitado e para o retorno do serviço. Dessa forma, a atuação da arquitetura MABal promove uma forma de realocação dos objetos da aplicação distribuída nos nós do sistema que contempla as colaborações entre estes objetos para as realizações dos casos de uso da aplicação;

- A terceira prioridade da arquitetura MABal consiste em executar o serviço requisitado em outro nó do sistema, exceto nos nós cliente e servidor. Este

terceiro nó também não contém o objeto servidor e, portanto, faz-se necessário realizar uma migração ou replicação do objeto servidor para o nó considerado. Em detrimento da ação de migração ou de replicação que deverá ser efetuada, a arquitetura MABal direciona a seleção de um nó, para receber o objeto servidor, que ofereça benefícios ao balanceamento de carga do sistema e ao desempenho deste. O nó escolhido, além de possuir nível de carga adequado para a execução do serviço requisitado, deve apresentar relacionamentos com o objeto servidor, ou seja, conter objetos que sejam invocados pelo objeto servidor. Estes critérios para a seleção do nó receptor promovem: (i) o balanceamento de carga do sistema, à medida que escolhem um nó ocioso do sistema para executar o serviço requisitado; e (ii) a realocação do objeto servidor para um nó do sistema onde existem objetos que colaboram com o objeto servidor para a realização do caso de uso considerado, evitando acessos à rede do sistema para a comunicação entre os objetos de um mesmo caso de uso.

7 ESTUDO DE CASO E ANÁLISE COMPARATIVA DA ARQUITETURA MABal

Este capítulo apresenta um exemplo de utilização da arquitetura MABal com o objetivo de mostrar o funcionamento desta abordagem multi-agente distribuída de balanceamento de carga. O exemplo consiste em utilizar uma aplicação de objetos distribuídos denominada Software Simulador de Satélites Distribuído (Software SSD) e aplicar à este software o balanceamento de carga proposto pela arquitetura MABal.

Primeiramente, apresenta-se neste capítulo uma descrição do Software SSD e, posteriormente, um estudo de caso que simula a arquitetura MABal sendo aplicada ao Software SSD. Para realizar esta simulação foi desenvolvida uma ferramenta de simulação de balanceamento de carga denominada ferramenta SimBal, a qual apresenta os resultados da realização do estudo de caso.

Para fins de comparação, além de aplicar ao Software SSD a abordagem de balanceamento de carga da arquitetura MABal, aplicou-se também ao Software SSD uma outra abordagem de balanceamento de carga – o serviço de nomes da especificação CORBA. Neste capítulo apresenta-se a análise comparativa realizada entre estas duas abordagens.

7.1 Software Simulador de Satélites Distribuído

A arquitetura MABal realiza balanceamento de carga em aplicações que são desenvolvidas utilizando a tecnologia de Objetos Distribuídos. O Software SSD foi desenvolvido utilizando esta tecnologia, ou seja, é um software composto por objetos que podem ser distribuídos no sistema e que usufruem dos benefícios da distribuição de objetos: o aumento da disponibilidade de serviços, a tolerância a

falhas, a flexibilidade, e outros. Ferreira (2001) mostra que estas características são importantes em Softwares de Controle de Satélites devido ao fato destes softwares apresentarem variações na demanda por serviços nas diferentes fases de seus ciclos de vida.

O Software SSD, desenvolvido neste trabalho, é uma implementação simplificada de um Simulador de Satélites real. A finalidade principal de um Simulador de Satélites é permitir treinar os operadores de satélite quanto ao controle e rastreio do mesmo e prover um ambiente real que possa ser utilizado para elaboração de testes do software de controle de satélites.

Um Simulador de Satélites deve realizar as seguintes funções:

- Simular características físicas necessárias à operação do satélite, como: órbita, atitude, campo magnético e incidência da luz solar;
- Simular todas as funções dos subsistemas do satélite real como: telemetria, telecomando, temperatura, medidas de distância, azimute, elevação, carga e descarga da bateria, e ainda funções das estações de rastreio.

Sendo assim, um Simulador de Satélites tem como objetivo simular o funcionamento de um satélite, inclusive no que diz respeito aos efeitos das condições ambientais, as quais o mesmo está sujeito em órbita, tornando-o indistinguível do satélite real.

O Software SSD, utilizado no estudo de caso apresentado neste capítulo, é um Simulador de Satélites Distribuído. Este software foi implementado com uma quantidade reduzida de funcionalidades se comparado a um Simulador de Satélites real.

7.2 Modelagem do Software Simulador de Satélites Distribuído

Para a modelagem do Software Simulador de Satélites Distribuído utilizou-se os seguintes diagramas da UML: diagrama de casos de uso, de classes e de seqüência.

7.2.1 Diagrama de Casos de Uso

O diagrama de casos de uso descreve as funcionalidades de uma aplicação desempenhadas por atores externos. Um ator é alguém ou algo externo à aplicação que interage com a mesma. Graficamente os atores são representados como bonecos.

Além dos atores, o diagrama de casos de uso é composto por elementos denominados casos de uso. Um caso de uso representa uma funcionalidade oferecida pelo sistema, ou seja, é a descrição de um conjunto de transações realizadas pela aplicação que produz resultados observáveis para um determinado ator (FOWLER et al., 2005). Um caso de uso mostra o comportamento pretendido da aplicação, sem especificar “como” o comportamento é implementado. Graficamente um caso de uso é representado por uma elipse de linhas contínuas, incluindo somente seu nome.

A Figura 7.1 mostra o diagrama de casos de uso do Software Simulador de Satélites Distribuído, o qual constitui a base para o desenvolvimento dos outros diagramas desta aplicação.

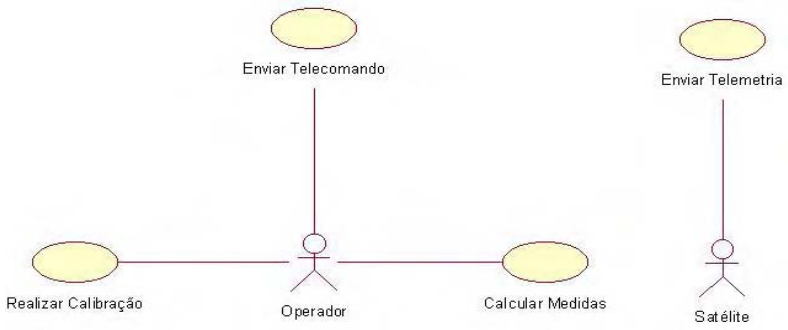


Figura 7.1 – Diagrama de casos de uso do Software Simulador de Satélites Distribuído

7.2.2 Diagrama de Classes

Uma aplicação orientada a objetos é composta por classes e por um conjunto de objetos que interagem para execução das funcionalidades (casos de uso) oferecidas pela aplicação. As classes de uma aplicação e o grau do relacionamento entre elas são mostrados pelo diagrama de classes, o qual descreve a estrutura estática de uma aplicação. A Figura 7.2 apresenta o diagrama de classes do Software Simulador de Satélites Distribuído.

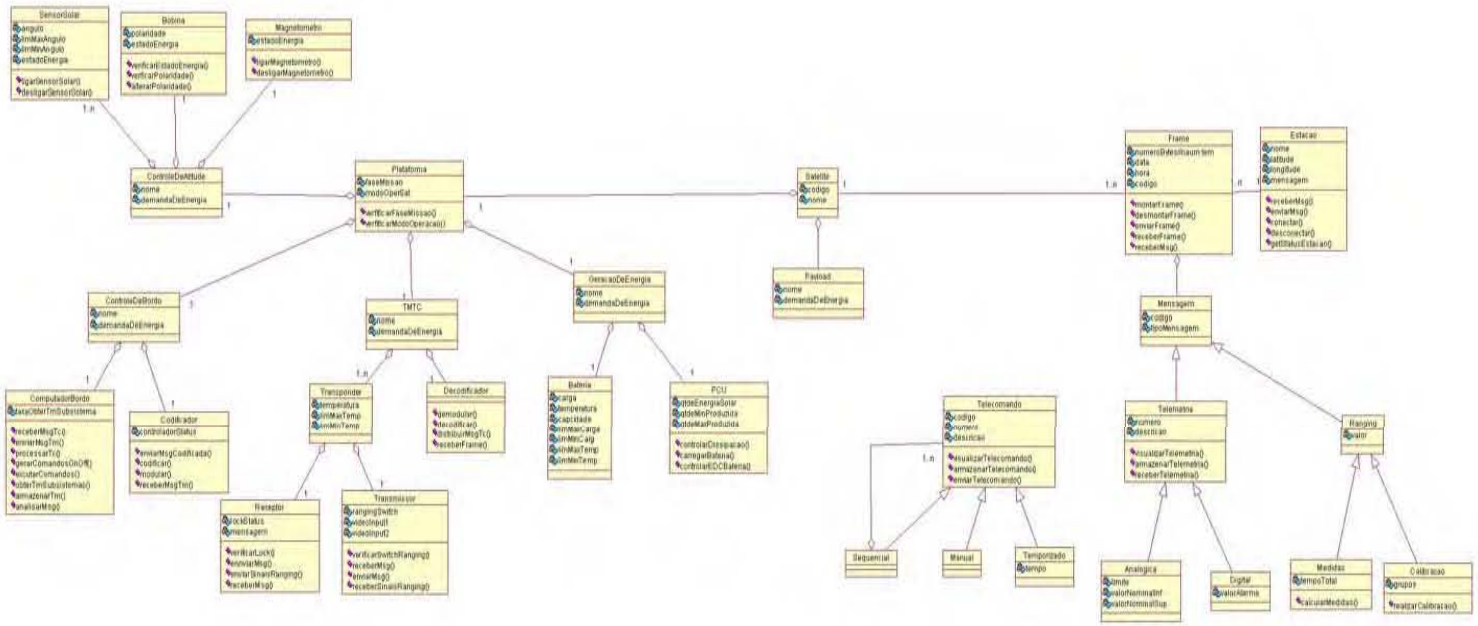


Figura 7.2 – Diagrama de classes do Software Simulador de Satélites Distribuído

7.2.3 Diagrama de Seqüência

Em uma aplicação orientada a objetos as funcionalidades (casos de uso) são fornecidas através da interação de grupos de objetos. Os objetos interagem através de comunicações de forma que juntos, cada um com suas responsabilidades, eles realizem os casos de uso.

O objetivo de um diagrama de seqüência é descrever as comunicações necessárias entre os objetos para a realização de uma funcionalidade (caso de uso). Um diagrama de seqüência é um diagrama de objetos, ou seja, ele contém um conjunto de objetos de diferentes classes e é denominado diagrama de seqüência porque descreve ao longo de uma linha do tempo a seqüência de comunicações entre os objetos envolvidos, modelando, assim, os aspectos dinâmicos da aplicação (Fowler et al., 2005).

Realização do Caso de Uso Enviar Telecomando

O caso de uso Enviar Telecomando do Software SSD foi o caso de uso utilizado no estudo de caso do funcionamento da arquitetura MABal. Por esse motivo, é importante que seja apresentado os objetos e a sequência de comunicações entre eles para a realização deste caso de uso. Sendo assim, estas informações estão ilustradas na Figura 7.3, a qual apresenta o diagrama de seqüência do caso de uso Enviar Telecomando do Software SSD.

Telecomandos são mensagens enviadas ao satélite para corrigir ou mudar posições de chaves, ligar e desligar sensores, enfim, alterar o estado de algum equipamento. Estas mensagens chegam ao satélite por intermédio de uma estação, a qual é responsável pela recepção dos dados dos satélites. Atualmente os satélites controlados pelo INPE utilizam duas estações, a estação de Cuiabá e a de Alcântara.

O operador de satélites é quem inicia o caso de uso Enviar Telecomando. O operador seleciona o telecomando a ser enviado, estabelece conexão com a estação (Alcântara ou Cuiabá) e envia o telecomando, o qual passa por várias etapas até chegar ao satélite.

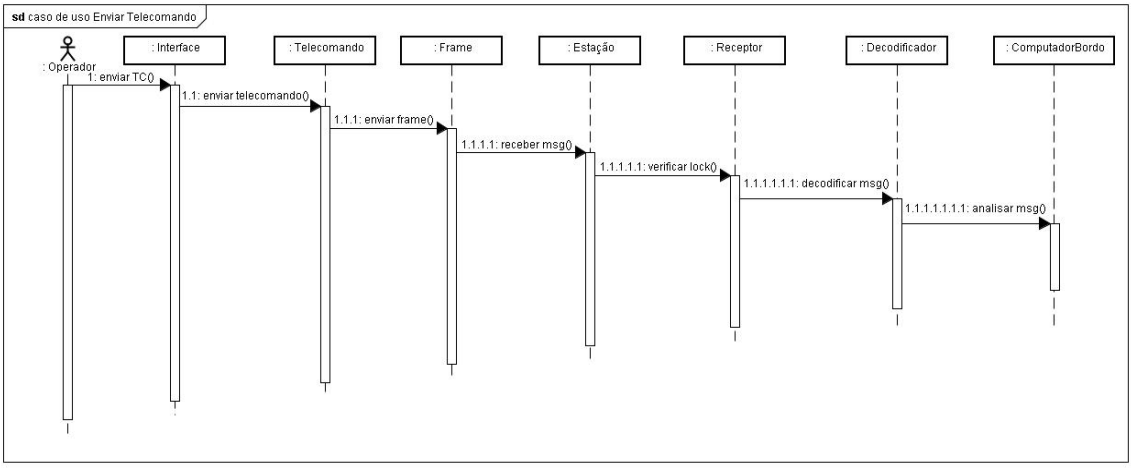


Figura 7.3 – Diagrama de seqüência do caso de uso Enviar Telecomando

Para a modelagem do Software SSD (Simulador de Satélites Distribuído) utilizou-se um conjunto de diagramas da UML para mostrar os objetos que compõem esta aplicação e como eles interagem entre si para prover as funcionalidades (casos de uso) da aplicação.

O Software SSD foi utilizado como o programa-exemplo do estudo de caso do funcionamento da arquitetura MABal e o caso de uso Enviar Telecomando foi o caso de uso utilizado no estudo de caso. A seguir apresenta-se o cenário que foi definido para a realização do estudo de caso.

7.3 Cenário de Execução do Estudo de Caso

O estudo de caso do funcionamento da arquitetura MABal consiste em executar uma aplicação de objetos distribuídos aplicando-se o balanceamento de carga proposto neste trabalho de pesquisa.

Para mostrar as idéias propostas, o estudo de caso foi realizado de forma simulada. Para isso, desenvolveu-se uma ferramenta de simulação de balanceamento de carga, denominada ferramenta SimBal, a qual simula a execução de uma aplicação de objetos distribuídos aplicando o balanceamento de carga proposto na arquitetura MABal. Para o desenvolvimento da ferramenta SimBal, utilizou-se as seguintes ferramentas e linguagens: gerenciador de base de dados Microsoft Access 2007; NetBeans IDE 5.5.1 e linguagem Java - jdk 1.6.0_02.

O primeiro passo para realizar a simulação foi definir um cenário de execução para o estudo de caso. Um cenário de execução deve apresentar as seguintes informações:

- A aplicação de objetos distribuídos a ser executada (um caso de uso desta aplicação);
- a quantidade de nós do sistema;
- a distribuição dos objetos nos nós do sistema;
- o tempo de execução dos objetos;
- os níveis de carga dos nós do sistema para cada invocação de serviço da aplicação.

Estas informações foram definidas no cenário de execução do estudo de caso. Definiu-se o caso de uso Enviar Telecomando do Software Simulador de Satélites Distribuído (SSD) como sendo a aplicação de objetos distribuídos a ser utilizada no estudo de caso. Conforme ilustra a Figura 7.3, o caso de uso

Enviar Telecomando é constituído pelos seguintes objetos: computadorBordo (c), decodificador (d), estação (e), frame (f), interface (i), receptor (r) e telecomando (t). Estes objetos devem localizar-se de forma distribuída entre os nós do sistema e o sistema deve apresentar níveis de carga variados no decorrer da execução destes objetos.

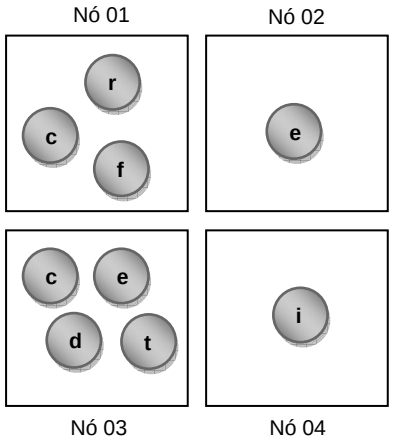


Figura 7.4 – Distribuição dos objetos do caso de uso Enviar Telecomando em um sistema constituído por quatro nós

A Figura 7.4 apresenta as seguintes informações do cenário de execução do estudo de caso:

- A quantidade de nós do sistema: quatro nós (este valor foi definido de forma empírica);
- A distribuição dos objetos no sistema: os objetos do caso de uso Enviar Telecomando foram distribuídos entre os quatro nós do sistema. Os objetos comunicam entre si conforme foi retratado pelo diagrama de sequência do caso de uso Enviar Telecomando, ilustrado na Figura 7.3. Um objeto invoca o serviço de outro objeto e a execução desta sequência de serviços resulta na realização do caso de uso Enviar Telecomando.

O objeto que invoca um serviço de outro objeto assume o papel de Objeto Cliente e o objeto que atende à esta invocação executando o serviço solicitado assume o papel de Objeto Servidor. A Figura 7.5 ilustra esta definição de papéis em uma invocação de serviço.

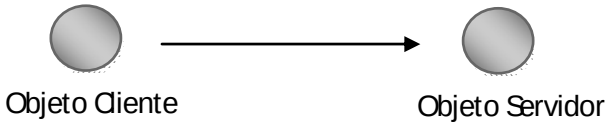


Figura 7.5 – Invocação de serviço

É importante ressaltar que em uma determinada invocação de serviço, o objeto pode assumir o papel de objeto servidor e na invocação seguinte este mesmo objeto pode assumir o papel de cliente.

Para simular a execução dos objetos da aplicação, um tempo de execução foi associado a cada objeto em forma de pesos. Estes valores são apresentados na Tabela 7.1 e foram definidos de forma empírica.

Tabela 7.1 – Tempo de execução associado aos objetos do caso de uso enviar telecomando

Objeto	Tempo de Execução
compBordo	1
decodificador	3
estação	3
frame	1
receptor	2
telecomando	2

É importante ressaltar que os nós do sistema não estão dedicados somente à execução da aplicação de objetos distribuídos, ou seja, outros processos

podem ser executados nestes nós de forma concomitante. Sendo assim, os níveis de carga dos nós podem oscilar com frequência. Para que fosse possível retratar esta realidade na simulação do funcionamento da arquitetura MABal, foi definido, no cenário de execução, os níveis de carga dos nós do sistema para cada invocação de serviço da aplicação. Estas informações são apresentadas na Tabela 7.2.

Tabela 7.2 – Níveis de carga dos nós do sistema para cada invocação de serviço do caso de uso enviar telecomando

Invocação do Objeto Servidor	**Níveis de Carga dos Nós**			
	Nó 01	**Nó 02**	**Nó 03**	**Nó 04**
telecomando (1ª invocação de serviço)	4	2	5	1
frame (2ª invocação de serviço)	5	4	3	2
estação (3ª invocação de serviço)	1	5	4	5
receptor (4ª invocação de serviço)	2	5	4	3
decodificador (5ª invocação de serviço)	4	3	5	2
compBordo (6ª invocação de serviço)	5	3	4	2

A Tabela 7.2 apresenta os níveis de carga dos nós do sistema durante a execução da aplicação, pois sempre que o serviço de um objeto servidor for requisitado, cada nó do sistema apresentará, naquele instante, uma situação de carga.

As informações descritas nesta seção são fornecidas, pelo usuário, à ferramenta de simulação SimBal para definir o cenário de execução. A Figura 7.6 ilustra a tela principal do editor da base de dados da ferramenta SimBal. Neste editor, o usuário fornece as informações do cenário de execução, tais como: a quantidade de nós do sistema; os objetos da aplicação e seus respectivos tempos de execução; a distribuição destes objetos nos nós do sistema; e os casos de uso da aplicação com a sequência de invocação dos objetos para cada caso de uso.

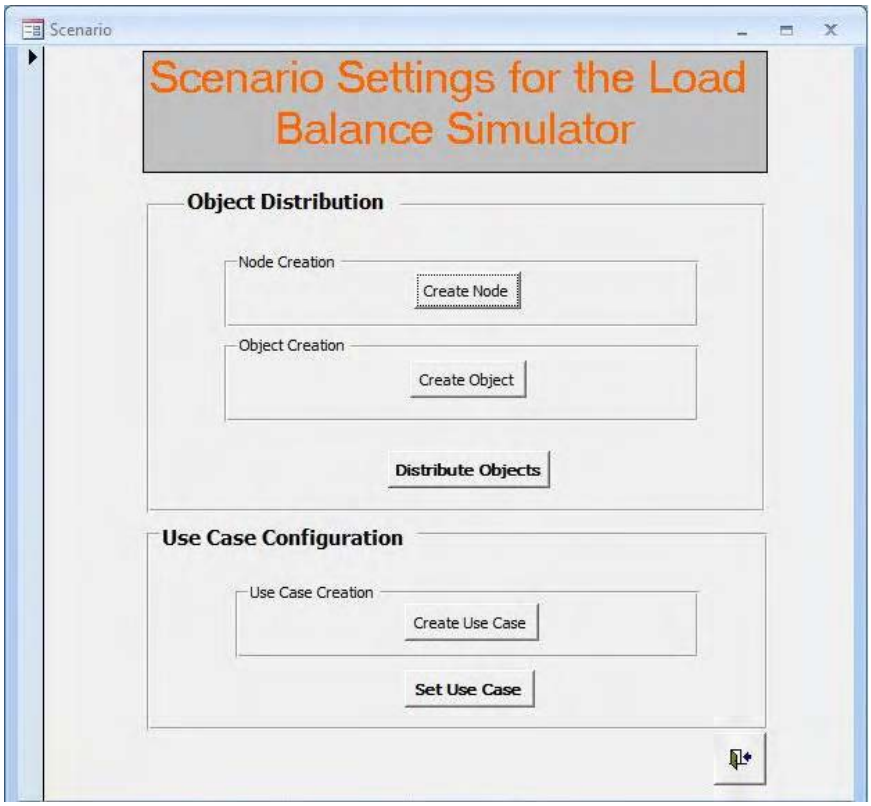


Figura 7.6 – Tela do editor da base de dados da ferramenta SimBal para a definição do cenário de execução

As informações sobre o nível de carga dos nós do sistema para cada invocação de serviço são fornecidas, pelo usuário, para a ferramenta SimBal durante a execução das invocações de serviço por meio da entrada de dados padrão.

7.4 Medidas do Processo de Simulação

A ferramenta de simulação SimBal calcula algumas medidas durante o processo de simulação, que são:

- O tempo de execução do objeto servidor em um determinado instante de tempo;
- O tempo gasto para realizar um caso de uso;
- A existência ou não de acesso à rede por uma invocação de serviço;
- A quantidade de acessos à rede durante a realização de um caso de uso.

Estas medidas encontram-se descritas a seguir.

7.4.1 Medidas de Tempo de Execução

A ferramenta de simulação SimBal calcula uma medida, denominada neste trabalho de pesquisa como sendo o Tempo de Execução do Objeto Servidor em um Determinado Instante de Tempo (*tt*), a qual expressa o tempo gasto para executar um objeto servidor levando em consideração o nível de carga do nó onde ele está localizado. Esta medida *tt* é calculada da seguinte forma:

$$tt = to * nc \qquad (7.1)$$

onde:

- to representa o tempo gasto para executar o serviço disponibilizado pelo objeto servidor;
- nc representa o nível de carga do nó onde o objeto servidor reside no instante da invocação deste objeto.

O cálculo realizado para obter *tt* expressa a influência que o nível de carga do nó exerce no tempo gasto para executar um objeto servidor. Por exemplo, considerando um nó sobrecarregado, o tempo de execução de um objeto neste

nó é maior do que o tempo de execução deste mesmo objeto em um nó com menor carga.

Outra medida calculada pela ferramenta de simulação SimBal é a medida denominada Tempo de Execução do Caso de Uso (*tc*), a qual expressa o tempo gasto para realizar um caso de uso da aplicação.

Como um caso de uso representa uma sequência de invocações de serviço que provê uma funcionalidade ao usuário da aplicação, o tempo gasto para realizar um caso de uso é o somatório dos tempos de execução de todos os objetos deste caso de uso. Assim, a ferramenta SimBal calcula a medida *tc* realizando o somatório das medidas *tt* de todos os objetos de um caso de uso.

7.4.2 Medidas de Acesso à Rede

A ferramenta de simulação SimBal define também uma medida que expressa se uma invocação de serviço utiliza ou não a rede, para realizar transferência de dados, na comunicação entre o objeto cliente e o objeto servidor.

Em uma aplicação de objetos distribuídos, o objeto cliente pode invocar um serviço remoto (disponibilizado pelo objeto servidor) e o middleware se encarrega de fornecer ao cliente uma referência que lhe permita utilizar esse serviço. Uma conexão é, então, estabelecida entre o objeto cliente e o objeto servidor. Em seguida, é realizada a comunicação entre estes objetos através da rede.

O objetivo do middleware é tornar transparente para o programador o mecanismo que permite que os objetos se comuniquem através de uma rede. Sendo assim, o middleware realiza todas as funções de rede e ordenação

(*marshalling*) dos dados, ou seja, realiza o empacotamento de argumentos de função e de valores de retorno para serem transmitidos pela rede.

Em uma aplicação de objetos distribuídos, as invocações de serviço geram transmissão de dados pela rede se o objeto cliente e o objeto servidor estiverem localizados em nós distintos. Neste trabalho de pesquisa, denomina-se Acesso à Rede pela Invocação de Serviço (rs) como sendo a transmissão de dados pela rede causada pela comunicação entre o objeto cliente e o objeto servidor em uma invocação de serviço.

O *rs* é definido, para cada invocação de serviço, da seguinte forma:

$$rs = \begin{cases} 0, & \text{se o objeto cliente e o objeto servidor localizam-se no mesmo nó, o que implica em ausência de acesso à rede na comunicação entre os objetos} \\ 1, & \text{se o objeto cliente e o objeto servidor localizam-se em nós distintos, o que implica em realizar acesso à rede na comunicação entre os objetos} \end{cases} \quad (7.2)$$

Além da medida *rs*, a ferramenta SimBal calcula uma outra medida que expressa quantas vezes houve acesso à rede durante a realização de um caso de uso. Esta medida, denominada neste trabalho de pesquisa como sendo a Quantidade de Acesso à Rede pelos Objetos de um Caso de Uso (qr), é calculada pelo somatório das medidas *rs* de todas as invocações de serviço de um caso de uso.

7.5 Execução do Cenário Aplicando CORBA e a Arquitetura MABal

O estudo de caso apresentado neste capítulo consiste em aplicar o balanceamento de carga da arquitetura MABal ao cenário de execução definido

na Seção 7.3 com o objetivo de mostrar o funcionamento da arquitetura MABal. A seção atual descreve a execução deste cenário aplicando-se a arquitetura MABal e apresenta os valores obtidos para as medidas de tempo de execução e de acesso à rede dos objetos envolvidos.

Além disso, aplicou-se para o mesmo cenário de execução o serviço de nomes da especificação CORBA com o objetivo de comparar as duas abordagens – CORBA e MABal. Sendo assim, encontra-se descrito a seguir o mecanismo de balanceamento de carga realizado pelo serviço de nomes CORBA.

7.5.1 O Serviço de Nomes da Especificação CORBA

Nos nós de um sistema podem existir cópias (réplicas) de um objeto. Na existência de réplicas de um objeto no sistema e diante da ocorrência de uma invocação de serviço, mais de um objeto pode atender ao serviço requisitado. Neste caso é preciso selecionar, dentre as réplicas, o objeto que atenderá a invocação de serviço, ou seja, é preciso selecionar qual objeto desempenhará o papel de objeto servidor.

O serviço de nomes CORBA realiza a seleção do objeto servidor por meio de uma fila circular das réplicas. A seleção do objeto é realizada de forma sequencial. Por exemplo, para um objeto que tem três réplicas, diante de uma invocação de serviço ao objeto, CORBA seleciona o primeiro objeto da fila, em uma nova invocação de serviço, seleciona o segundo objeto da fila, em uma próxima invocação, seleciona o terceiro objeto e tendo em vista que a fila é circular, em uma próxima invocação, seleciona novamente o primeiro objeto da fila e assim por diante.

7.5.2 Invocações de Serviço

O caso de uso Enviar Telecomando é constituído por seis invocações de serviço conforme encontra-se definido no diagrama de sequência deste caso de uso ilustrado na Figura 7.3. Uma invocação de serviço retrata um objeto invocando o serviço de outro objeto.

A Figura 7.7 apresenta: (i) as invocações de serviço do caso de uso Enviar Telecomando; (ii) a sequência na qual estas invocações devem ocorrer para realizar o caso de uso Enviar Telecomando; e (iii) os papéis desempenhados pelos objetos, ou seja, se um objeto desempenha o papel de objeto cliente ou de objeto servidor em uma determinada invocação de serviço.

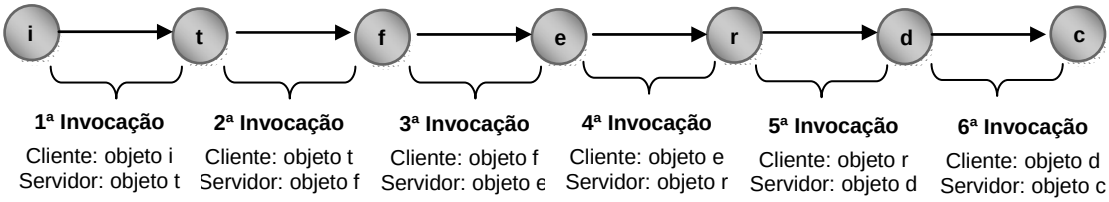


Figura 7.7 – Sequência de invocação de serviço para a realização do caso de uso enviar telecomando

A seguir apresenta-se, passo a passo, a execução do cenário definido na Seção 7.3 aplicando-se a este cenário o balanceamento de carga da arquitetura MABal e também o serviço de nomes da especificação CORBA. O procedimento realizado por MABal e por CORBA na execução das invocações de serviço da Figura 7.7 encontra-se detalhado a seguir.

7.5.3 Primeira Invocação de Serviço

A Figura 7.8 é constituída por duas partes: a parte (a) retrata a execução da primeira invocação de serviço do caso de uso Enviar Telecomando aplicando-se o serviço de nomes da especificação CORBA e a parte (b) retrata a

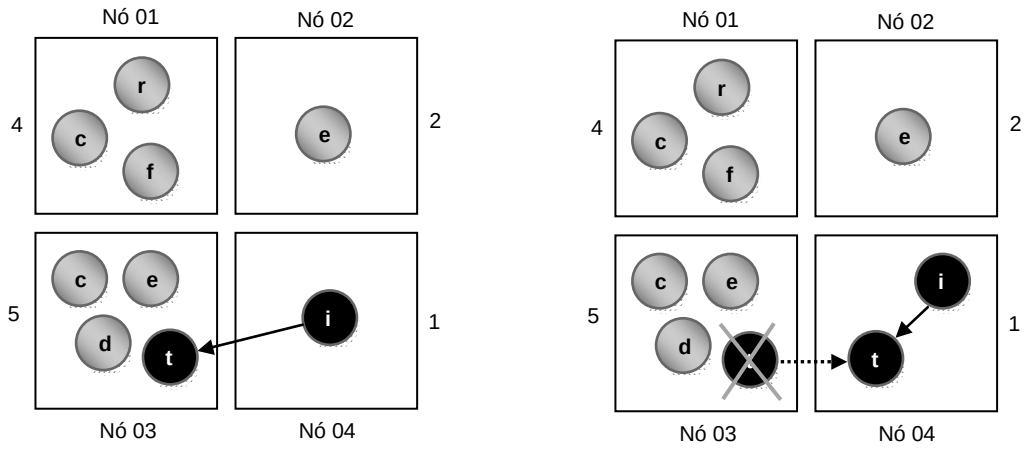
execução desta mesma invocação de serviço aplicando-se a arquitetura MABal.

Na primeira invocação de serviço do caso de uso Enviar Telecomando, o objeto interface (objeto i), que desempenha o papel de objeto cliente, invoca o serviço do objeto telecomando (objeto t), que desempenha o papel de objeto servidor.

Na Figura 7.8, nas partes (a) e (b), são apresentados somente os objetos envolvidos na realização do caso de uso Enviar Telecomando. Os demais objetos do Software SSD também estão presentes nos nós do sistema, porém não foram ilustrados na figura, pois considera-se relevante ilustrar somente os objetos envolvidos no caso de uso considerado.

Nota-se na Figura 7.8, nas partes (a) e (b), que existe um valor posicionado graficamente na lateral de cada nó do sistema. Este valor representa o nível de carga do nó no instante em que a invocação de serviço é executada no sistema.

É importante destacar que para uma determinada invocação de serviço os nós do sistema apresentam uma determinada situação de carga, ou seja, apresentam determinados níveis de carga para os nós. Entretanto, esta situação de carga dos nós pode ser diferente para as outras invocações de serviço da aplicação, pois os nós do sistema não são dedicados somente à execução da aplicação, ou seja, outros processos podem ser executados nestes nós simultaneamente. Os valores que foram definidos para os níveis de carga dos nós para cada invocação de serviço do caso de uso Enviar Telecomando encontram-se na Tabela 7.2.



parte (a) – Middleware CORBA

parte (b) – Arquitetura MABal: migração para nó cliente

Figura 7.8 – Primeira invocação de serviço do caso de uso enviar telecomando

7.5.3.1 Simulação da 1ª Invocação de Serviço Utilizando CORBA

A parte (a) da Figura 7.8 ilustra a primeira invocação de serviço do caso de uso Enviar Telecomando, ou seja, a invocação do serviço do objeto telecomando (t) pelo objeto interface (i). Na simulação desta invocação de serviço, realizada pela ferramenta SimBal, aplicou-se o serviço de nomes da especificação CORBA.

O serviço de nomes CORBA atuou nesta invocação de serviço da seguinte maneira:

- Devido ao fato de não existirem réplicas do objeto telecomando (t) nos nós do sistema, o serviço de nomes CORBA selecionou o objeto telecomando (t), localizado no nó 03, como sendo o objeto servidor da primeira invocação de serviço. Portanto, o objeto telecomando (t) foi executado no nó 03.

Os valores das medidas coletadas na simulação da primeira invocação de serviço aplicando-se o serviço CORBA foram:

- Aplicou-se a Equação 7.1 para calcular o tempo de execução do objeto telecomando (t) e obteve-se o seguinte resultado: tt = 10.
- Aplicou-se a Equação 7.2 para definir a existência ou não de acesso à rede na comunicação entre o objeto interface (i) e o objeto telecomando (t) e o resultado obtido foi: rs = 1.

7.5.3.2 Simulação da 1ª Invocação de Serviço Utilizando a Arquitetura MABal

A parte (b) da Figura 7.8 ilustra a execução da primeira invocação de serviço do caso de uso Enviar Telecomando aplicando-se a arquitetura MABal. Durante a simulação desta invocação de serviço, a arquitetura MABal atuou conforme encontra-se descrito a seguir.

Com a ocorrência da invocação do objeto telecomando (t), primeiramente, o agente Gerenciador de Nó Servidor (localizado no nó servidor, ou seja, nó 03) solicitou ao agente Neural a classificação da carga do nó servidor. A classificação gerada pelo agente Neural para o nó 03 foi “muito carregado”, ou seja, nível 5. Uma vez que o nó servidor apresentou-se sobrecarregado, o objeto servidor (objeto t) não será executado neste nó e sim em outro nó do sistema que seja mais adequado.

Para realizar a escolha do nó mais adequado, o agente Gerenciador de Nó Servidor solicitou a atuação do agente Gerenciador de Nó Cliente (localizado no nó cliente, nó 04). Por sua vez, este agente solicitou ao agente Neural a classificação da carga do nó cliente. O agente Neural classificou o nó 04 em “muito ocioso”, ou seja, nível 1. Uma vez que o nível de carga do nó cliente apresentou-se ocioso, o objeto servidor será executado neste nó.

No entanto, o objeto servidor não se encontra fisicamente localizado no nó cliente (nó 04). Portanto, é necessário migrar o objeto servidor do nó 03 para o nó 04. Para realizar esta migração, o agente Gerenciador de Nó Cliente solicitou ao agente Migrador de Objetos que realizasse a migração. Assim, após realizada a migração, o objeto servidor (objeto t) foi executado no nó 04, ou seja, no mesmo nó onde o objeto cliente (objeto i) está localizado.

As medidas calculadas para o objeto servidor (objeto telecomando) foram:

- Aplicou-se a Equação 7.1 para calcular o tempo de execução do objeto telecomando (t) e obteve-se o seguinte resultado: tt = 2.
- Aplicou-se a Equação 7.2 para definir a existência ou não de acesso à rede na comunicação entre o objeto interface (i) e o objeto telecomando (t) e o resultado obtido foi: rs = 0.

7.5.4 Segunda Invocação de Serviço

Na segunda invocação de serviço do caso de uso Enviar Telecomando, o objeto telecomando (t) desempenha o papel de objeto cliente e o objeto frame (f), de objeto servidor.

A parte (a) da Figura 7.9 ilustra a execução da segunda invocação de serviço aplicando-se o serviço de nomes da especificação CORBA e a parte (b) da figura ilustra a execução desta mesma invocação de serviço aplicando-se a arquitetura MABal. O procedimento realizado por ambas as abordagens encontra-se descrito a seguir.

7.5.4.1 Simulação da 2ª Invocação de Serviço Utilizando CORBA

Uma vez que existe nos nós do sistema somente um único objeto frame (f), ou seja, não existem réplicas deste objeto no sistema, o serviço de nomes CORBA selecionou como sendo o objeto servidor o objeto frame (f), o qual está localizado no nó 01 do sistema.

Os valores obtidos para as medidas de tempo de execução e de acesso à rede durante a execução do objeto servidor frame no nó 01 foram:

- Aplicou-se a Equação 7.1 para calcular o tempo de execução do objeto frame (f) e obteve-se o seguinte resultado: tt = 5.
- Aplicou-se a Equação 7.2 para definir a existência ou não de acesso à rede na comunicação entre o objeto telecomando (t) e o objeto frame (f) e o resultado obtido foi: rs = 1.

7.5.4.2 Simulação da 2ª Invocação de Serviço Utilizando a Arquitetura MABal

Conforme ilustra a parte (b) da Figura 7.9, no instante em que a segunda invocação de serviço ocorre, os nós do sistema apresentam a seguinte situação de carga:

- Nó servidor (nó 01): nível de carga igual a 5, ou seja, “muito carregado”;
- Nó cliente (nó 04): nível de carga igual a 2, ou seja, “pouco ocioso”;

Diante da invocação do objeto servidor frame (objeto f), a arquitetura MABal analisou a situação de carga dos nós cliente e servidor. Tendo em vista que o nó servidor estava sobrecarregado e o nó cliente estava ocioso, a arquitetura MABal decidiu realizar uma migração do objeto servidor para o nó cliente. O procedimento realizado pelos agentes da arquitetura MABal para tomar esta decisão de balanceamento foi o mesmo realizado pelos agentes durante a primeira invocação de serviço do caso de uso Enviar Telecomando. Sendo

assim, este procedimento não será detalhado para a segunda invocação de serviço, pois o mesmo já foi descrito com detalhes para a primeira invocação de serviço.

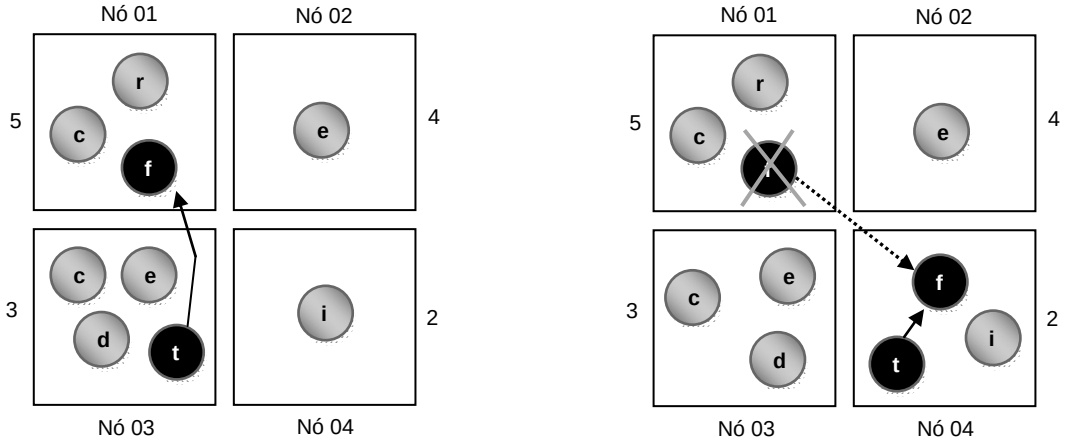
A arquitetura MABal atuou na segunda invocação de serviço de modo que o objeto servidor (objeto f), que no instante da invocação de serviço encontrava-se em um nó sobrecarregado (nó 01), foi migrado para o nó cliente que encontrava-se ocioso (nó 04).

Após realizada a migração, o objeto servidor foi executado no nó 04 e as medidas calculadas para este objeto foram:

- Aplicou-se a Equação 7.1 para calcular o tempo de execução do objeto servidor frame (f) e obteve-se o seguinte resultado: tt = 2.
- Aplicou-se a Equação 7.2 para definir se houve ou não acesso à rede na comunicação entre o objeto telecomando (t) e o objeto frame (f) e o resultado obtido foi: rs = 0.

A decisão de balanceamento da arquitetura MABal em executar o objeto servidor no nó 04 promoveu:

- o balanceamento da carga no sistema, pois não sobrecarregou ainda mais o nó 01;
- a redução no tempo de execução do objeto servidor, pois este executou em um nó ocioso do sistema (nó 04);
- a redução no tráfego de rede, pois o objeto servidor foi executado no nó onde o objeto cliente está localizado (nó 04). Portanto, não foi necessário transmitir dados pela rede na comunicação entre o objeto cliente e o objeto servidor.



parte (a) – Middleware CORBA

parte (b) – Arquitetura MABal: migração para nó cliente

Figura 7.9 – Segunda invocação de serviço do caso de uso enviar telecomando

7.5.5 Terceira Invocação de Serviço

Na terceira invocação de serviço do caso de uso Enviar Telecomando o objeto frame (f) desempenha o papel de objeto cliente, pois invoca o serviço do objeto estação (e), o qual realiza o serviço requisitado desempenhando o papel de objeto servidor.

7.5.5.1 Simulação da 3ª Invocação de Serviço Utilizando CORBA

A parte (a) da Figura 7.10 ilustra o resultado da simulação da terceira invocação de serviço aplicando-se o serviço de nomes da especificação CORBA. Diante da invocação de serviço ao objeto estação (e), o serviço CORBA atuou no sistema conforme encontra-se descrito a seguir.

Nota-se que existe mais de um objeto estação (e) nos nós do sistema, localizados nos nós 02 e 03. Como existem réplicas do objeto estação (e) no

sistema, o serviço CORBA selecionou uma das réplicas para desempenhar o papel de objeto servidor na invocação de serviço. Tendo em vista que o serviço de nomes CORBA realiza esta seleção de forma sequencial por meio de uma fila circular das réplicas, o objeto selecionado foi o primeiro da fila, ou seja, o objeto estação (e) localizado no nó 02. Esta política de seleção faz com que a cada nova invocação realizada ao objeto estação, CORBA selecione a próxima réplica da fila para atender ao serviço.

Os valores gerados para as medidas de tempo de execução e de acesso à rede do objeto estação (localizado no nó 02) foram:

- Aplicou-se a Equação 7.1 para calcular o tempo de execução do objeto estação (e) e obteve-se o seguinte resultado: tt = 15.
- Aplicou-se a Equação 7.2 para definir a existência ou não de acesso à rede na comunicação entre o objeto frame (f) e o objeto estação (e) e o resultado obtido foi: rs = 1.

7.5.5.2 Simulação da 3ª Invocação de Serviço Utilizando a Arquitetura MABal

A parte (b) da Figura 7.10 ilustra a simulação da terceira invocação de serviço do caso de uso Enviar Telecomando aplicando-se a arquitetura MABal.

Como existem réplicas do objeto estação (e) no sistema, é preciso selecionar a réplica que desempenhará o papel de objeto servidor na invocação de serviço. Para realizar esta seleção, a arquitetura MABal analisa o nível de carga dos nós onde estão localizadas as réplicas e seleciona a réplica localizada no nó que apresenta o menor nível de carga.

Durante a simulação da terceira invocação de serviço, os nós onde as réplicas do objeto estação (e) estão localizadas (nós 02 e 03) apresentaram níveis de carga 5 (muito carregado) e 4 (pouco carregado), respectivamente. Sendo

assim, a réplica selecionada foi o objeto estação localizado no nó 03, pois este nó apresentou nível de carga inferior ao nível de carga do nó 02.

Uma vez que a arquitetura MABal selecionou a réplica localizada no nó 03 para desempenhar o papel de objeto servidor da invocação de serviço, o próximo passo a ser realizado pela arquitetura MABal consiste em analisar a situação de carga dos nós do sistema para selecionar o nó mais adequado para executar o objeto servidor. Tendo em vista que o nó servidor (nó 03) encontra-se sobrecarregado (com nível de carga igual a 4), o objeto servidor (objeto e) não será executado neste nó e sim em outro nó do sistema.

Para realizar a escolha do nó mais adequado, o agente Gerenciador de Nó Servidor solicitou a atuação do agente Gerenciador de Nó Cliente (localizado no nó cliente, nó 04). Por sua vez, este agente solicitou ao agente Neural a classificação da carga do nó cliente. O agente Neural classificou o nó 04 em “muito carregado”, ou seja, nível 5. Uma vez que o nível de carga do nó cliente apresentou-se sobrecarregado, o objeto servidor não será executado neste nó e sim em outro nó do sistema.

Portanto, o nó servidor e o nó cliente não são os nós mais adequados do sistema para executar o objeto servidor. Sendo assim, o agente Gerenciador de Nó Cliente solicitou a atuação do agente Seletor de Nó, o qual selecionará o nó mais adequado do sistema para executar o objeto servidor. Este nó é denominado, neste trabalho de pesquisa, como sendo o terceiro nó, pois a seleção deste nó ocorre na terceira tentativa em encontrar o nó mais adequado para executar o objeto servidor.

É importante relembrar que a primeira tentativa em encontrar o nó mais adequado consiste em analisar o nó servidor, a segunda tentativa consiste em analisar o nó cliente e a terceira tentativa, um nó do sistema que satisfaça a

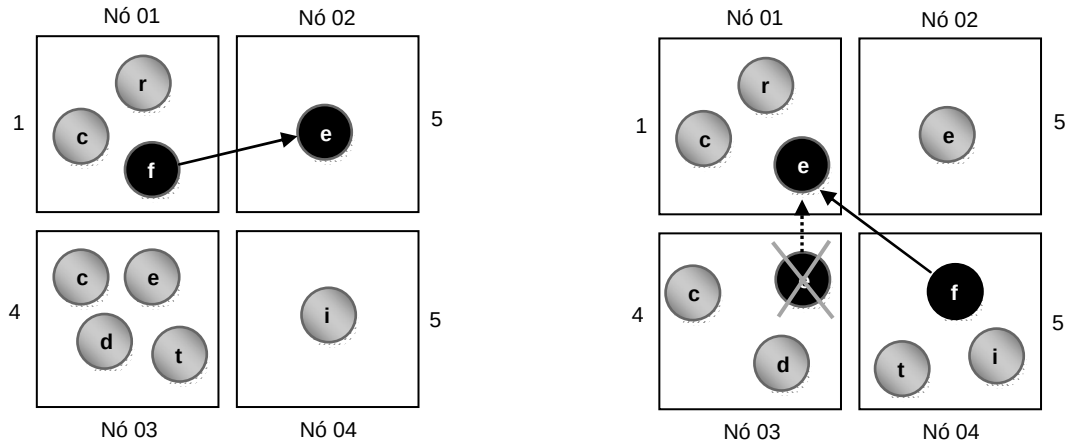
política de seleção de terceiro nó descrita na Figura 6.13. Esta figura ilustra o fluxo de tarefas do agente Seletor de Nó e descreve a política que o agente segue para realizar a seleção do terceiro nó.

Durante a simulação, o agente Seletor de Nó aplicou no sistema a política de Seleção de Terceiro Nó e o nó que satisfez as condições impostas pela política foi o nó 01 do sistema. Portanto, o nó 01 apresentou-se como sendo o nó mais adequado para executar o objeto servidor (objeto estação) pelos seguintes motivos: (i) o nó 01 possui o objeto receptor (objeto r), que é um objeto invocado pelo objeto servidor; e (ii) o nó 01 apresenta nível de carga ocioso (nível igual a 1, ou seja, “muito ocioso”).

No entanto, o objeto servidor não se encontra fisicamente localizado no nó 01. É necessário migrar o objeto servidor do nó 03 para o nó 01. Para realizar esta migração, o agente Gerenciador de Nó Cliente solicitou ao agente Migrador de Objetos do nó 01 que realizasse a migração.

Após realizada a migração, o objeto servidor (objeto e) foi executado no nó 01 do sistema. As medidas calculadas para o objeto servidor foram:

- Aplicou-se a Equação 7.1 para calcular o tempo de execução do objeto estação e obteve-se o seguinte resultado: tt = 3.
- Aplicou-se a Equação 7.2 para definir a existência ou não de acesso à rede na comunicação entre o objeto frame (f) e o objeto estação (e) e o resultado obtido foi: rs = 1.



parte (a) – Middleware CORBA

parte (b) – Arquitetura MABal: migração para terceiro nó

Figura 7.10 – Terceira invocação de serviço do caso de uso enviar telecomando

7.5.6 Quarta Invocação de Serviço

A Figura 7.11 ilustra a quarta invocação de serviço do caso de uso Enviar Telecomando, ou seja, a invocação do serviço do objeto receptor (r) pelo objeto estação (e). Nesta invocação, o objeto estação desempenha o papel de objeto cliente e o objeto receptor desempenha o papel de objeto servidor. A parte (a) da figura apresenta a simulação desta invocação aplicando-se o serviço de nomes da especificação CORBA e a parte (b) da figura apresenta a simulação desta invocação aplicando-se a arquitetura MABal. O procedimento realizado por ambas as abordagens encontra-se descrito a seguir.

7.5.6.1 Simulação da 4ª Invocação de Serviço Utilizando CORBA

O serviço de nomes CORBA atuou na quarta invocação de serviço da seguinte maneira:

- Como o objeto receptor (r) é único, ou seja, não existem réplicas deste objeto nos nós do sistema, o serviço de nomes CORBA selecionou o objeto receptor (r) como sendo o objeto servidor da quarta invocação de serviço. Portanto, o objeto receptor (r) foi executado no nó 01 do sistema.

Os valores das medidas coletadas nesta simulação foram:

- Aplicou-se a Equação 7.1 para calcular o tempo de execução do objeto receptor (r) e obteve-se o seguinte resultado: tt = 4.
- Aplicou-se a Equação 7.2 para definir a existência ou não de acesso à rede na comunicação entre o objeto estação (e) e o objeto receptor (r) e o resultado obtido foi: rs = 1.

7.5.6.2 Simulação da 4ª Invocação de Serviço Utilizando a Arquitetura MABal

A simulação da quarta invocação de serviço aplicando-se a arquitetura MABal encontra-se descrita a seguir.

Com a ocorrência desta invocação, primeiramente, o agente Gerenciador de Nó Servidor (localizado no nó servidor, ou seja, nó 01) solicitou ao agente Neural a classificação da carga do nó servidor. A classificação gerada pelo agente Neural para o nó 01 foi “pouco ocioso”, ou seja, nível 2. Uma vez que o nó servidor apresentou-se ocioso, o objeto servidor (objeto r) será executado neste nó.

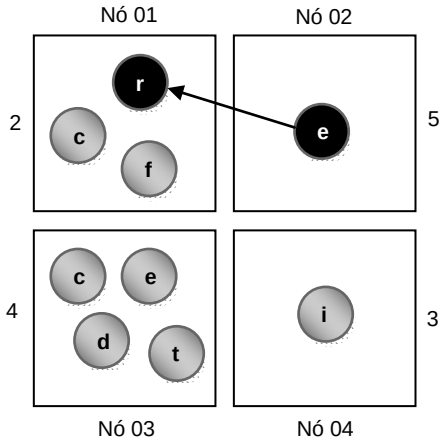
Sendo assim, não houve necessidade de realizar migração e o objeto servidor foi executado no nó 01, conforme ilustra a parte (b) da Figura 7.11.

As medidas calculadas para este objeto foram:

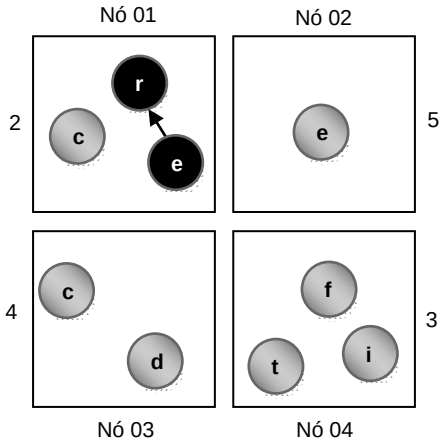
- Aplicou-se a Equação 7.1 para calcular o tempo de execução do objeto receptor (r) e obteve-se o seguinte resultado: tt = 4.
- Aplicou-se a Equação 7.2 para definir se houve ou não acesso à rede na comunicação entre o objeto estação e o objeto receptor e o resultado obtido foi: rs = 0.

É importante destacar que o valor da medida de acesso à rede pela invocação de serviço (rs) foi zero. Este valor foi obtido devido à migração que foi realizada anteriormente no momento da terceira invocação de serviço, na qual o objeto estação foi migrado para o nó onde estava localizado o objeto receptor.

Na terceira invocação de serviço, a decisão de balanceamento da arquitetura MABal foi a de migrar o objeto servidor para um terceiro nó. Esta decisão de balanceamento visa evitar o acesso à rede na próxima invocação de serviço. Este objetivo foi alcançado conforme mostrou o valor obtido para o acesso à rede (rs) na quarta invocação de serviço, na qual o objeto servidor (objeto receptor) executou no mesmo nó onde estava localizado o objeto cliente (objeto estação).



parte (a) – Middleware CORBA



parte (b) – Arquitetura MABal: não migrar objeto servidor

Figura 7.11 – Quarta invocação de serviço do caso de uso enviar telecomando

7.5.7 Quinta Invocação de Serviço

Na quinta invocação de serviço do caso de uso Enviar Telecomando o objeto receptor (r) invoca o serviço do objeto decodificador (d) e este provê o serviço requisitado ao cliente.

Realizou-se uma simulação desta invocação de serviço aplicando-se o serviço de nomes CORBA e realizou-se também uma simulação desta mesma invocação aplicando-se a arquitetura MABal. A parte (a) e a parte (b) da Figura 7.12 retratam estas simulações. O procedimento realizado por ambas as abordagens encontra-se descrito a seguir.

7.5.7.1 Simulação da 5ª Invocação de Serviço Utilizando CORBA

Uma vez que não existem réplicas do objeto decodificador (d) nos nós do sistema, o serviço de nomes CORBA selecionou como sendo o objeto servidor o objeto decodificador (d), o qual foi executado no nó 03 do sistema.

Os valores obtidos para as medidas de tempo de execução e de acesso à rede foram:

- Aplicou-se a Equação 7.1 para calcular o tempo de execução do objeto decodificador (d) e obteve-se o seguinte resultado: tt = 15.
- Aplicou-se a Equação 7.2 para definir a existência ou não de acesso à rede na comunicação entre o objeto receptor (r) e o objeto decodificador (d) e o resultado obtido foi: rs = 1.

7.5.7.2 Simulação da 5ª Invocação de Serviço Utilizando a Arquitetura MABal

Diante da invocação do serviço do objeto decodificador (d), localizado no nó 03, pelo objeto cliente receptor (r), localizado no nó 01, a arquitetura MABal analisou a situação de carga dos nós do sistema com o objetivo de selecionar o nó mais adequado para executar o objeto decoficador, o qual desempenha o papel de objeto servidor nesta invocação.

Tendo em vista que o nó servidor (nó 03) apresentou-se sobrecarregado (com nível de carga igual a 5), o objeto servidor (objeto d) não foi executado neste nó. Sendo assim, o próximo passo da arquitetura MABal foi analisar o nó cliente (nó 01). No entanto, o nível de carga do nó cliente também apresentou-se sobrecarregado, com nível de carga igual a 4.

Levando em consideração que o nó servidor e o nó cliente não apresentaram nível de carga adequado para executar o objeto servidor, a decisão de balanceamento da arquitetura MABal foi a de migrar o objeto servidor para um terceiro nó. Esta decisão de balanceamento também foi realizada por MABal durante a terceira invocação de serviço do caso de uso Enviar Telecomando. A atuação dos agentes da arquitetura MABal na realização desta decisão de balanceamento já foi descrita para a terceira invocação de serviço, por isso a

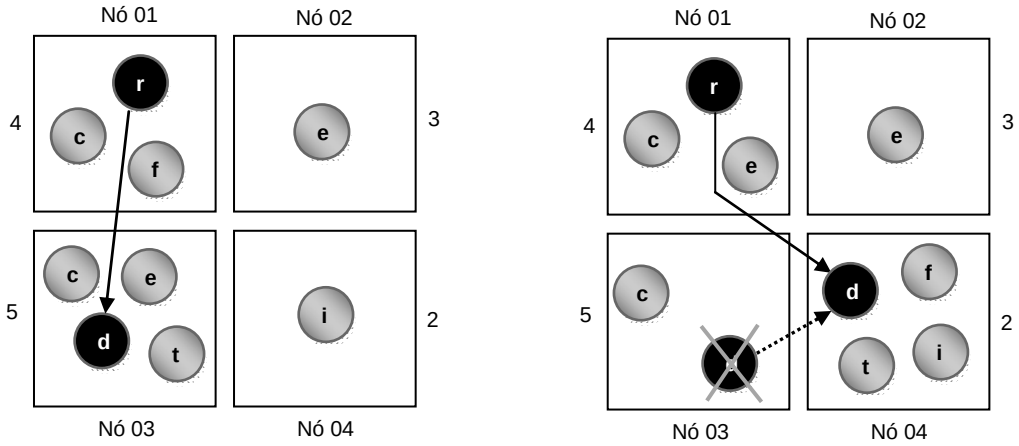
atuação dos agentes não será descrita em detalhes para a quinta invocação de serviço.

Para realizar a migração do objeto servidor para um terceiro nó, o primeiro passo foi selecionar o nó mais adequado para executar o objeto servidor. Para realizar esta seleção, a arquitetura MABal aplicou a política de Seleção de Terceiro nó. Esta política encontra-se descrita, em detalhes, na Figura 6.13. Aplicando-se esta política, a arquitetura MABal verificou que o objeto invocado pelo objeto servidor, que é o objeto compBordo (c) tinha suas réplicas localizadas em nós sobrecarregados (nó 01 e nó 03). Sendo assim, como não existiu no sistema um nó que possuísse um objeto invocado pelo objeto servidor e que apresentasse nível de carga ocioso, a arquitetura MABal selecionou o nó de menor nível de carga do sistema.

Portanto, o nó selecionado para executar o objeto servidor foi o nó 04, que apresentou nível de carga igual a 2 (“pouco ocioso”). O próximo passo foi migrar o objeto servidor do nó 03 para o nó 04 e executá-lo neste nó.

As medidas calculadas para o objeto servidor (objeto d) foram:

- Aplicou-se a Equação 7.1 para calcular o tempo de execução do objeto decodificador (d) e obteve-se o seguinte resultado: tt = 6.
- Aplicou-se a Equação 7.2 para definir a existência ou não de acesso à rede na comunicação entre o objeto receptor (r) e o objeto decodificador (d) e o resultado obtido foi: rs = 1.



parte (a) – Middleware CORBA

parte (b) – Arquitetura MABal: migração para terceiro nó

Figura 7.12 – Quinta invocação de serviço do caso de uso enviar telecomando

7.5.8 Sexta Invocação de Serviço

Na sexta invocação de serviço do caso de uso Enviar Telecomando o objeto decodificador (d) desempenha o papel de objeto cliente, pois invoca o serviço do objeto compBordo (c), o qual realiza o serviço requisitado desempenhando o papel de objeto servidor.

7.5.8.1 Simulação da 6ª Invocação de Serviço Utilizando CORBA

A parte (a) da Figura 7.13 ilustra o resultado da simulação da sexta invocação de serviço aplicando-se o serviço de nomes da especificação CORBA. Diante da invocação de serviço ao objeto compBordo (c), o serviço CORBA atuou no sistema conforme encontra-se descrito a seguir.

Nota-se que existe mais de um objeto compBordo (c) nos nós do sistema, localizados nos nós 01 e 03. Como existem réplicas do objeto compBordo (c) no sistema, o serviço CORBA selecionou uma das réplicas para desempenhar o papel de objeto servidor na invocação de serviço. Tendo em vista que o serviço de nomes CORBA realiza esta seleção de forma sequencial por meio

de uma fila circular das réplicas, o objeto selecionado foi o primeiro da fila, ou seja, o objeto compBordo (c) localizado no nó 01.

Os valores gerados para as medidas de tempo de execução e de acesso à rede do objeto compBordo (localizado no nó 01) foram:

- Aplicou-se a Equação 7.1 para calcular o tempo de execução do objeto compBordo (c) e obteve-se o seguinte resultado: tt = 5.
- Aplicou-se a Equação 7.2 para definir a existência ou não de acesso à rede na comunicação entre o objeto decodificador (d) e o objeto compBordo (c) e o resultado obtido foi: rs = 1.

7.5.8.2 Simulação da 6ª Invocação de Serviço Utilizando a Arquitetura MABal

A parte (b) da Figura 7.13 ilustra a simulação da sexta invocação de serviço do caso de uso Enviar Telecomando aplicando-se a arquitetura MABal.

Como existem réplicas do objeto compBordo (c) no sistema, é preciso selecionar a réplica que desempenhará o papel de objeto servidor na invocação de serviço. Para realizar esta seleção, a arquitetura MABal aplicou a política de Seleção de Réplica, a qual consiste em analisar o nível de carga dos nós onde estão localizadas as réplicas e selecionar a réplica localizada no nó que tenha o menor nível de carga. Aplicando-se esta política, a réplica selecionada foi o objeto compBordo localizado no nó 03, pois este nó apresentou nível de carga inferior ao nível de carga do nó 01.

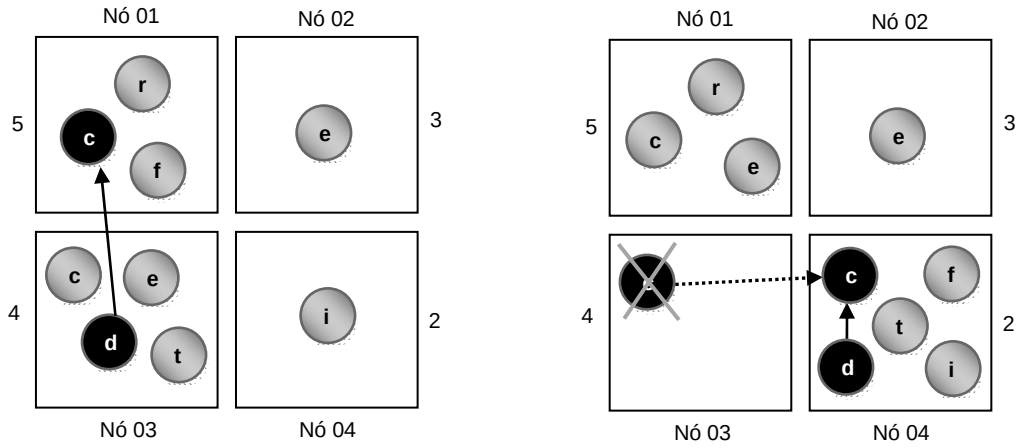
Uma vez que a arquitetura MABal selecionou a réplica localizada no nó 03 para desempenhar o papel de objeto servidor da invocação de serviço, o próximo passo consiste em analisar a situação de carga dos nós do sistema para selecionar o nó mais adequado para executar o objeto servidor.

Após realizada esta análise, a arquitetura MABal tomou a seguinte decisão de balanceamento: migrar o objeto servidor (objeto c) para o nó cliente (nó 04). O procedimento realizado para se tomar esta decisão e efetuá-la foi o mesmo aplicado à primeira e à segunda invocação de serviço do caso de uso Enviar Telecomando. Visto que este procedimento já foi descrito anteriormente para a primeira e para a segunda invocação de serviço, este procedimento não será descrito para a sexta invocação.

A parte (b) da Figura 7.13 ilustra a migração realizada. Portanto, o objeto servidor (objeto c), que no instante da invocação de serviço encontrava-se em um nó sobrecarregado (nó 03), foi migrado para o nó cliente que encontrava-se ocioso (nó 04).

Após realizada a migração, o objeto servidor foi executado no nó 04 e as medidas calculadas para este objeto foram:

- Aplicou-se a Equação 7.1 para calcular o tempo de execução do objeto compBordo e obteve-se o seguinte resultado: tt = 2.
- Aplicou-se a Equação 7.2 para definir se houve ou não acesso à rede na comunicação entre o objeto decodificador e o objeto compBordo e o resultado obtido foi: rs = 0.



parte (a) – Middleware CORBA

parte (b) – Arquitetura MABal: migração para nó cliente

Figura 7.13 – Sexta invocação de serviço do caso de uso enviar telecomando

Na seção seguinte apresenta-se a execução, pela ferramenta de simulação SimBal, das seis invocações de serviço que foram apresentadas nesta seção aplicando-se a arquitetura MABal.

7.6 Ferramenta SimBal – Simulação do Cenário Aplicando MABal

A Figura 7.14 apresenta a tela da ferramenta de simulação de balanceamento de carga SimBal contendo os resultados da simulação do caso de uso Enviar Telecomando aplicando-se a arquitetura MABal.

O procedimento que foi realizado pela arquitetura MABal no balanceamento de carga deste cenário de execução já foi descrito na Seção 7.5. Portanto, o mesmo não encontra-se descrito novamente nesta seção.

A Figura 7.14 apresenta o resultado da simulação, realizada pela ferramenta SimBal, de cada invocação de serviço do caso de uso Enviar Telecomando.

Este resultado é constituído pelas seguintes informações:

- O objeto servidor que foi executado, o qual encontra-se apresentado na coluna *Object*. Nesta coluna os objetos apresentam-se ordenados pela sequência de invocações de serviço para a realização do caso de uso;
- a decisão de balanceamento de carga realizada pela arquitetura MABal, a qual encontra-se apresentada na coluna *Balancing Decision*;
- o nó onde o objeto servidor estava localizado (nó de origem), o qual encontra-se apresentado na coluna *Source Node*;
- o nó para onde o objeto servidor foi migrado (nó receptor), o qual encontra-se apresentado na coluna *Final Node*;
- o tempo gasto para executar o objeto servidor no nó receptor (medida *tt*), o qual encontra-se apresentado na coluna S*pent Time*;
- a existência (valor 1) ou ausência (valor 0) de acesso à rede durante a comunicação entre o objeto cliente e o objeto servidor (medida *rs*), a qual encontra-se apresentada na coluna *Network Usage*.

Na parte inferior da Figura 7.14 no campo *Spent Time* apresenta-se o valor da medida denominada Tempo de Execução do Caso de Uso (*tc*). Esta medida expressa o tempo gasto para realizar um determinado caso de uso da aplicação e é calculada como sendo o somatório do tempo de execução de cada objeto (medida *tt*) do caso de uso. O resultado calculado pela ferramenta SimBal para esta medida durante a simulação do caso de uso Enviar Telecomando, aplicando-se a arquitetura MABal, foi *tc* = 19.

Além desta medida, apresenta-se na parte inferior da Figura 7.14 no campo *Network Usage* o valor da medida denominada Quantidade de Acesso à Rede pelos Objetos de um Caso de Uso (*qr*). Esta medida expressa quantas vezes houve acesso à rede durante a realização de um caso de uso e é calculada como sendo o somatório das medidas *rs* de todas as invocações de serviço de

um caso de uso. O cálculo realizado pela ferramenta SimBal para esta medida durante a simulação do caso de uso Enviar Telecomando resultou em *qr* = 2.

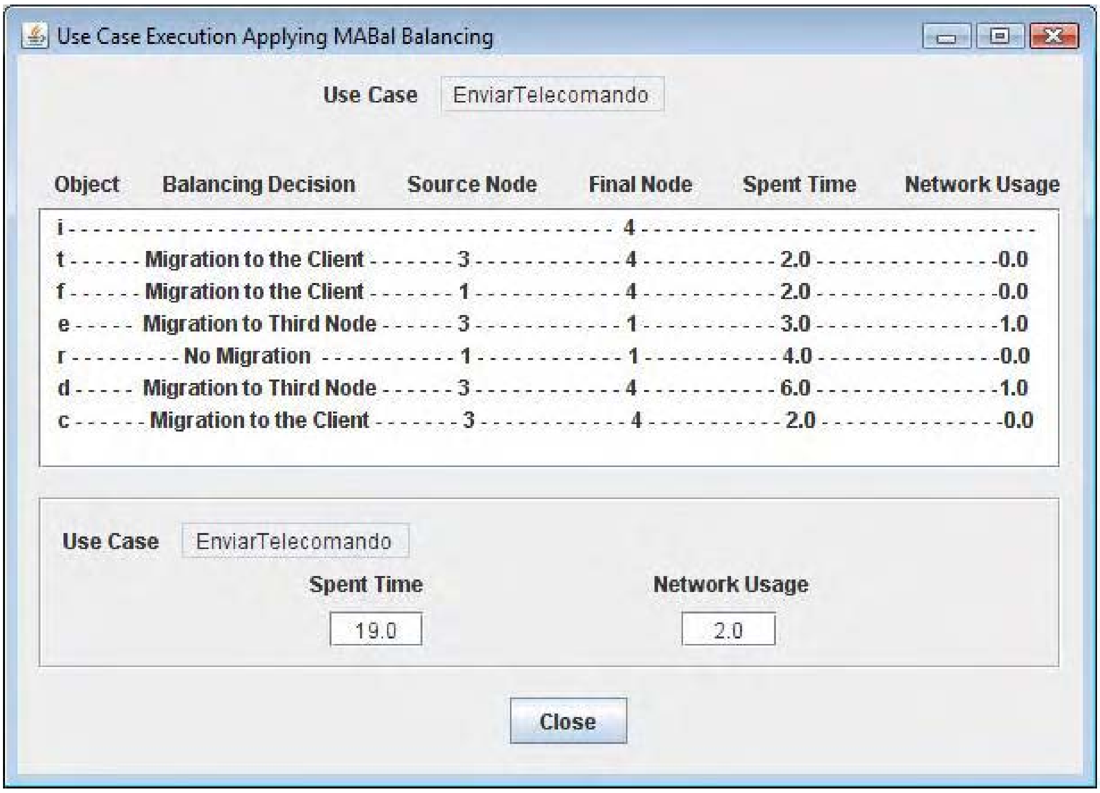


Figura 7.14 – Tela da ferramenta de simulação SimBal aplicando o balanceamento de carga da arquitetura MABal

7.7 Discussão sobre o Estudo de Caso – CORBA x MABal

Na Seção 7.5 foi apresentada a simulação do caso de uso Enviar Telecomando aplicando-se a arquitetura MABal e a simulação deste mesmo caso de uso aplicando-se o serviço de nomes da especificação CORBA.

Com o objetivo de comparar as duas abordagens – CORBA e MABal, as medidas *tc* e *qr*, que foram calculadas para a simulação do caso de uso aplicando-se a arquitetura MABal, foram também calculadas para a simulação do caso de uso aplicando-se o serviço de nomes CORBA. Os valores

resultantes destas medidas aplicando-se o serviço CORBA foram *tc* = 54 e *qr* = 6.

Os valores obtidos para as medidas *tc* e *qr* aplicando-se o serviço CORBA são maiores que os valores obtidos aplicando-se a abordagem MABal, os quais apresentaram os valores *tc* = 19 e *qr* = 2. Isto indica que, para o estudo de caso realizado, a arquitetura MABal mostrou-se favorável nos quesitos: tempo de execução do caso de uso e quantidade de acessos à rede durante a realização do caso de uso.

Tendo em vista que o objetivo de se realizar balanceamento de carga é promover equilíbrio de carga entre os nós do sistema e como conseqüência ter um aumento no desempenho da aplicação, a arquitetura MABal mostrou-se condizente com este objetivo, pois observou-se, na realização do estudo de caso, que houve uma redução no tempo de execução do caso de uso e uma redução no tráfego de rede para realizar o caso de uso.

Esta redução no tempo de execução do caso de uso ocorreu porque a arquitetura MABal levou em consideração a carga dos nós no momento de tomar as decisões de balanceamento e a redução no tráfego de rede ocorreu porque a arquitetura MABal realizou migrações do tipo “migrar o objeto servidor para o nó cliente” a fim de que ambos ficassem localizados fisicamente no mesmo nó (caso o nível de carga do nó cliente estivesse ocioso). Outro motivo para a redução no tráfego de rede foi a realização de migrações do tipo “migrar o objeto servidor para um terceiro nó”, pois este tipo de migração tem como objetivo fazer com que o objeto cliente e o objeto servidor da próxima invocação de serviço fiquem localizados no mesmo nó do sistema.

Com estas migrações, houve um menor uso da rede para realizar as comunicações entre os objetos clientes e os objetos servidores durante a

execução do caso de uso Enviar Telecomando. As migrações realizadas acabaram promovendo a mesma localização física dos objetos interface (i), telecomando (t), frame (f), decodificador (d) e compBordo (c) no nó 04, pois o mecanismo de balanceamento de carga da arquitetura MABal busca realocar os objetos de modo que os objetos que pertencem ao mesmo caso de uso, ou seja, os objetos que têm uma forte interação entre si, fiquem localizados fisicamente no mesmo nó, evitando acessos à rede durante a interação entre eles.

Este resultado pode ser visto na Figura 7.14 no campo *Final Node*, o qual apresenta o nó onde o objeto servidor foi executado. Observa-se na figura que o nó 04 possui uma maior concentração de objetos, o que mostra que a arquitetura MABal direcionou o balanceamento de forma a contemplar o caso de uso como sendo uma unidade.

Observou-se nas simulações realizadas que diante da existência de réplicas do objeto servidor no sistema, o serviço CORBA realizou a seleção da réplica que desempenharia o papel de objeto servidor sem levar em consideração a carga dos nós onde as réplicas estavam localizadas. Em contrapartida, a arquitetura MABal aplicou uma política de Seleção de Réplica guiada pelo nível de carga dos nós, selecionando a réplica localizada no nó mais ocioso.

Além disso, considerando que não existem réplicas do objeto servidor no sistema, o mecanismo de atuação do serviço CORBA não se preocupou em selecionar o nó mais adequado do sistema para executar o objeto servidor. Já a arquitetura MABal realizou migrações do objeto servidor com o objetivo de realocá-lo para o nó mais adequado do sistema, sendo que a escolha do nó mais adequado foi realizada levando-se em consideração: (i) a carga dos nós do sistema; e (ii) a localização física do objeto cliente com relação ao objeto

servidor. Devido a estes aspectos, as simulações do estudo de caso mostraram resultados satisfatórios para a arquitetura MABal.

8 CONCLUSÕES

Neste trabalho de pesquisa propõe-se uma arquitetura multi-agente distribuída de balanceamento de carga para aplicações de objetos distribuídos denominada arquitetura MABal. Esta arquitetura é constituída por um grupo de agentes que trabalham de forma cooperativa sobre aplicações distribuídas para oferecer uma solução de equilíbrio de carga a um sistema, ou seja, a um conjunto de nós de uma rede.

A arquitetura MABal atua, a cada invocação de serviço de um objeto da aplicação, de forma distribuída e dinâmica, redistribuindo a carga entre os nós do sistema. Esta redistribuição é realizada por meio de migrações e replicações de objetos localizados em nós sobrecarregados para nós ociosos, com base em um conjunto de políticas que estabelece como estas realocações de objetos devem ser conduzidas.

Os objetivos de se realocar objetos no sistema consistem em: (i) obter um maior equilíbrio de carga no sistema; e (ii) obter uma distribuição dos objetos no sistema que contemple a aglutinação dos objetos que pertencem a um mesmo caso de uso em um único nó do sistema de forma a reduzir o número de acessos à rede para a comunicação dos objetos entre si. O diferencial da arquitetura MABal em relação a outros serviços de balanceamento de carga propostos na literatura centra-se neste segundo objetivo.

Existem dois importantes aspectos que fortemente contribuem para que o serviço de balanceamento de carga da arquitetura MABal atue de modo orientado a caso de uso: (i) a ordem de prioridade de consulta aos nós para analisar os seus respectivos níveis de carga a fim de identificar o nó mais adequado a receber o objeto servidor e, portanto, a prover o serviço requisitado (1º: análise do nó servidor; 2º: análise do nó cliente; e 3º: análise de outro nó

do sistema que satisfaça à política de Seleção de Terceiro Nó); e (ii) a análise da quantidade de relacionamentos que um nó candidato a receber o objeto servidor possui com o objeto servidor, ou seja, durante a escolha do nó candidato a receber o objeto servidor dá-se preferência ao nó que contém objetos que sejam invocados pelo objeto servidor. Ambos estes aspectos conferem ao serviço de balanceamento de carga da arquitetura MABal a característica diferencial de aglutinar os objetos que colaboram para a realização de um mesmo caso de uso em um único nó do sistema.

A arquitetura MABal teve o seu comportamento simulado pela ferramenta de simulação SimBal, desenvolvida neste trabalho de pesquisa. O cenário de execução utilizado foi a execução de um caso de uso específico do Software Simulador de Satélites Distribuído. Os resultados desta simulação foram comparados aos resultados da simulação da execução do mesmo caso de uso aplicando-se o Serviço de Nomes da especificação CORBA.

Comparando-se os resultados das simulações de ambas as abordagens (MABal e CORBA) concluiu-se que, para o estudo de caso realizado, a arquitetura MABal apresentou menor tempo de execução do caso de uso e menor quantidade de acessos à rede para realizar o caso de uso. Assim, a arquitetura MABal mostra-se condizente com o objetivo de promover o balanceamento de carga de um sistema a fim de aumentar o desempenho das aplicações.

8.1 Contribuições

A proposta da arquitetura MABal provê um serviço de balanceamento de carga que reúne um conjunto de características que representam contribuições aos esforços de solução do problema de balanceamento de carga para aplicações distribuídas.

Portanto, a arquitetura MABal oferece as seguintes contribuições:

Serviço de Balanceamento de Carga Distribuído

Cada nó do sistema contém ambos os Serviços de Prevenção de Sobrecarga da arquitetura MABal: o Serviço de Prevenção de Sobrecarga Cliente e o Serviço de Prevenção de Sobrecarga Servidor. Esta distribuição de ambos os Serviços da arquitetura MABal em todos os nós do sistema faz-se necessária para tornar qualquer nó do sistema capaz de participar do processo de balanceamento de carga nos papéis de nó cliente e de nó servidor. A característica de distribuição da arquitetura MABal confere tolerância a falhas ao serviço de balanceamento de carga uma vez que, no caso de falha do nó servidor corrente, a arquitetura MABal invoca o Serviço de Prevenção de Sobrecarga Servidor de outro nó do sistema que contém uma réplica do objeto servidor a fim de executar o serviço requisitado neste nó.

Serviço de Balanceamento de Carga de Controle Descentralizado

Cada nó do sistema possui autonomia para ativar o serviço de balanceamento de carga uma vez que todos os nós possuem o Serviço de Prevenção de Sobrecarga Servidor. A descentralização também confere tolerância a falhas à arquitetura MABal por não existir um único nó controlador de todo o processo de balanceamento de carga, que indisponibilizaria o serviço de balanceamento de carga no sistema em caso de falha neste nó.

Serviço de Balanceamento de Carga Preventivo

A arquitetura MABal impede a execução de objetos em nós sobrecarregados ao invés de posteriormente balancear a carga do sistema por meio da realocação de objetos que já encontram-se em execução. Esta forma preventiva de atuação deve-se à ativação do serviço de balanceamento de carga da arquitetura MABal ser por demanda, ou seja, ocorrer a cada invocação de um objeto servidor (solicitação de serviço de um objeto cliente a um objeto servidor para a realização de um caso de uso). Este modo de ativação do serviço de balanceamento de carga permite com que a arquitetura MABal atue na escolha do nó mais adequado para executar o serviço requisitado a cada invocação de serviço entre os objetos de uma aplicação, prevenindo que o sistema fique em desequilíbrio de carga, devido à carga da aplicação distribuída, à medida que o sistema executa a aplicação.

Serviço de Balanceamento de Carga Dinâmico

O dinamismo de replicar ou migrar objetos servidores confere flexibilidade às aplicações distribuídas à medida que permitem que estas aplicações atendam à demanda de novas requisições de serviço sem comprometer o balanceamento de carga do sistema. Assim, a capacidade de criar uma réplica de um objeto servidor ou migrá-lo para um nó ocioso a fim de atender a uma nova invocação de serviço, quando o objeto servidor encontra-se fisicamente localizado em um nó sobrecarregado, consiste em uma decisão de balanceamento de carga que previne uma maior sobrecarga ao nó servidor, o que comprometeria o desempenho do sistema.

Serviço de Balanceamento de Carga Orientado a Caso de Uso

A arquitetura MABal leva em consideração as cargas dos nós no momento de executar um objeto servidor, de modo que este objeto não seja executado em um nó sobrecarregado do sistema. No entanto, existe uma segunda informação que norteia o processo de decisão de balanceamento de carga da arquitetura MABal: a localização física do objeto servidor em relação ao objeto cliente ou em relação aos objetos invocados pelo objeto servidor para a realização de um caso de uso. Estas informações de níveis de carga dos nós e das localizações físicas dos objetos constituem a base do processo de busca da arquitetura MABal pelo nó mais adequado para executar o objeto servidor, promovendo: (i) um menor tempo de execução do objeto servidor devido à escolha de um nó ocioso; e (ii) uma redução do tráfego de rede para a comunicação entre objetos que pertencem a um mesmo caso de uso devido à escolha de um nó, para receber o objeto servidor, que possua objetos que se comuniquem com o objeto servidor. A migração ou a replicação do objeto servidor quando efetuada conforme estas diretrizes de informação promovem o agrupamento de objetos que se comunicam entre si para a realização de um caso de uso em um único nó do sistema.

Serviço de Balanceamento de Carga Genérico

A arquitetura MABal é aplicável a sistemas heterogêneos que executem quaisquer aplicações de objetos distribuídos.

Serviço de Balanceamento de Carga com Suporte à Tomada de Decisão

Um agente neural, que possui a rede neural Perceptron de Múltiplas Camadas (MLP) como mecanismo de raciocínio, gera classificações de carga para os nós do sistema. Estas classificações auxiliam a arquitetura MABal no processo

de decisão de quando realizar realocações de objetos a fim de promover o balanceamento de carga orientado a caso de uso.

Serviço de Balanceamento de Carga com Modelagem Definida

O projeto da arquitetura MABal provê modelos de definição do problema de balanceamento de carga para aplicações de objetos distribuídos. Por meio da aplicação da metodologia de Engenharia de Software Orientada a Agentes MESSAGE ao problema de balanceamento de carga foi possível elaborar modelos que retratam diferentes visões deste problema (Visão Organizacional, Visão de Objetivos e Visão de Agentes). Estes modelos permitiram representar o problema de balanceamento de carga de forma bem definida.

Em relação a publicações, este trabalho de pesquisa gerou publicações nacionais e internacionais. Estas publicações encontram-se listadas no Apêndice A.

8.2 Trabalhos Futuros

A ferramenta SimBal realiza a simulação do funcionamento da arquitetura MABal em um sistema distribuído. Como trabalhos futuros pretende-se implementar a arquitetura MABal para esta, de fato, atuar em um sistema distribuído. A real implementação da arquitetura MABal requer a sua incorporação a um *middleware*, como por exemplo, a uma implementação de código aberto da especificação CORBA.

Esta incorporação faz-se necessária porque a arquitetura MABal depende da atuação de um ORB (*broker*) para realizar a busca pelo objeto servidor entre os nós da rede. O ORB provê serviços como a troca de mensagens entre cliente e servidor, serviços de segurança e transparência de localização, dentre outros.

O objeto cliente não precisa saber onde se localiza o objeto ao qual ele deseja invocar um serviço (objeto servidor). Ele precisa apenas conhecer o nome do objeto servidor. O ORB se responsabiliza por buscar o objeto servidor entre os nós da rede.

Um outro aspecto a ser considerado refere-se à utilização da arquitetura MABal no sistema onde opera o Software de Controle de Satélites do INPE (rede de computadores dedicada ao controle de múltiplos satélites). Diante da demanda de controle de novos satélites ao INPE, a otimização da utilização dos recursos computacionais apresenta-se como uma solução para que o Software de Controle de Satélites do INPE possa ser incrementado para atender às novas missões espaciais e continue a operar em condições computacionais (de capacidade de hardware) adequadas, evitando-se sobrecargas.

REFERÊNCIAS BIBLIOGRÁFICAS

BERNON, C.; GLEIZES, M. P.; PICARD, G.; GLIZE, P. The ADELFE methodology for an intranet system design. In: INTERNATIONAL BI-CONFERENCE WORKSHOP ON AGENT-ORIENTED INFORMATION SYSTEMS (AOIS), 4., 27-28 May 2002, Toronto, Ontario, Canada. **Proceedings...** Toronto: [s.n], 2002, p. 1-15.

BERNSTEIN, P. A., **Middleware:** a model for distributed system services, Communication of the ACM, v. 39, n. 2, p. 86-98, fev. 1996.

BRAGA, A.; CARVALHO, A.; LUDERMIR, T. B. **Redes neurais artificiais -** teorias e aplicações. Rio de Janeiro: LTC, 2000. 262 p.

BRANCO, K., R., L., J., C. **Índice de carga e desempenho em ambientes paralelos/distribuídos** – modelagem e métricas. Universidade de São Paulo (USP) – ICMC, São Carlos-SP, outubro, 2002. Relatório FAPESP Doutorado.

BRESCIANI, P.; GIORGINI, P.; GIUNCHIGLIA, F.; MYLOPOLOUS, J.; PERINI, A. Tropos: An agent-oriented software development methodology. **Autonomous Agents and Multi-Agent Systems**, v. 8, n. 3, p. 203-236, 2004.

CAIRE, G.; COULIER, W.; GARIJO, F.; GOMEZ, J.; PAVON, J.; LEAL, F.; CHAINHO, P.; KEARNEY, P.; STARK, J.; EVANS, R.; MASSONET, P. Agent-oriented analysis using MESSAGE/UML. **Lecture Notes in Computer Science**. v. 2222, p. 119-135, 2002. Agent-oriented software engineering II (AOSE-2001), Montreal, Canada, 29 May 2001.

COLEMAN, D.; ARNOLD, P.; BODOFF, S.; DOLLIN, C.; GILCHRIST, H. **Object oriented development**: the fusion method. Englewood Cliffs, New Jersey: Prentice Hall, 1994.

COSSENTINO, M. From requirements to code with the PASSI methodology. In: B. Henderson-Sellers & P. Giorgini (Eds.). **Agent-oriented methodologies**. chapter 4. Hershey, PA: Idea Group, 2005.

COULOURIS, G.; DOLLIMORE, J.; KINBERG. **Sistemas distribuídos**: conceitos e projetos. 4a. ed. Bookmann, 2007.

DAM, K. H.; WINIKOFF, M. Comparing agent-oriented methodologies. In: INTERNATIONAL BI-CONFERENCE WORKSHOP ON AGENT-ORIENTED INFORMATION SYSTEMS, 5., 2003, Melbourne, Australia. **Proceedings...** Melbourne, Australia: [s.n], 2003. p. 52-59.

DEBENHAM, J.; HENDERSON-SELLERS, B. Designing agent-based process systems - Extending the OPEN Process Framework. In: _____. **Intelligent agent software engineering**. Hershey, PA: Idea Group Publishing, 2003, chapter 8. p. 160-190.

EL-ABD, A.; EL-BENDARY, M. Neural-based selection and location policies for dynamic load balancing in distributed computing systems. In: IASTED INTERNATIONAL CONFERENCE ON MODELING AND SIMULATION, 13-16 Mai 1998, Pensylvania, USA. **Proceedings...** Pensylvania, USA: [s.n], 1998.

ELÄSSER, R., MONIEN, B., PREIS, R. Diffusive load balancing schemes on heterogeneous networks. In: ACM SYMPOSIUM ON PARALLEL ALGORITHMS AND ARCHITECTURES, 12., 9 - 13 Jul 2000, Bar Harbor, ME USA. **Proceedings...** Bar Harbor, ME USA: [s.n], 2000.

FEBER, J. **Multi-agent systems:** an introduction to distributed artificial intelligence. London: Addison-Wesley (ed). 1999. 509 p. (ISBN 0-201-36048-9)

FERRARI, D.; ZHOU, S. An empirical investigation of load indices for load balancing applications. In: PERFORMANCE'87- INTEL SYMPOSIUM ON COMPUTER PERFORMANCE MODELING, MEASUREMENT AND EVALUATION, 12., 1987, North Holland. **Proceedings...** North Holland: [s.n], 1987. p. 515-528.

FERREIRA, M. G. V. **Uma arquitetura flexível e dinâmica para objetos distribuídos aplicada ao software de controle de satélites**. 2001. 244 p. (INPE-8602-TDI/787). Tese (Doutorado em Computação Aplicada) - Instituto Nacional de Pesquisas

Espaciais, São José dos Campos. 2001. Disponível em: <http://urlib.net/dpi.inpe.br/lise/2003/01.16.09.56>. Acesso em: 30 jan. 2009.

FOWLER, M., KOBRYN, C., BOOCH, G. **UML essencial**. 3ª ed. Porto Alegre: Bookman, 2005. 160 p., ISBN 8536304545.

FRANKLIN, S.; GRAESSER, A. Is it an agent, or just a program? A taxonomy for autonomous agents. In: INTERNATIONAL WORKSHOP ON AGENT THEORIES, ARCHITECTURES, AND LANGUAGES (ATAL), 3., 1996, Berlin, Germany. **Proceedings...** Berlin, Germany: Springer-Verlag, 1996. (ISBN 3-540-62507-0).

HAYKIN, S. **Redes neurais:** princípios e prática. 2ª ed. Porto Alegre: Bookman, 2001. 900 p., ISBN 85-7307-718-2.

HENDERSON-SELLERS, B.; GIORGINI, P. **Agent-oriented methodologies**. London, United Kingdom: Idea Group Publishing, 2005. 413 p. (ISBN 1-59140-587-4).

HERNANDEZ, E. D. M. **Inteligência computacional e redes neurais em engenharia elétrica**. PSI-EPUSP, 2003. PSI-2222, Práticas de Eletricidade e Eletrônica II.

IGLESIAS, C. A.; GARIJO, M.; GONZALEZ, J. C.; VELASCO, J. R. Analysis and design of multi-agent systems using MAS-CommonKADS. **Lecture Notes in Artificial Intelligence**, v. 1365, p. 313-326, 1998. In: INTELLIGENT AGENTS: AGENT THEORIES, ARCHITECTURES, AND LANGUAGES, 4., 1998, Berlin.

KINNY, D.; GEORGEFF, M.; RAO, A. A methodology and modelling techniques for systems of BDI agents. In: EUROPEAN WORKSHOP ON MODELLING AUTONOMOUS AGENTS IN A MULTI-AGENT WORLD (MAAMAW), 7., 1996. **Proceedings...** Springer-Verlag, 1996. p. 56-71.

KUNZ, T. The influence of different workload descriptions on a heuristic load balancing scheme. **IEEE Transactions on Software Engineering**, v.17, n.7, p.725-730, julho, 1991.

LAWRENCE, J. **Introduction to neural networks and expert systems**. Nevada City, Califórnia: California Scientific Software, 1992.

LIEBERMAN, H. Autonomous interface agents. In: CONFERENCE ON HUMAN FACTORS IN COMPUTING SYSTEMS, 1997, Atlanta, U.S.A. **Proceedings…** Atlanta: ACM/SIGCHI, 1997. Disponível em: <http://www.sigchi.org/chi97/proceedings/paper/hl.htm>. Acesso em: 03 fev. 2009.

MICROSOFT CORPORATION (Microsoft). **DCOM architecture**. Desenvolvido por Markus Horstmann e Mary Kirtland. 2009. Disponível em: <http://msdn.microsoft.com/en-us/library/ms809311.aspx>. Acesso em: 12 jan 2009.

MILGROM. E. **MESSAGE:** methodology for engineering systems of software agents. Project P907 Report, EDIN 0215-0907, EURESCOM – European Institute for Research and Strategic Studies in Telecommunications GmbH, Sept., 2001. Disponível em: http://www.eurescom.de/~public-webspace/P900-series/P907/index.htm. Acesso em: 03 fev. 2009.

NOGUEIRA, M. L. B.; YAMIN, A. C.; VARGAS, P. K.; GEYER, C. F. R. Balanceamento de carga em sistemas distribuídos: uma proposta de ambiente para avaliação. In: CONFERÊNCIA LATINO AMERICANA DE INFORMÁTICA – CLEI, 27., set 2001, Merida, Venezuela. **Proceedings...** Merida, Venezuela: [s.n], 2001.

OBJECT MANAGEMENT GROUP (OMG). **CORBA 3.0 specification**. Desenvolvido por Object Management Group. Nov 2008. Disponível em: <http://www.omg.org/technology/documents/formal/corba_2.htm>. Acesso em: 08 jan 2009.

ODELL, J.; PARUNAK, H. V. D.; BAUER, B. Extending UML for agents. In: NATIONAL CONFERENCE ON ARTIFICIAL INTELLIGENCE, 17., 2000, Austin, Texas. **Proceedings…** Austin, Texas: AAAI Press, 2000. p. 3-17. ISBN 978-0-262-51112-4.

PADGHAM, L.; WINIKOFF, M. Prometheus: a methodology for developing intelligent agents. **Lecture Notes in Computer Science (LNCS)**, v. 2585, p. 174-185, 2003. Special issue on the Third International Workshop on Agent-Oriented Software Engineering (AOSE'02). ISBN 978-3-540-00713-5.

PAVÓN, J.; GOMEZ-SANZ, J.; FUENTES, R. The INGENIAS methodology and tools. In: ___. **Agent-oriented methodologies**. Hershey, PA: Idea Group, 2005, chapter 4.

PERINI A.; SUSI A.; GIUNCHIGLIA F. Coordination specification in multi-agent systems: from requirements to architecture with the Tropos methodology. In: INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING AND KNOWLEDGE ENGINEERING (SEKE), 14., Jul 2002, Ischia, Italy. **Proceedings…** Ischia, Italy: ACM Press, 2002. p. 51-54.

REZENDE, S. O. **Sistemas inteligentes:** fundamentos e aplicações. Barueri, SP: editora Manole, 2003. ISBN 85-204-1683-7.

RUMBAUGH, J.; BLAHA, M.; PREMERLANI, W.; EDDY, F.; LORENSEN, W. **Object-oriented modeling and design**. Englewood Cliffs, New Jersey: Prentice-Hall, 1991.

RUMBAUGH, J.; JACOBSON, I.; BOOCH, G. **The unified modeling language reference manual**. 2. ed., Addison-Wesley, 2004.

RUSSELL, S.; NORVIG, P. **Inteligência artificial**. 2. ed., São Paulo: Editora Campus, 2004. 1040 p. ISBN (8535211772).

SCHLEMER, E. **A scheduling solution for DPC++**. 2002. Dissertação em Ciência da Computação – Universidade Federal do Rio Grande do Sul (UFRGS), Porto Alegre-RS, 2002.

SHIVARATRI, N. G.; KREUGER, P.; SINGHAL, M. Load distribution for locally distributed systems. **Computer**, v. 25, p. 33-44, dez 1992.

STURM, A.; SHEHORY, O. A framework for evaluating agent-oriented methodologies. In: INTERNATIONAL BI-CONFERENCE WORKSHOP ON AGENT-ORIENTED INFORMATION SYSTEMS (AOIS), 5., 14 Jul 2003, Melbourne, Australia. **Proceedings...** Melbourne, Australia: [s.n], 2003. p. 60-67.

SUN MICROSYSTEMS (Sun). **Remote method invocation home**. Desenvolvido por Sun Microsystems. 2009. Disponível em: <http://java.sun.com/javase/technologies/core/basic/rmi/index.jsp>. Acesso em: 09 jan 2009.

SURI, S., T´OTH, C. D., AND ZHOU, Y. Selfish load balancing and atomic congestion games. **Algorithmica**, Springer-Verlag, New York, v. 47, n. 1, p. 79 – 96, 2007.

SYCARA, K. P. The many faces of agents. **Artificial Intelligence Magazine**, v. 19, n. 2, p. 11-12, 1998.

TANENBAUM, A. S.; STEEN, M. V. **Distributed systems:** principles and paradigms. Prentice Hall, 2002, 803 p.

TAVETER, K.; WAGNER, G. Towards radical agent-oriented software engineering processes based on AOR modelling. In: HENDERSON-SELLERS, B.; GIORGINI, P. **Agent-oriented methodologies**. Hershey, PA: Idea Group, chapter 10, 2005.

TRAN, Q. N.; LOW, G.; WILLIAMS, M. A. A feature analysis framework for evaluating multi-agent system development methodologies. In: INTERNATIONAL SYMPOSIUM ON METHODOLOGIES FOR INTELLIGENT SYSTEMS (ISMIS), 14., 28-31 out 2003, Maebashi, Japan. **Proceedings...** Maebashi, Japan: Springer-Verlag, 2003. p. 613-617.

WAGNER, G. The agent-object relationship metamodel: towards a unified view of state and behaviour. **Information Systems**, v. 28, n. 5, p. 475-504, 2003.

WOOD, M.; DELOACH, S. A. An overview of the multiagent systems engineering methodology. **Lecture Notes in Computer Science (LNCS)**, v. 1957, p. 207-222,

Springer-Verlag, 2001. Special Issue on the First International Workshop on Agent-Oriented Software Engineering (AOSE 2000).

WOOLDRIDGE, M.; JENNINGS, N. R.; KINNY, D. The Gaia methodology for agent-oriented analysis and design. **Journal Autonomous Agents and Multi-Agent Systems**, v. 3, p. 285-312, 2000.

YANG, J.; JIZHOU, S.; ZUNCE, W. Load balance in a new group communication system for the WAN. In: CANADIAN CONFERENCE ON ELECTRICAL AND COMPUTER ENGINEERING (CCECE), 4-7 mai 2003, Montreal, Canada. **Proceedings...** Montreal, Canada: [s.n], 2003. p. 931-934.

YU, E. **Modelling strategic relationships for process reengineering**. PhD Thesis - University of Toronto, Department of Computer Science, Toronto, 1995.

ZAMBONELLI, F.; JENNINGS, N.; WOOLDRIDGE, M. Developing multiagent systems: the Gaia methodology. **ACM Transactions on Software Engineering and Methodology**, v. 12, n. 3, p. 317-370, 2003.

ANEXO A – PUBLICAÇÕES DESTE TRABALHO DE PESQUISA

A seguir, apresentam-se as publicações que foram geradas no decorrer do desenvolvimento deste trabalho de pesquisa.

CARNIELLO, A.; CARNIELLO, A.; FERREIRA, M. G. V.; SILVA, J. D. S. A Better Performance to INPE Satellite Control Software. In: 10th International Conference on Space Operations – 10th SpaceOps, 2008, Alemanha. Proceedings of the 10th International Conference on Space Operations. Reston, Virginia: American Institute of Aeronautics and Astronautics, Inc., 2008.

CARNIELLO, A.; CARNIELLO, A.; FERREIRA, M. G. V.; SILVA, J. D. S. Agentes Cooperativos para o Problema de Balanceamento de Carga de Aplicações de Objetos Distribuídos. Revista Integração (São Paulo), 2007.

CARNIELLO, A.; CARNIELLO, A.; FERREIRA, M. G. V.; SILVA, J. D. S. Satellite Control Software Load Balancing Under the Distributed Object Paradigm. In: 24th AIAA International Communications Satellite Systems Conference – 24th ICSSC, 2006, San Diego, California. Proceedings of the 24th AIAA International Communications Satellite Systems Conference. Reston, Virginia: American Institute of Aeronautics and Astronautics, Inc., 2006. v. 11.

CARNIELLO, A.; CARNIELLO, A.; FERREIRA, M. G. V.; SILVA, J. D. S. A Multi-agent Based Load Balancing Architecture to Support INPE Satellite Control Software. In: 9th International Conference on Space Operations – 9th SpaceOps, 2006, Roma. Proceedings of the 9th International Conference on Space Operations. Reston, Virginia: American Institute of Aeronautics and Astronautics, Inc., 2006.

CARNIELLO, A.; CARNIELLO, A.; FERREIRA, M. G. V.; SILVA, J. D. S. A Multi-agent Load Balancing Architecture for Distributed Object Applications. In: The 2006 International Conference on Parallel and Distributed Processing Techniques and Applications – PDPTA 2006, 2006, Las Vegas, Nevada. Proceedings of the 2006 International Conference on Parallel and Distributed Processing Techniques and Applications, 2006.

CARNIELLO, A.; FERREIRA, M. G. V.; SILVA, J. D. S.; DIAS, L. R. Uma Arquitetura Multi-agente de Balanceamento de Carga para Aplicações de Objetos Distribuídos Utilizando Redes Neurais Artificiais. In: V Workshop dos Cursos de Computação Aplicada do INPE – V Worcap, 2005, São José dos Campos. Anais do V Workshop dos Cursos de Computação Aplicada do INPE, 2005.

CARNIELLO, A.; FERREIRA, M. G. V.; SILVA, J. D. S. Uma Arquitetura de Balanceamento de Carga em Sistemas Distribuídos Utilizando Redes Neurais Artificiais. In: IV Workshop dos Cursos de Computação Aplicada do INPE – IV Worcap, 2004, São José dos Campos. Anais do IV Workshop dos Cursos de Computação Aplicada do INPE, 2004.

PUBLICAÇÕES TÉCNICO-CIENTÍFICAS EDITADAS PELO INPE

Teses e Dissertações (TDI)

Teses e Dissertações apresentadas nos Cursos de Pós-Graduação do INPE.

Notas Técnico-Científicas (NTC)

Incluem resultados preliminares de pesquisa, descrição de equipamentos, descrição e ou documentação de programa de computador, descrição de sistemas e experimentos, apresentação de testes, dados, atlas, e documentação de projetos de engenharia.

Propostas e Relatórios de Projetos (PRP)

São propostas de projetos técnico-científicos e relatórios de acompanhamento de projetos, atividades e convênios.

Publicações Seriadas

São os seriados técnico-científicos: boletins, periódicos, anuários e anais de eventos (simpósios e congressos). Constam destas publicações o Internacional Standard Serial Number (ISSN), que é um código único e definitivo para identificação de títulos de seriados.

Pré-publicações (PRE)

Todos os artigos publicados em periódicos, anais e como capítulos de livros.

Manuais Técnicos (MAN)

São publicações de caráter técnico que incluem normas, procedimentos, instruções e orientações.

Relatórios de Pesquisa (RPQ)

Reportam resultados ou progressos de pesquisas tanto de natureza técnica quanto científica, cujo nível seja compatível com o de uma publicação em periódico nacional ou internacional.

Publicações Didáticas (PUD)

Incluem apostilas, notas de aula e manuais didáticos.

Programas de Computador (PDC)

São a seqüência de instruções ou códigos, expressos em uma linguagem de programação compilada ou interpretada, a ser executada por um computador para alcançar um determinado objetivo. São aceitos tanto programas fonte quanto executáveis.