Universidade Federal do Ceará
Departamento de Engenharia de Teleinformática
Programa de Pós-Graduação em Engenharia de Teleinformática

Arquitetura de Reconfiguração Dinâmica de Recursos em Clusters de Servidores Web Utilizando Sistemas Multiagentes

Carla Katarina de Monteiro Marques

Fortaleza – Ceará
Junho 2010

Universidade Federal do Ceará
Departamento de Engenharia de Teleinformática
Programa de Pós-Graduação em Engenharia de Teleinformática

Arquitetura de Reconfiguração Dinâmica de Recursos em Clusters de Servidores Web Utilizando Sistemas Multiagentes

Autor
Carla Katarina de Monteiro Marques

Orientador
Giovanni Cordeiro Barroso
Co-orientador
Antônio de Barros Serra

Tese de Doutorado apresentada à Coordenação do Curso de Pós-Graduação em Engenharia de Teleinformática da Universidade Federal do Ceará como parte dos requisitos para obtenção do grau de ***Doutor em Engenharia de Teleinformática****.*

Fortaleza – Ceará
Junho 2010

Carla Katarina de Monteiro Marques

Arquitetura de Reconfiguração Dinâmica de Recursos em Clusters de Servidores Web Utilizando Sistemas Multiagentes

Esta Tese foi julgada adequada para a obtenção do título de Doutor em Engenharia de Teleinformática e aprovada em sua forma final pelo Programa de Pós-Graduação em Engenharia de Teleinformática da Universidade Federal do Ceará.

Nome do Aluno

Banca Examinadora:

Prof. Dr. Giovanni Cordeiro Barroso
Orientador

Prof. Dr. Antônio de Barros Serra
Co-orientador

Prof. Dr. Celso Maciel da Costa
Pontíficia Universidade Católica do Rio Grande do Sul

Prof. Dr. André Luís Vasconcelos Coelho
Universidade de Fortaleza

Prof. Dr. Pedro Fernandes Ribeiro Neto
Universidade do Estado do Rio Grande do Norte

Prof. Dr. José Marques Soares
Universidade Federal do Ceará

Prof. Dr. José Neuman de Sousa
Universidade Federal do Ceará

Fortaleza, 11 de junho de 2010

Resumo

O planejamento da alocação de recursos disponíveis em *clusters* de servidores Web é realizado, geralmente, pelos administradores do ambiente computacional baseado nos diferentes picos de carga e na observação da necessidade dos recursos em situações anteriores. A presença do administrador do *cluster* seria necessária para realizar o planejamento dos recursos da plataforma de servidores Web, já que é ele quem possui o conhecimento básico para a alocação dos recursos iniciais. Como a Internet é muito dinâmica no que diz respeito ao uso de recursos, esta tarefa pode se tornar crítica e ineficaz se realizada de forma manual. Neste trabalho é descrita a Arquitetura DARC (*Dynamic Architecture for Reconfiguration of Web servers Clusters*), uma arquitetura de reconfiguração dinâmica de servidores de *clusters* Web utilizando um sistema multiagente. Os agentes dessa arquitetura exercerão ações sobre o mecanismo de gerência de QoS (Qualidade de Serviço), a fim de melhorar a eficiência da utilização dos recursos disponíveis, sendo também capazes de tomar decisões dinâmicas a respeito da taxa de utilização dos recursos e da carga do sistema. Nesta tese são abordadas a concepção, a especificação, a modelagem em redes de Petri (Coloridas e Estocástica) e a implementação desta arquitetura na linguagem Java, utilizando Java RMI (*Remote Method Invocation*) para comunicação dos agentes. Vários experimentos e simulações são apresentados comprovando a eficácia da arquitetura proposta.

Palavras-chave: Reconfiguração dinâmica, sistemas multiagentes, servidores web.

Abstract

The resource allocation planning in a web server cluster is usually accomplished nowadays by the administrators of the computing environment based on different peak loads and the observation of the need of resources in previous situations. Once the Internet is quite dynamics as far as the use of resources is concerned, such task may be considered critical and ineffective if accomplished manually. This thesis describes the DARC architecture (Dynamic Reconfiguration Architecture for Clusters of Web Servers), an architecture for dynamic reconfiguration of servers clusters Web using a multiagent system. The agents carry out actions over the mechanism of QoS (Quality of Services) management in order to improve the efficiency of the use of available resources and they will be able to make dynamic decisions concerning the rate of utilization of resources and the load of the system. The multiagent system of DARC architecture will perform the necessary support to changes in available resources in the Web servers platform, without interference from the administrator. The presence of this would be needed in resource planning platform for Web servers, as it is he/she who has the initial knowledge for the allocation of resources. This thesis addressed the conception, specification, modeling Petri nets (Colored and Stochastic) and the implementation of this architecture in Java using RMI (Remote Method Invocation) for communication of agents. Several experiments and simulations which prove the effectiveness of the proposed architecture are presented.

Keywords: Dynamic reconfiguration, multiagent systems, web servers.

Dedico este trabalho a Deus.

Agradecimentos

Em primeiro lugar, eu gostaria de agradecer ao meu marido Henrique Jorge, aos meus filhos Pedro e Juliana.

Aos meus pais, Odivan Nunes Marques e Maria Geralda de Monteiro Marques.

Às minhas irmãs Yáskara, Socorro Angélica e ao meu irmão Odivan Nunes Marques Júnior.

Aos meus orientadores Professores Giovanni Cordeiro Barroso e Antônio de Barros Serra pelo apoio, simpatia de sempre, e incentivo para a realização deste trabalho

A Capes, à UERN e ao IFRN, pelos auxílios concedidos, sem os quais este trabalho não poderia ter sido realizado.

Aos meus colegas da UFC, Isabel, Aparecida e Carlos Henrique (Caíque).

Em especial ao professor Sergio Ilarri Artigas, da Universidade de Zaragoza, que muito me ajudou.

Por fim, agradeço a Deus, pela força concedida nos momentos mais difíceis.

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Símbolos

ρ_{ki}	carga estimada do *cluster* em um período k
R_{emk}	valor do coeficiente de reatividade alcançado pelo *cluster* de uma classe
R_{max}	valor-limite que o coeficiente de reatividade do *cluster* da classe pode assumir para que o mesmo trabalhe no modo compartilhado
R_{ac}	valor-limite que o coeficiente de reatividade do *cluster* da classe pode assumir para que o mesmo trabalhe no modo exclusivo
γ_{low}	parâmetros limite inferior da carga do *cluster* que controlam a atuação dos agentes de execução
γ_{high}	parâmetros limite superior da carga do *cluster* que controlam a atuação dos agentes de execução
δ_1	parâmetro de atualização dos *thresholds* do cluster

Lista de Abreviaturas e Acrônimos

AMBLE	*Awareness Model for Balancing the Load in Collaborative Grid Environment*
CAMBLE	*Cooperative AMBLE*
CPNTools	*Computer Tool for Coloured Petri Nets*
CPU	*Central Processing Unit*
CRM	*Cluster Resource Manager*
CSM	*Corel Scenario Model*
DARC	*Dynamic Architecture for Reconfiguration of Web Servers Clusters*
DLBDMA	*Dynamic Load Balancing Distributed using Mobile Agents*
E/S	Entrada e Saída
GRM	*Global Resource Manager*
GreatSPN	*Great Stochastic Petri Nets*
GSPN	*Generalized Stochastic Petri Nets*
IDL	*Interface Definition Language*
J2EE	*Java 2 Platform, Enterprise Edition*
JOnAS	*Java Open Source Application Server*
LGA	*Load Gathering Agent*
MA	*Monitor Agent*
MALB	*Mobile Agent Based Load Balancing*
PMADE	*Platform for Mobile Agent Distribution and Execution*
PSO	*Particle Swarm Optimization*
QoS	*Quality of Service*
RdP	Rede de Petri
RFID	*Radio Frequency Identification*
RLBA	*RFID Load Balancing Agent*
RMI	*Remote Method Invocation*

SLA	*Service Level Agreements*
SMA	Sistemas Multiagentes
SPE	*Software Performance Engineering*
TSK	*Takagi-Sugeno-Kang*
UML	*Unified Modeling Language*
UML-SPT	*Standard UML Profile for Schedulability, Performance and Time Specification*
VA	Visão Artificial
WS-DSAC	*Web Servers - DiffServ Admission Control*
WWW	*World Wide Web*

Capítulo 1

Introdução

O crescimento recente da Internet está relacionado à expansão da *World Wide Web* e ao crescimento dos provedores comerciais. O tráfego na rede aumentou em algumas ordens de grandeza, numa tendência que se mantém até hoje. Uma parte significativa deste crescimento pode ser creditada ao aumento do número de serviços, usuários e aplicações que se utilizam da Internet (tais como bibliotecas digitais, educação à distância, comércio eletrônico, áudio e vídeo sob demanda, entre outras). Por outro lado, um grande número de usuários implica em sobrecarga nos servidores responsáveis pelos serviços. No entanto, mesmo uma sobrecarga não deve afetar a disponibilidade e a qualidade dos serviços providos (Figura 1.1).

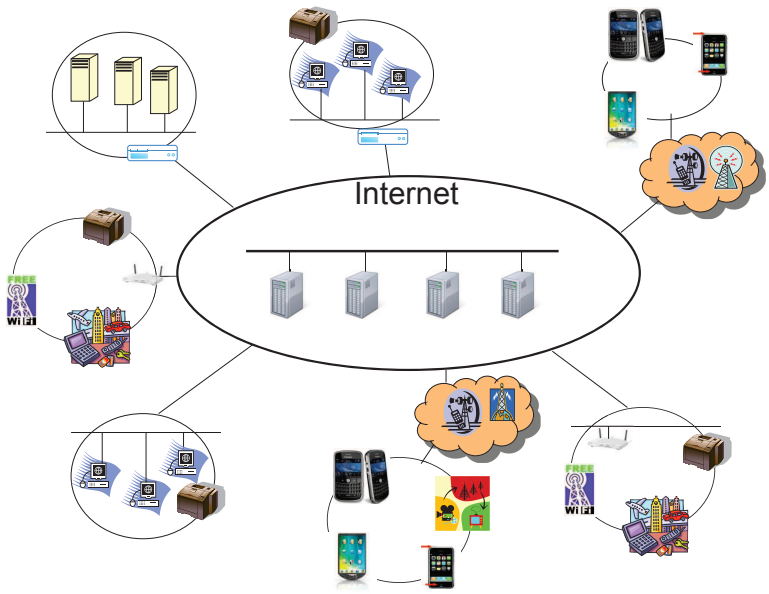

Figura 1.1: Crescimento da Internet.

Um dos maiores desafios para o amplo uso da Internet é a escalabilidade dos servidores, ou seja, a sua capacidade de suportar uma demanda crescente sem que a qualidade dos serviços providos seja afetada. Para conseguir um aumento de desempenho, reduzir o tempo de espera por parte dos usuários e, principalmente, atender às requisições com a qualidade de serviço (QoS) desejada, sem necessariamente aumentar o custo financeiro, pode-se utilizar *clusters* de servidores Web. Estes *clusters* são agrupamentos de servidores interconectados que trabalham juntos para realizar várias tarefas (MORRISON, 2003). É necessário que todos os computadores de um *cluster* trabalhem de forma equilibrada, não permitindo que algum esteja ocioso ou mais ocupado, isto é, seja realizado o balanceamento de cargas (Figura 1.2).

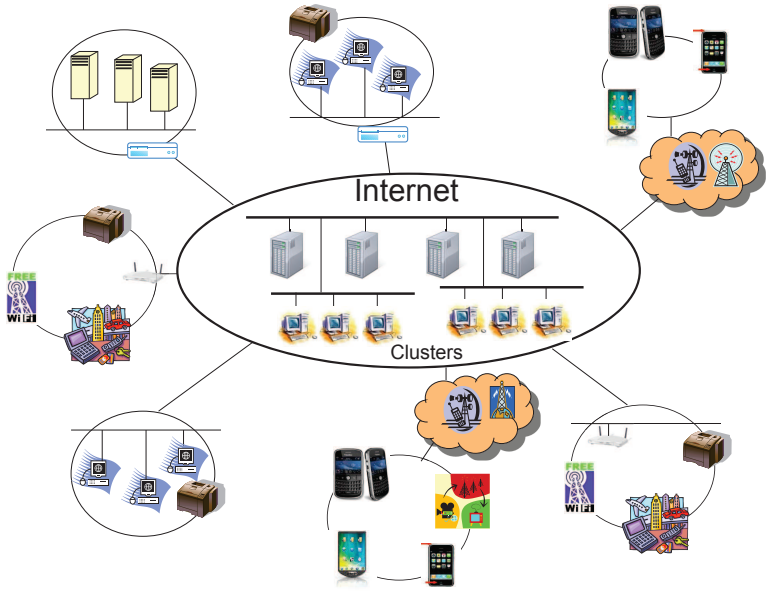

Figura 1.2: Definição de *Clusters* de Servidores Web.

Os *clusters* são usados quando os conteúdos são críticos ou quando os serviços têm que ter disponibilidade e devem ser processados o mais rápido possível, como os serviços Web, por exemplo. Utilizar esta tecnologia tem sido uma saída para substituir os supercomputadores, pois podem oferecer desempenho semelhante e com um custo financeiro mais baixo. Algumas vantagens dos *clusters* são (MORRISON, 2003):

- **Escalabilidade**: é possível aumentar o desempenho do *cluster* adicionando ou trocando os microcomputadores que o compõem;

- **Tolerância a Falhas**: o *cluster* mantém o seu funcionamento mesmo com a paralisação de alguns nós;
- **Baixo Custo Financeiro**: utilizam recursos de fácil acesso e de uso comum;
- **Independência de Fornecedores**: principalmente por utilizarem microcomputadores comuns (inclusive de plataformas heterogêneas), não estão presos a uma tecnologia específica; e
- **Disponibilidade de Serviços**: aumentam a continuidade da utilização dos serviços do *cluster* mesmo se houver falhas em um ou mais computadores que pertençam a este *cluster*.

Nos *clusters* de servidores Web, pode acontecer de alguns destes servidores completarem suas tarefas antes de outros e tornarem-se ociosos porque a carga não está igualmente distribuída ou porque alguns servidores são mais rápidos que outros. Idealmente, é desejável que todos os computadores que formam um *cluster* operem continuamente para a realização das tarefas de modo que a execução das mesmas seja feita no menor tempo possível. Mas, para isso, a distribuição de requisições entre os servidores de um *cluster* impõe novas demandas de processamento para garantir o equilíbrio de trabalho dos computadores que o compõem e uma constante redistribuição dos recursos disponíveis. Os recursos em um *cluster* devem estar sempre disponíveis e sua distribuição deve ser rápida e, principalmente, segura.

1.1 Motivação

Alguns problemas existentes atualmente no uso da Internet frustram usuários e causam preocupação em muitos administradores de *clusters*. Um dos principais problemas enfrentados por estes é adequar os recursos para atender às exigências dos usuários. Até mesmo para administradores experientes, o gerenciamento de um *cluster* é um trabalho exaustivo. Particularmente, alocar os recursos de um *cluster* manualmente pode, facilmente, tornar-se uma tarefa difícil de gerenciar. Isto pode ocorrer porque as necessidades de processamento podem mudar rapidamente em um ambiente dinâmico, como a Internet.

Os administradores de *clusters* possuem uma grande quantidade de tarefas a serem realizadas, tais como monitorar a carga dos servidores, estabelecer a quantidade adequada de recursos no *cluster* para atender as suas tarefas com a QoS

contratada pelos usuários, verificar picos de carga dos servidores, adequar os recursos alocados inicialmente, caso seja necessário, e outras. Estas tarefas muitas vezes são repetitivas, e necessitam de um monitoramento constante, devido às mudanças constantes que ocorrem em um ambiente distribuído, como é o caso de *clusters* de servidores Web. Nos vários trabalhos pesquisados, a maioria dessas tarefas é realizada de forma manual e isso pode levar a uma subutilização ou superutilização dos recursos disponíveis, já que as informações que devem ser monitoradas mudam constantemente.

1.2 Objetivos

Baseado na motivação acima descrita, este trabalho apresenta uma arquitetura de reconfiguração dinâmica de servidores Web para aumentar a disponibilidade de serviços em *clusters* de servidores Web usando sistemas multiagentes. Esta arquitetura visa aliviar a tarefa do administrador de *clusters*, no sentido de reduzir a possibilidade de erros em períodos de picos de carga, devido às características inconstantes e imprevisibilidade da carga de trabalho dos servidores Web. O sistema multiagente da arquitetura DARC realizará o suporte necessário à reconfiguração dos recursos disponíveis na plataforma de servidores Web, já que a abordagem multiagente (WOOLDRIDGE, 2002) é uma boa forma para prover o gerenciamento dinâmico dos recursos em uma solução inerentemente distribuída.

Nesta tese é proposta uma Arquitetura de Reconfiguração Dinâmica de Recursos em Clusters de Servidores Web (*Dynamic Architecture for Reconfiguration of Web servers Clusters* - DARC), que realiza uma reconfiguração dinâmica de recursos em uma plataforma de servidores Web. Para testes e validação da mesma, esta arquitetura foi integrada à Plataforma Distribuída com Balanceamento de Cargas para Servidores Web Baseada na Diferenciação de Serviços (*Web Servers - Differentiated Services Admission Control*) (WS-DSAC) (SERRA et al., 2005). Esta reconfiguração dinâmica tem como objetivo auxiliar a tarefa do administrador da plataforma, minimizando seu *stress* em momentos de sobrecarga. A arquitetura DARC, através do monitoramento automático dos recursos de um *cluster* de servidores Web, inserirá suporte necessário às mudanças dinâmicas de recursos entre *clusters* sem a interferência direta do administrador. Os agentes da arquitetura DARC realizam esse suporte, pois são capazes de se adaptar às necessidades atuais de recursos, utilizando para tanto, as experiências passadas de alterações de carga

para realizar distribuição dos servidores Web em cada cluster.

Os principais objetivos buscados nesta tese são:

- Realizar a tarefa de reconfiguração dinâmica de recursos através das interações de agentes reativos ou cognitivos com a plataforma de servidores Web;
- Melhorar a eficiência na utilização dos recursos disponíveis no *cluster* através das ações que os agentes exercerão sobre o mecanismo de gerência da QoS;
- Tomar decisões dinâmicas a respeito da taxa de utilização dos recursos do sistema baseadas em informações de carga dos *clusters* da plataforma de servidores Web que são colhidas pelos agentes da arquitetura DARC;
- Aliviar o trabalho do administrador do *cluster* e reduzir a possibilidade de erros em períodos de picos de carga da Internet;
- Validar o sistema através de modelagem, simulação e experimentos; e
- Comparar a estratégia de balanceamento de cargas sem reconfiguração dinâmica com a estratégia com reconfiguração dinâmica através de experimentos.

A plataforma WS-DSAC foi projetada para realizar o controle de admissão e o balanceamento de cargas em um ambiente distribuído. A plataforma trabalha baseada na diferenciação de serviços, com o objetivo de fornecer níveis diferenciados de QoS em *clusters* de servidores Web. Requisições que chegam podem pertencer a diferentes classes de serviços. O administrador da plataforma associa a cada classe de serviços uma carga máxima que pode ser atingida por esta classe. Tal mecanismo necessita de um administrador para dar suporte às mudanças constantes que ocorrem em um ambiente distribuído.

1.3 Contribuições da Tese

As principais contribuições desta tese são:

i. Solução multiagente "adaptativa" genérica para a realocação dinâmica de recursos, que pode ser aplicada em diferentes contextos;

ii. Aplicação da solução no contexto da plataforma WS-DSAC;

iii. Modelagem da solução permite a simulação e análise em diferentes cenários de implementação;

iv. Realização de experimentos em ambientes reais e a comparação de resultados com as simulações executadas; e

v. Análise da solução proposta através de simulações e experimentos que mostraram a eficácia da mesma.

1.4 Divisão do trabalho

Para uma maior compreensão, este trabalho encontra-se estruturado da seguinte maneira: no Capítulo 2, é apresentado o estado da arte do balanceamento de cargas e da reconfiguração dinâmica de recursos. No Capítulo 3, apresenta-se a fundamentação teórica necessária ao entendimento deste trabalho: Redes de Petri, Sistemas Multiagentes, Linguagem Java e Java RMI, bem como da plataforma WS-DSAC. No Capítulo 4, é apresentada a Arquitetura de Reconfiguração Dinâmica de Recursos. No Capítulo 5, é apresentada a modelagem e análise da arquitetura em redes de Petri coloridas e o cenário de análise de desempenho da Arquitetura de Reconfiguração Dinâmica de Recursos. No Capítulo 6, é apresentada a implementação da Arquitetura de Reconfiguração Dinâmica de Recursos, bem como os testes realizados em um ambiente real. No Capítulo 7, são apresentadas as conclusões do trabalho realizado.

Capítulo 2

Revisão Bibliográfica nas Áreas de Balanceamento de Cargas e Reconfiguração Dinâmica de *Clusters*

Nos *clusters* de servidores Web, pode acontecer de alguns dos computadores que compõem o *cluster* completarem suas tarefas antes de outros e tornarem-se ociosos porque a carga não está igualmente distribuída ou porque alguns servidores são mais rápidos que outros. Idealmente, é desejável que todos os computadores que formam um *cluster* operem continuamente para a realização das tarefas de modo que a execução das mesmas seja feita no menor tempo possível. Para resolver esse problema utiliza-se o balanceamento de carga. Neste capítulo são apresentados alguns trabalhos relevantes nas áreas de balanceamento de cargas e reconfiguração dinâmica de *clusters*.

Em (SERRA et al., 2005), é apresentada uma plataforma que realiza o balanceamento de carga em Servidores Web baseada na diferenciação de serviços. Os servidores são agrupados em diferentes *clusters* Web de acordo com as classes de serviços estabelecidas. Cada *cluster* Web é responsável por atender uma classe específica de serviços e por garantir a QoS (Qualidade de serviço) medida por meio do "coeficiente de reatividade" (OLEJNIK; BOUCHI; TOURSEL, 2002)[1]. Este coeficiente

[1]Diretamente relacionado ao tempo de resposta ao cliente

mede a carga dos servidores Web, estimando o tempo que uma tarefa espera para utilizar a CPU. A utilização de agentes nessa plataforma é apenas no sentido de medir a carga dos servidores Web que compõem o cluster.

Vários trabalhos utilizam a abordagem de agentes para realizar balanceamento de carga dinâmico. Nos artigos (SHI et al., 2007; CUI; CHAE, 2007) são utilizadas as abordagens middleware e agentes.

Em (SHI et al., 2007) é apresentada uma solução de balanceamento de carga de *middleware* para aplicações orientadas a serviço. Este *middleware* é introduzido na camada de aplicação usando interfaces IDL (*Interface Definition Language*) e tem como funções evitar gargalos na camada de aplicação, realizar o balanceamento da carga de trabalho entre os diferentes serviços e permitir a replicação de componentes de serviços de forma escalável, fornecendo mais acesso aos recursos, melhorando assim o desempenho das aplicações. Esta é uma solução baseada em agente que suporta controle, resposta e alocação rápida de recursos entre os diferentes serviços.

Em (CUI; CHAE, 2007), é proposta uma solução baseada em agentes que realiza balanceamento de carga para a abordagem de *middleware* de RFID (*Radio Frequency Identification*). Esta abordagem realiza o balanceamento de cargas utilizando as políticas de Informação, de Iniciação, de Transferência, de Seleção e de Localização. Dois agentes são desenvolvidos, LGA (*Load Gathering Agent*) e RLBA (RFID *Load Balancing Agent*). LGA é um agente móvel e é responsável por realizar a primeira política, enquanto o agente RLBA é um agente estacionário e compatível com as outras políticas. Esta solução melhora a eficiência, disponibilidade e escalabilidade do sistema para middlewares de RFID.

Nos artigos (NEHRA; PATEL, 2007; NEHRA; PATEL; BHAT, 2006; PONCI; DESHMUKH, 2009) é apresentado o balanceamento de cargas utilizando a abordagem de agentes móveis (AM).

Em (NEHRA; PATEL; BHAT, 2006), é apresentado um sistema multiagente para balanceamento de cargas, que é implementado na plataforma PMADE (*Platform for Mobile Agent Distribution and Execution*). Esta técnica de balanceamento é chamada DLBDMA (*Dynamic Load Balancing Distributed using Mobile Agents*). Foram introduzidos tipos de agentes necessários para satisfazer os requisitos da proposta de equilíbrio da carga. São apresentadas estratégias para minimizar o tempo médio de conclusão das aplicações que rodam em paralelo.

Em (NEHRA; PATEL, 2007), é proposto o algoritmo MALB (*Mobile Agent Based Load Balancing*), que realiza balanceamento de cargas dinâmico utilizando agentes móveis na plataforma PMADE. Neste artigo foram considerados três tipos de carga a serem balanceados: CPU, Memória e E/S. Com este algoritmo, o tempo de resposta dos processos que executam paralelamente e estão competindo pelos mesmos recursos no *cluster* foi reduzido.

Em (PONCI; DESHMUKH, 2009), é apresentada uma arquitetura que integra agentes móveis e estáticos para controlar e avaliar a qualidade da energia em sistemas com controle distribuído. O agente móvel é usado para identificar as cargas que impactam na qualidade da energia, com base nos índices de cada carga. Esta informação é usada pelo agente estático como nível de prioridade dinâmica para decidir quais cargas devem ser reduzidas se a demanda da energia exceder a quantidade de energia necessária.

Os artigos (WANG et al., 2006; HERRERO et al., 2006, 2007) apresentam balanceamento de cargas em grades (*grids*).

Em (WANG et al., 2006), é apresentado um modelo baseado em agentes de escalonamento distribuído para serviços em grades. Os agentes foram utilizados para disponibilizar balanceamento de cargas em nível de serviço, bem como possibilitar tolerância a falhas. Para garantir um escalonamento eficiente, foi definido um grau de confiabilidade para permitir a disponibilidade dos recursos e serviços da grade. Os agentes são responsáveis por verificar o ambiente e realizar o escalonamento através de um ajuste dinâmico da carga dos nós que compõem a grade.

Um dos problemas mais importantes no gerenciamento de ambientes em grades é equilibrar a carga dos nós computacionais. Com o objetivo de resolver esse problema foi desenvolvido o modelo AMBLE (*Awareness Model for Balancing the Load in Collaborative Grid Enviroment*) como um ambiente colaborativo para balancear a carga. Em (HERRERO et al., 2006), sistemas multiagentes são aplicados para criar um ambiente colaborativo para gerenciar os recursos disponíveis em uma grade. A implementação deste modelo é aberta, flexível, utiliza interfaces públicas e padrões, bem como é baseada em especificações de *Web Services*.

Em (HERRERO et al., 2007), o modelo CAMBLE (Cooperative AMBLE) estende o modelo AMBLE ao introduzir cooperação entre os nós que compõem a grade, ou seja, os agentes desse modelo trabalham como equipes e comportam-se de forma a incrementar a utilidade global do sistema e não sua utilidade individual. Este modelo

oferece funcionalidades específicas para gerenciar as tarefas de balanceamento de carga em uma grade cooperativa, tais como o balanceamento de carga realizado por agentes replicados para cada um dos nós do *cluster*.

No artigo (SUGAWARA et al., 2006), investigam-se as estratégias de busca de parceiros para auxiliar os agentes na realização de suas tarefas e como esta busca pode afetar o desempenho total de um SMA (Sistema Multiagente). Esta busca por parceiros pode ser realizada de forma estática, adaptativa ou colaborativa e é baseada em informações armazenadas localmente. Este trabalho está focado na estrutura das redes de agente, e varia de acordo com a carga das tarefas realizadas por estes. São discutidas também as conclusões obtidas a partir dos resultados experimentais: (1) a aprendizagem dos parceiros deve ser derivada dos dados observados; e (2) para essa aprendizagem ser mais precisa, cada agente deve ter em conta os estados da rede e dos outros agentes.

Em todos os trabalhos acima citados, se houver necessidade de reconfiguração de *clusters* (isto é, alocação/desalocação de servidores), isso não é realizado dinamicamente. Dessa forma, a presença de um administrador para gerenciar as constantes modificações que ocorrem em um ambiente distribuído é necessária, embora o trabalho manual seja cansativo e passível de falhas.

Os trabalhos citados a seguir apresentam soluções que realizam reconfiguração dinâmica de *cluster*.

No artigo (NETTO et al., 2005) é apresentada uma estratégia transparente para alocação de recursos que possibilita solicitar recursos ociosos em um *cluster* para executar as aplicações necessárias em uma grade. Quando o *cluster* necessitar dos recursos que estão alocados à grade, estes podem ser recuperados. Porém é necessária que o administrador do *cluster* permita a alocação e dos recursos disponíveis do *cluster*.

Em (ADAM; STADLER, 2005) é apresentada uma solução para alocar servidores em um *cluster* para o processamento de requisições e ativar automaticamente um outro servidor em espera quando a carga do *cluster* aumentar acima de um determinado limite. Inicialmente, uma requisição é alocada para um servidor aleatoriamente. Se este servidor não puder processar a requisição, esta é então enviada aleatoriamente para outro servidor, e assim por diante, até que um servidor seja capaz de processá-la ou o tempo máximo para atendimento de uma requisição seja esgotado.

O trabalho mais semelhante ao que estamos propondo é apresentado por (ZHANG et al., 2006). Nesse trabalho é realizada a reconfiguração dinâmica de *clusters* utilizando agentes. É introduzido aqui o *Fire Phoenix*, um software capaz de gerenciar *clusters*. Este pode ser utilizado tanto em aplicações científicas como também em aplicações comerciais e executa em plataformas heterogêneas. Esta solução utiliza agentes para realizar a reconfiguração dos recursos em um *cluster*. Este software possibilita um gerenciamento fácil e confiável para *clusters*, porém, o gerenciamento é feito em nível do Sistema Operacional (*Fire Phoenix Cluster Operating System*).

No trabalho apresentado em (SUNG et al., 2007), é proposto um mecanismo de auto-configuração de *clusters* utilizando a abordagem agentes. Este mecanismo permite que servidores pertençam a vários *clusters*, dinamicamente, sem a necessidade de intervenção humana. Qualquer servidor pertencente a um *cluster* pode se tornar o servidor de backup para outro servidor, bem como escolher o seu servidor de backup. Para realizar esta auto-configuração, mecanismos de segurança são implementados. Porém, quem gerencia estas tarefas é um servidor central.

Em (ABDELLATIF; KORNAS; STEFANI, 2007), é demonstrado como uma simples reengenharia do JOnAS (*Java Open Source Application Server*) é capaz de realizar uma reconfiguração dinâmica adicionando e removendo réplicas de servidores em um *cluster*, sem ser necessária a parada de todo o sistema. Em aplicações baseadas na Web, assegurar um serviço contínuo por um determinado tempo, independentemente do acordo de nível de eventos externos, é importante e exige que os servidores de hospedagem apoiem a reconfiguração dinâmica.

No artigo (LI et al., 2008), é apresentado um novo algoritmo para a reconfiguração dinâmica da rede de distribuição de energia elétrica baseada em sistema multiagente (SMA). Este sistema multiagente é composto de um agente-coordenador e vários agentes de trabalho (*work agent*). Neste trabalho, um dia é dividido em vários intervalos de tempo e cada intervalo é administrado por um agente de trabalho usando o algoritmo *Particle Swarm Optimization* (PSO). O agente-coordenador gerencia os agentes de trabalho.

Em (MOHEUDDIN; NOORE; CHOUDHRY, 2009), é proposta uma nova solução decentralizada baseada em sistema multiagente para aplicações de sistemas de potência. A técnica de agrupar uma grande rede em *clusters* logicamente relacionados é utilizada e agentes são designados para monitorar esses *clusters*

dedicados em vez de controlar cada dispositivo ou nó. O sistema multiagente proposto usa três tipos de agentes: Agente de Barramento (*Bus Agent*), Agente-Processador (*Processor Agent*) e Agente de Comutação (*Switch Agent*). Simulações foram realizadas e seus resultados mostram que o sistema multiagente proposto é escalável e utiliza poucos agentes que tomam decisões mais rápidas durante as reconfigurações e, quando ocorre uma falha, a sobrecarga na comunicação entre os agentes é bastante reduzida.

Nos trabalhos acima citados a reconfiguração dinâmica é realizada com e sem o uso de adaptabilidade, ou seja, os ambientes se adaptam as necessidades destes. Nesta tese é apresentada a arquitetura DARC (*Dynamic Architecture for Reconfiguration of Web servers Clusters*). O objetivo desta arquitetura é aliviar a tarefa do administrador de *clusters* no sentido de reduzir o número de erros possíveis de serem cometidos por ele em períodos de picos de carga, devido às características inconstantes e imprevisibilidade da carga de trabalho dos servidores Web. Esta arquitetura realiza a reconfiguração dinâmica de clusters de servidores Web, utilizando sistemas multiagentes. Os agentes são capazes de se adaptar às necessidades atuais de recursos utilizando experiências passadas para modificar, se necessário, a distribuição dos servidores Web em cada cluster (e.x., alocando um servidor para um cluster que está sobrecarregando). As vantagens de se utilizar um sistema multiagente é que este é uma boa forma de prover o gerenciamento dinâmico dos recursos já que esta é uma solução inerentemente distribuída. Uma comparação entre os vários trabalhos citados na revisão bibliográfica apresentada nesse capítulo é feita na Tabela 2.1. Não foi possível realizar comparações de desempenho com outras plataformas por não termos acesso a nenhum estudo de outras abordagens com esse enfoque.

Tabela 2.1: Relação entre trabalhos apresentados na revisão bibliográfica.

Autores	Balanceamento de cargas utilizando agentes	Reconfiguração dinâmica de clusters	Presença constante do administrador de clusters
(SERRA et al., 2005)	Não	Não	Sim

continua para próxima página

continuação da página anterior

Autores	Balanceamento de cargas utilizando agentes	Reconfiguração dinâmica de clusters	Presença constante do administrador de clusters
(SHI et al., 2007)	Sim, abordagem middleware	Não	Sim
(CUI; CHAE, 2007)	Sim, abordagem middleware	Não	Sim
(NEHRA; PATEL; BHAT, 2006)	Sim, agentes móveis	Não	Sim
(NEHRA; PATEL, 2007)	Sim, agentes móveis	Não	Sim
(PONCI; DESHMUKH, 2009)	Sim, agentes móveis	Não	Sim
(WANG et al., 2006)	Sim, ambiente Grid	Não	Sim
(HERRERO et al., 2006)	Sim, ambiente Grid	Não	Sim
(HERRERO et al., 2007)	Sim, ambiente Grid	Não	Sim
(SUGAWARA et al., 2006)	Sim, ambiente Grid	Não	Sim
(NETTO et al., 2005)	Sim	Sim, aplicação aloca recursos ociosos em clusters para atender requisições em grids	Sim

continua para próxima página

continuação da página anterior

Autores	Balanceamento de cargas utilizando agentes	Reconfiguração dinâmica de clusters	Presença constante do administrador de clusters
(ADAM; STADLER, 2005)	Sim	Sim, aloca servidores ociosos para processar requisições em clusters	Sim
(ZHANG et al., 2006)	Sim	Sim, ao nível do Sistema Operacional	Sim
(SUNG et al., 2007)	Sim	Sim, aplicação específica para backups de servidores	Sim
(ABDELLATIF; KORNAS; STEFANI, 2007)	Sim	Sim, aplicação gerencia réplicas de códigos em servidores de clusters	Sim
(LI et al., 2008)	Sim	Sim, propõem um algoritmo para a reconfiguração dinâmica da rede de distribuição de energia	Sim
(MOHEUDDIN; NOORE; CHOUDHRY, 2009)	Sim	Sim, aplicação específica aplicações de sistemas de potência	Sim

continua para próxima página

continuação da página anterior			
Autores	Balanceamento de cargas utilizando agentes	Reconfiguração dinâmica de clusters	Presença constante do administrador de clusters
MARQUES; BARROSO; SERRA, 2010	Não, os agentes realizam a reconfiguração dinâmica	Sim, proposta a arquitetura de reconfiguração dinâmica em clusters de servidores web	Não

2.1 Conclusões do Capítulo 2

Nesse capítulo foram apresentados alguns trabalhos relevantes nas áreas de balanceamento de cargas e reconfiguração dinâmica de *clusters* importantes para o desenvolvimento dessa tese.

No próximo capítulo serão apresentadas a plataforma de servidores Web à qual a arquitetura DARC foi integrada e a fundamentação teórica necessária ao entendimento dessa tese.

Capítulo 3

Fundamentação Teórica

Neste capítulo, são descritos os principais fundamentos utilizados para o desenvolvimento da arquitetura DARC: Sistemas Multiagentes e Redes de Petri. Além disso, é apresentada a plataforma de servidores web à qual a arquitetura DARC foi integrada.

Na Seção 3.1, é apresentada a plataforma WS-DSAC. Na Seção 3.2, são apresentados definições e conceitos de agentes, bem como estratégias utilizadas no trabalho que definem o comportamento dos agentes. Na Seção 3.3, são apresentadas a Especificação do Padrão FIPA. Na Seção 3.4, são apresentadas as Redes de Petri e suas extensões. Na Seção 3.5, é apresentada a extensão UML-SPT. Na Seção 3.6, é apresentada uma descrição da linguagem Java e Java RMI.

3.1 Plataforma WS-DSAC

Para a validação da proposta de realocação dinâmica de recursos utilizamos a plataforma de servidores Web WS-DSAC. A utilização dessa plataforma viabilizou a construção e avaliação de um protótipo do sistema multiagente da arquitetura DARC. Nesta seção, será detalhado o seu funcionamento.

3.1.1 Visão Geral da Plataforma

A Plataforma WS-DSAC (SERRA et al., 2005) foi projetada para realizar o controle de admissão e o balanceamento de carga de servidores Web. Esta plataforma oferece diferentes níveis de qualidade de serviço (QoS), baseando-se na diferenciação de serviços, e é composta por um conjunto de elementos básicos: *Class Switch*,

Cluster Gateways e *Servidores Web* (Figura 3.1). Estes elementos são descritos a seguir:

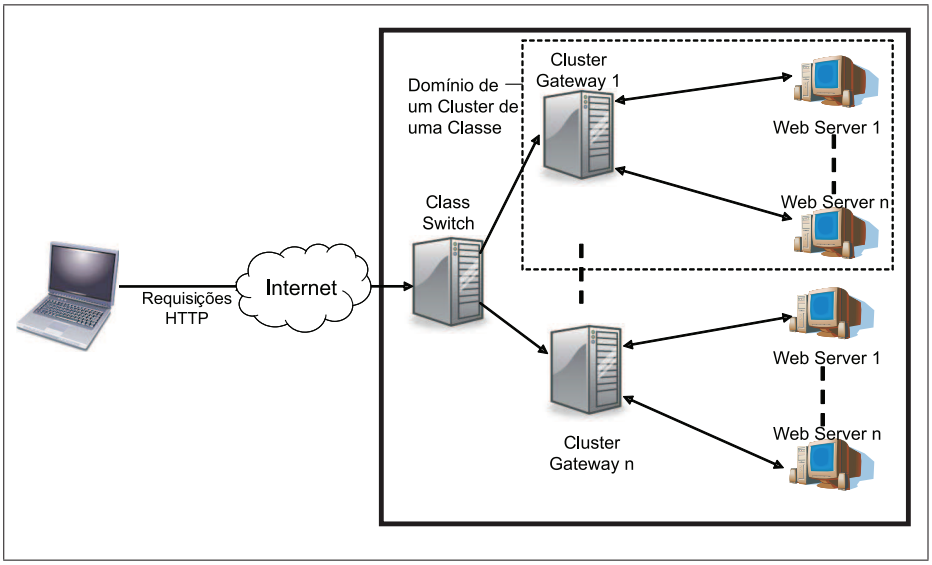

Figura 3.1: Visao geral da plataforma WS-DSAC.

- O *Class Switch* é responsável pela classificação e pelo controle de admissão de novas requisições. Ele recebe requisições HTTP, identifica qual a classe de serviços a que a requisição pertence e, utilizando o algoritmo de controle de admissão e de balanceamento de cargas, envia cada requisição para o *Cluster Gateway* menos carregado dentre aqueles vinculados à classe de serviço da requisição;

- O *Cluster Gateway* escolhe o servidor menos carregado para processar a requisição enviada pelo *Class Switch*. Cada cluster está associado a um domínio de classe e atende a requisições nativas dessa classe prioritariamente; e

- Os *Servidores Web* processam as requisições HTTP enviadas pelos *Clusters Gateways*.

3.1.2 Componentes da Plataforma

Os componentes da plataforma são classificados como: Mecanismos de Monitoramento; Mecanismos de Gestão da QoS; Algoritmo de Controle de Admissão e Balanceamento de carga e Mecanismos de Aplicações.

Mecanismos de Monitoramento

Os mecanismos de monitoramento são compostos por três componentes básicos (Figura 3.2):

- *Monitor Agent* (MA): este elemento monitora e estima a carga de cada servidor Web;
- *Cluster Resource Manager* (CRM): este elemento monitora e estima a carga do cluster de uma classe; e
- *Global Resource Manager* (GRM): este elemento solicita e mantém a informação de carga de cada CRM, ou seja, de cada cluster.

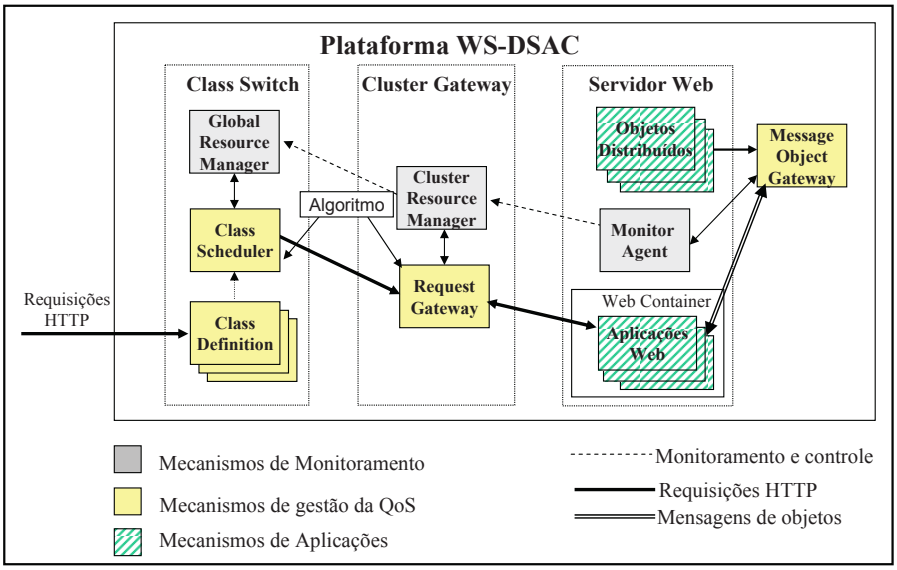

Figura 3.2: Comunicação entre os componentes da plataforma.

O MA possui três funções básicas:

i. A primeira é de registrar o servidor Web em um domínio específico, ou seja, em um cluster de uma determinada classe de serviços. Ele envia uma mensagem solicitando o registro para o CRM responsável pelo domínio da classe pedindo para adicioná-lo ao domínio;

ii. A segunda função do MA é de monitorar a carga do servidor Web. O MA usa uma métrica de monitoramento baseada em um "Coeficiente de Reatividade"

(*Reactivity Coefficient* - RC) (OLEJNIK; BOUCHI; TOURSEL, 2002) (Figura 3.3). Esta métrica dá uma ideia da tendência de carga do servidor, estimando o tempo de espera de uma tarefa pela CPU. O MA possui uma *thread* que passa o controle da CPU para um outro processo e espera ser novamente "acordado" pelo sistema. O tempo de espera da *thread* depende diretamente da carga do servidor. Para eliminar a tomada de decisão baseada em variações momentâneas da carga, esta operação é repetida diversas vezes. Finalmente, é calculada a média do tempo de espera pela CPU (coeficiente de reatividade) que está diretamente relacionada com a carga do servidor; e

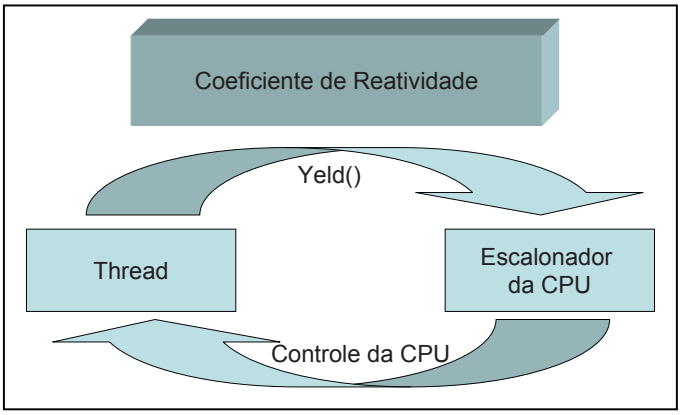

Figura 3.3: Cálculo do Coeficiente de Reatividade.

iii. A terceira funcionalidade básica do MA é de responder ao CRM quando este solicita periodicamente informações sobre a carga de cada servidor Web.

Cada domínio de classe (*Class Cluster Domain*) executa um CRM (*Cluster Resource Manager*), que solicita informações de carga de cada servidor Web em intervalos constantes de tempo. Ele analisa todos os coeficientes de reatividade dos servidores do cluster e faz uma estimativa para a carga do cluster inteiro para o próximo período.

Mecanismos de Gerência da QoS

Os mecanismos de controle de admissão e balanceamento de carga são compostos por quatro elementos (Figura 3.2):

- *Class Scheduler* (CS): este elemento é responsável pela classificação e pelo controle de admissão de requisições HTTP que chegam;

- *Class Definition*: é responsável pela definição das características de cada classe;
- *HTTP Request Gateway* (HRG): este elemento envia as requisições HTTP redirecionadas pelo CS para o servidor menos carregado do cluster; e
- *Message Object Gateway* (MOG): este é responsável pela interceptação e redirecionamento das mensagens trocadas entre os objetos distribuídos dentro de um cluster mantendo a carga balanceada entre os diversos servidores do domínio.

O CS recebe requisições que são enviadas pelos clientes. Depois de classificar a requisição identificando a classe de serviços à qual ela pertence, ele verifica as informações de carga mantidas pelo GRM e analisa se o nível de QoS estabelecido para a classe da requisição pode ser fornecida. Se é possível garantir o nível de QoS especificado, o CS redireciona a requisição para o "Cluster Gateway" eleito pelo mecanismo; senão, ele retorna uma mensagem para o cliente informando que o nível de QoS requerido para aquela classe de serviços não pode ser garantida.

O HRG recebe as requisições redirecionadas pelo CS. Depois, ele decide qual servidor Web irá processar cada requisição, baseando-se na informação de carga mantida pelo CRM.

A plataforma permite a coexistência de serviços Web e de objetos distribuídos que podem interagir conjuntamente para prover serviços aos clientes. Quando um objeto distribuído é instalado na plataforma, o serviço de administração utiliza a interface de comunicação distribuída do objeto para automaticamente criar seu "Message Gateway". O MOG intercepta as invocações de métodos remotos dentro de um cluster específico e, baseado na carga informada pelo Monitor Agent, ele redireciona as invocações para o servidor menos carregado. Desta maneira, se um serviço Web utiliza objetos distribuídos para resolver suas sub-tarefas, a carga imposta por estas sub-tarefas também será balanceada entre os servidores do cluster.

O administrador pode definir diferentes classes de serviços, bem como as características de cada classe, utilizando o serviço de definição de classes. As características de cada classe estão diretamente relacionadas com o coeficiente de reatividade que é requerido para cada requisição de cliente pertencente àquela classe.

Mecanismos de Aplicações

Um conjunto de serviços foi desenvolvido para permitir a administração da plataforma. Eles são agrupados em: Serviços de Instalação e Serviços de Monitoramento dos Clusters.

Usando os serviços de instalação, o administrador da plataforma pode instalar serviços Web e objetos distribuídos na plataforma. Os serviços são instalados em um certo número de servidores Web. Estes serviços podem ser compostos por servidores Web e objetos distribuídos. Os serviços são replicados na plataforma e os mecanismos de balanceamento de carga são automaticamente criados.

Os serviços de monitoramento de carga exibem informações estatísticas (taxa de requisições recusadas, média do coeficiente de reatividade por classe de serviços, etc.) sobre todo o sistema. Usando estas informações, o administrador do sistema pode redefinir a estratégia de distribuição dos recursos entre os diversos clusters.

3.1.3 Funcionamento Geral da Plataforma

Um dos objetivos da plataforma WS-DSAC é utilizar de forma eficaz os recursos de processamento disponíveis na infraestrutura, aumentando a QoS oferecida às requisições atendidas, independentemente da classe de serviços à qual elas pertençam. Isso significa que, enquanto existirem recursos de processamento disponíveis e os parâmetros de QoS estabelecidos para cada classe de serviços forem respeitados, todos os recursos, mesmo os alocados prioritariamente para uma determinada classe, devem ser compartilhados de forma igualitária. A estratégia adotada para a realocação dinâmica de recursos entre as classes de serviços se baseia em uma mudança no modo de funcionamento de cada "domínio de um cluster".

A plataforma oferece diferentes níveis de QoS, um nível diferente para cada classe de serviço. Requisições que chegam podem pertencer a diferentes classes de serviço. O administrador da plataforma associa a cada classe de serviço uma carga máxima que pode ser atingida pelo seu "class cluster". Um parâmetro de QoS denominado "Coeficiente de Reatividade" é associado a cada classe de serviços pré-estabelecida e este parâmetro é usado em cada servidor Web.

Em um determinado intervalo de tempo, recursos reservados prioritariamente para uma classe podem estar em um de três modos possíveis: *modo compartilhado*; *modo exclusivo* e *modo saturado* (veja Figura 3.4).

- Quando está no *modo compartilhado*, o cluster da respectiva classe possui recursos disponíveis que podem ser utilizados por outras classes de serviços. Estando neste modo de trabalho, o "class cluster" atende requisições de diferentes tipos de classes, sem comprometer o contrato estabelecido com a classe nativa do cluster.

- Quando o cluster passa ao *modo exclusivo*, ele só aceita requisições da classe nativa. Isso significa que os níveis de carga chegaram a um patamar em que aceitar requisições de outras classes pode implicar na rejeição de sessões da classe nativa.

- Quando o cluster passa ao *modo saturado* ele não aceita nenhuma nova requisição. Isso significa que os recursos existentes não serão suficientes para garantir a QoS assegurada às requisições em processamento, caso o mesmo aceite novas requisições.

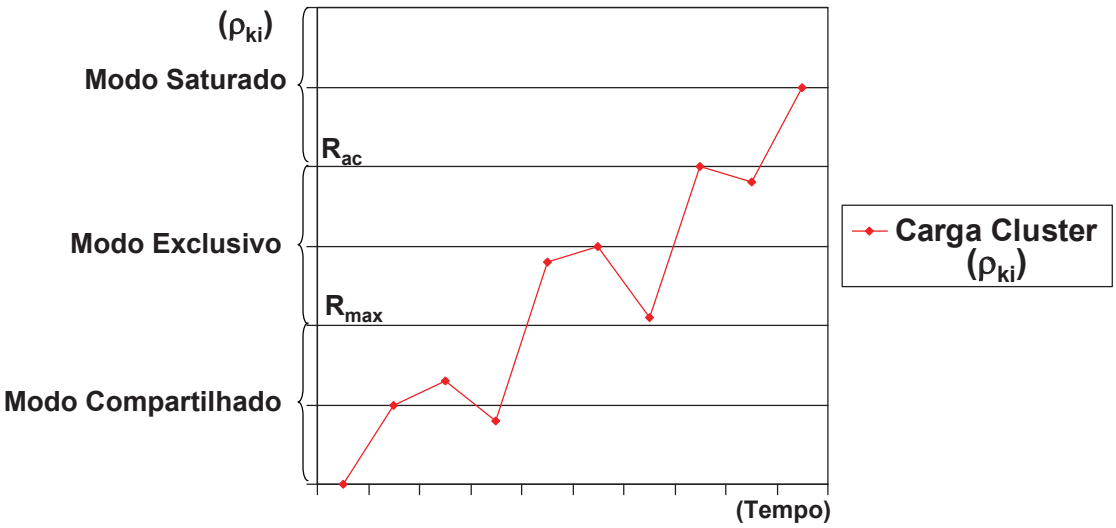

Figura 3.4: Modos de Trabalho do WS-DSAC.

A mudança de modo de funcionamento é baseada em dois parâmetros dinâmicos do cluster (recalculados a cada intervalo de tempo), R_{emk} e ρ_{ki}, e dois estáticos, que são R_{max} e R_{ac}, sendo que:

- ρ_{ki} representa a carga estimada do cluster para o período de tempo seguinte e determina uma margem de segurança para que os contratos de QoS estabelecidos com a classe nativa sejam respeitados;

- R_{emk} é o valor do coeficiente de reatividade alcançado pelo cluster de uma classe. O valor máximo que pode ser alcançado pela variável R_{emk} é o valor atribuido a R_{max};

- R_{max} é o valor-limite que o coeficiente de reatividade do cluster da classe pode assumir para que o mesmo trabalhe no modo compartilhado. Quando o coeficiente de reatividade está acima deste valor e abaixo de R_{ac}, o cluster trabalha em "modo exclusivo"; e

- R_{ac} é o valor-limite que o coeficiente de reatividade do cluster da classe pode assumir para trabalhar no modo exclusivo. Quando o coeficiente de reatividade está acima deste valor o cluster trabalha em "modo saturado".

3.1.4 O Algoritmo de Controle de Admissão e de Balanceamento de Carga

O algoritmo de controle de admissão e de balanceamento de carga integra estratégias dos mecanismos discutidos em (CHERKASOVA; PHAAL, 2002) e (ABDELZAHER; SHIN; BHATTI, 2001). Ele foi projetado de forma a respeitar as características de escalabilidade e da plataforma. Formalmente, este algoritmo pode ser definido mediante os seguintes parâmetros:

- T_1, T_2, ..., T_n é uma sequência de intervalos de tempo utilizados para tomar decisões em relação à admissão ou rejeição de requisições durante o próximo intervalo de tempo;

- N_c denota o número de classes de serviços diferentes (*Class Clusters*) definidas no contexto da plataforma, sendo que $(N_c > 1)$;

- N_{si} denota o número de servidores alocados no domínio da classe i, sendo que $(N_{si} > 0)$;

- r_{kij} é a carga (média do coeficiente de reatividade) do servidor Web j dentro do cluster da classe i, medida durante o *k-ésimo* período de tempo. Os valores desses índices variam em $(1 < i \leq N_c)$ e $(1 \leq j \leq N_{si})$; e

- r_{ki} é a carga média dos servidores registrados no domínio da classe i medida durante o *k-ésimo* período de tempo.

Quando uma requisição chega à plataforma o *class switch* verifica a qual classe ela pertence. Dado que a requisição pertence à classe i e que o cluster m é o menos carregado, o *class switch* atua segundo o algoritmo de controle de admissão e de balanceamento de carga. Os aspectos básicos deste algoritmo são resumidos na Figura 3.5.

Se ($\rho_{ki} \leq R_{max}$) então
  o cluster trabalha no **modo compartilhado**;
senão Se ($\rho_{ki} > R_{max}$ e $\rho_{ki} \leq R_{ac}$) então
  o cluster modifica o seu modo de trabalho para **exclusivo**;
  senão Se ($\rho_{ki} > R_{ac}$)
  o cluster modifica o seu modo de trabalho para **saturado**;

Figura 3.5: Algoritmo de controle de admissão e de balanceamento de cargas.

3.2 Abordagem de Agentes

Nesta seção, é discutida a abordagem de agentes. Nesta perspectiva, inicialmente são analisadas as áreas científicas que inspiraram os agentes. Logo após, são discutidas as diferenças entre Inteligência Artificial Distribuída (IAD) e Sistemas Multiagentes (SMA). São também discutidos a definição de agentes, bem como os principais atributos que um agente deve possuir. As linguagens de agentes e a forma de comunicação entre eles são descritas.

3.2.1 Áreas Científicas que Inspiraram os Agentes

Segundo (REIS, 2003), a área de investigação designada por agentes surgiu inspirada nas áreas científicas da Inteligência Artificial, Engenharia de Software, Sistemas Distribuídos e Redes de Computadores, Sociologia, Teoria dos Jogos e Economia (Figura 3.6).

Embora a Inteligência Artificial (IA) tenha exercido inicialmente uma influência muito forte sobre o campo dos Agentes/Sistemas Multiagentes, verifica-se hoje em dia que este campo já evoluiu muito, sendo considerado apenas uma sub-área da IA.

Paralelamente a este desenvolvimento, a investigação efetuada em IA, inspirada em conceitos clássicos de organizações e sociedades, começava a romper com as

Figura 3.6: Campos que inspiraram os Agentes e Sistemas Multiagentes.

soluções típicas até então adotadas na construção de sistemas inteligentes. A diferença entre essas duas linhas de investigação é apresentada a seguir (REIS, 2003):

- **Resolução Distribuída de Problemas (RDP)**: O trabalho de resolver um problema particular pode ser dividido entre um número de módulos que cooperam e compartilham conhecimento sobre o problema e sobre o desenvolvimento da solução. Sendo assim, o planejamento das ações a desenvolver é o resultado da decomposição do problema em vários sub-problemas que são distribuídos aos diversos agentes envolvidos. Como resultado desta partilha, os agentes cooperam apenas na divisão do esforço e na partilha de conhecimentos e resultados; e

- **Sistemas Multiagentes (SMA)**: Esta segunda linha faz uso de conceito de comunidade de agentes, cujo enfoque se baseia na existência de uma sociedade, composta por vários agentes que atuam no sistema por meio de cooperação e concorrência. Estes sistemas são compostos por dois ou mais agentes, que exibem um comportamento autônomo mas ao mesmo tempo interagem com os outros agentes presentes no sistema.

  Um Sistema Multiagente é um sistema computacional em que dois ou mais agentes interagem ou trabalham em conjunto de forma a desempenhar determinadas tarefas ou satisfazer um conjunto de objetivos. A investigação

científica e a implementação prática de Sistemas Multiagentes está focalizada na construção de padrões, princípios e modelos que permitam a criação de pequenas e grandes sociedades de agentes semi-autônomos, capazes de interagir convenientemente de forma a atingirem os seus objetivos (REIS, 2003). Estes agentes exibem duas características fundamentais:

i. Capacidade de agir de forma autônoma tomando decisões que levam à satisfação dos seus objetivos;

ii. Capacidade de interagir com outros agentes utilizando protocolos de interação social, inspirados nos humanos e incluindo pelo menos algumas das seguintes funcionalidades: coordenação, cooperação, competição e negociação.

Um dos pontos essenciais para permitir a construção de sociedades de agentes consiste em conseguir gerir as interações e as dependências das atividades dos diferentes agentes no contexto do Sistema Multiagente, isto é, coordenar esses agentes. Desta forma, a coordenação desempenha um papel essencial nos SMA porque estes sistemas são inerentemente distribuídos.

3.2.2 Definição de Agentes

Um sistema baseado em agentes é formado por entidades computacionais autônomas, que possuem capacidades e objetivos individuais, e podem ser agrupados para atuar cooperativamente visando atingir os objetivos de um sistema.

Não existe uma definição universalmente aceita para o termo agente de software. Wooldridge (WOOLDRIDGE, 2002) explica que parte da dificuldade em definir agente surge do fato de que, para diferentes domínios de aplicação, as propriedades associadas ao conceito de agente assumem diferentes níveis de importância.

Um conceito bastante aceito é que um agente é um programa de computador que é capaz de se comunicar, cooperar e aprender (WOOLDRIDGE, 2002).

Os agentes operam e existem em algum ambiente (Figura 3.7), o qual tipicamente pode ser de hardware ou software. O ambiente pode ser aberto ou fechado e pode, ou não, conter outros agentes. Embora existam situações em que um agente possa trabalhar sozinho, o aumento das redes de computadores transforma esta situação em algo raro e, no estado atual da tecnologia, os agentes interagem com outros agentes.

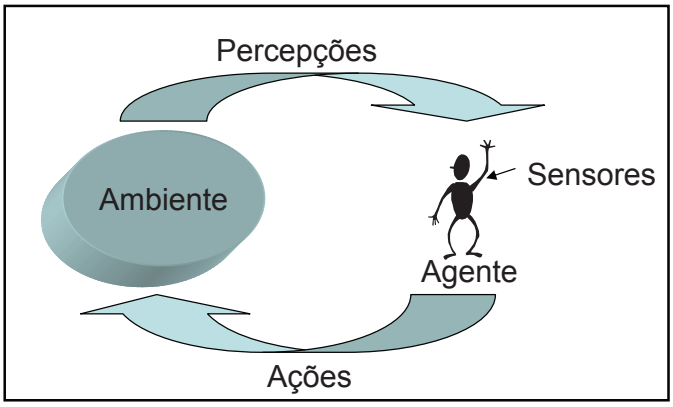

Figura 3.7: Agentes interagindo com um ambiente.

3.2.3 Propriedades Essenciais dos Agentes

Para Wooldridge (WOOLDRIDGE, 2002), um agente é um sistema computacional encapsulado que está situado em algum ambiente e é capaz de ação autônoma flexível neste ambiente, a fim de alcançar os seus objetivos de projeto. Com base nesta visão, o autor define um agente como tendo as seguintes propriedades essenciais:

- **Autonomia**: executa a maior parte de suas ações sem interferência direta de agentes humanos ou de outros agentes computacionais, possuindo controle total sobre suas ações e estado interno;
- **Habilidade Social**: por impossibilidade de resolução de certos problemas ou por outro tipo de conveniência, interage com outros agentes (humanos ou computacionais), para completar a resolução de seus problemas, ou ainda para auxiliar outros agentes. Disso, surge a necessidade de que os agentes tenham capacidade para comunicar seus requisitos a outros agentes e de um mecanismo decisório interno que defina quando e quais interações são apropriadas;
- **Capacidade de Reação**: percebe e reage a alterações no ambiente em que estiver inserido; e
- **Capacidade Pró-ativa**: um agente do tipo deliberativo além de atuar em resposta às alterações ocorridas em seu ambiente, apresenta um comportamento orientado para objetivos, tomando iniciativas quando julgar apropriado.

3.2.4 Propriedades Desejáveis dos Agentes

Para além das propriedades essenciais mencionadas, Wooldridge e Jennings definem ainda uma noção forte de agente, que deriva essencialmente da área da Inteligência Artificial. Consoante essa noção, um agente é visto como uma entidade cognitiva e com consciência, que é capaz de exibir sentimentos, percepções e emoções, à semelhança dos humanos. Assim, segundo essa noção, as seguintes propriedades podem também ser desejáveis para os agentes: Mobilidade, Verdade, Conhecimento e Crença, Racionalidade, Intenções e Obrigações (WOOLDRIDGE; JENNINGS, 1995).

Usualmente, além das propriedades apresentadas, são também referidas como propriedades comuns dos agentes:

- **Inteligência**: O estado de um agente é formalizado pelo seu conhecimento (isto é, suas crenças, seus objetivos e seus planos) e ele interage com outros agentes utilizando uma linguagem simbólica. Possui capacidade de raciocínio abstrato e de resolução de novos problemas e adaptação a novas situações;
- **Continuidade Temporal**: O agente é um processo que é executado continuamente ao longo do tempo. Usualmente, são utilizados também os termos persistente ou "com uma vida longa" para designar a continuidade temporal dos agentes;
- **Caráter**: O agente possui uma personalidade e eventualmente possui também um "estado emocional"; e
- **Capacidade de Adaptação**: faz com que o agente adquira novos conhecimentos e altere o seu comportamento baseado na sua experiência prévia.

Os agentes da arquitetura DARC possuem as características essenciais apresentadas, bem como as características desejáveis de continuidade temporal e capacidade de adaptação.

3.2.5 Linguagens de Agentes

As linguagens de agentes possibilitam a partilha de significados e sentido na informação trocada entre os agentes. Estas são as linguagens de programação que podem incorporar vários princípios propostos por teóricos que estudam os agentes.

Tais linguagens trabalham questões relativas a como programar a troca de mensagens entre os agentes, bem como quais as primitivas corretas devem ser empregadas para realizar esta tarefa.

3.2.6 Comunicação entre Agentes

A comunicação entre agentes é o processo pelo qual a informação é trocada numa interação entre os mesmos. Mensagens transmitidas possuem conteúdo e contexto. Conteúdo refere-se aos dados codificados na mensagem; contexto é a forma como as propriedades mudam desde os dados à informação (BICA, 2000).

3.3 Visão Geral das Especificações da FIPA

As primeiras tentativas de definir arquiteturas para SMA aconteceram ainda nos anos 80. Mas foi nos anos 90, com a explosão da Internet, que o interesse em SMA cresceu. Assim, em 1995, surgiu a FIPA (*Foundation for Intelligent Physical Agents*)[1], uma sociedade da IEEE (*Institute of Electric and Electronic Engineers*).

Esta fundação surgiu como uma associação, sem fins lucrativos, de empresas e organizações, com o intuito de se trabalhar na criação de padrões para a interoperabilidade de agentes de software heterogêneos (FIPA, 1999).

O primeiro conjunto de especificações da FIPA só seria lançado em 1997 e só no fim de 2002 essas especificações seriam finalizadas e definidas como padrão. As especificações da FIPA foram um grande incentivo para o surgimento de novos esforços para a criação de arquiteturas genéricas, que garantissem a interoperabilidade de SMA.

O trabalho de padronização da FIPA destina-se a facilitar a interoperabilidade de sistemas de agentes, uma vez que, além da linguagem de comunicação, a FIPA especifica também quais os principais agentes envolvidos na gerência de um sistema.

A FIPA definiu um conjunto de especificações para guiar a implementação de plataformas multiagentes (ambientes de execução onde operam os agentes). Tais especificações definem apenas um modelo de referência e, portanto, limitam-se a descrever o comportamento externo dos componentes do sistema, deixando em aberto detalhes relativos à estrutura interna e implementação. Essas especificações definem um conjunto de serviços que precisam ser providos pelas plataformas.

[1] [http://www.fipa.org/]

Na Figura 3.8, é apresentada uma visão funcional das especificações da FIPA, mostrando serviços obrigatórios e opcionais em uma plataforma.

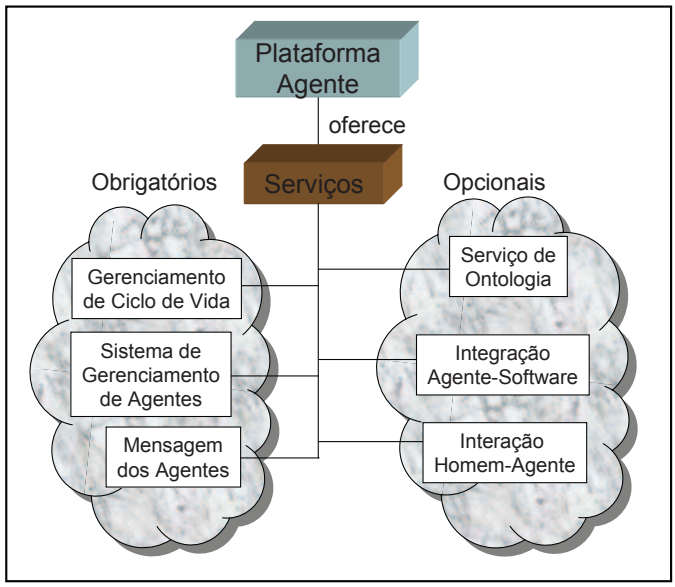

Figura 3.8: Arquitetura Abstrata FIPA.

No desenvolvimento de uma arquitetura de agentes, é necessária a descrição dos agentes, de seus serviços, do seu ciclo de vida e do transporte de mensagens entre eles. Dessa forma, a identificação dos agentes e suas relações fica mais fácil. Esses itens serão descritos nas próximas subseções.

3.3.1 Serviço de Gerenciamento do Ciclo de Vida

O serviço de gerenciamento do ciclo de vida do agente é responsável pela criação, remoção e migração de agentes entre plataformas. A FIPA define os possíveis estados no ciclo de vida de um agente e as transições que ocasionam as mudanças de estados. Quando um agente é criado, a plataforma deve associar ao agente um identificador, o Identificador do Agente (IdA). Este identificador deve ser imutável e único para permitir que um agente ou serviço possa contactar outro agente de maneira não-ambígua.

O serviço de transporte de mensagens é o responsável por entregar as mensagens (assíncronas) trocadas entre os agentes, estejam eles na mesma plataforma ou em plataformas distintas. O padrão FIPA prevê a possibilidade de implementação de mecanismos de segurança no transporte de mensagens. Os agentes se comunicam usando uma Linguagem de Comunicação de Agentes - ACL (KQML/KIF).

A FIPA também descreve a mudança de estado dos agentes, como apresentado a seguir:

- **Estado Inicial**: é nesse estado que os agentes são criados, mas ainda não foram registrados na arquitetura e portanto ainda não podem se comunicar entre si;
- **Estado Ativo**: é nesse estado que os agentes são registrados na arquitetura e podem, a partir de então, comunicar-se entre si; e
- **Estado de Espera**: é nesse estado que os agentes estão esperando a chegada de mensagens para poderem realizar uma ação.

A mudança entre os estados possíveis de um agente é feita através da chamada de métodos, como pode ser visto na Figura 3.9 (criação, espera e acordar).

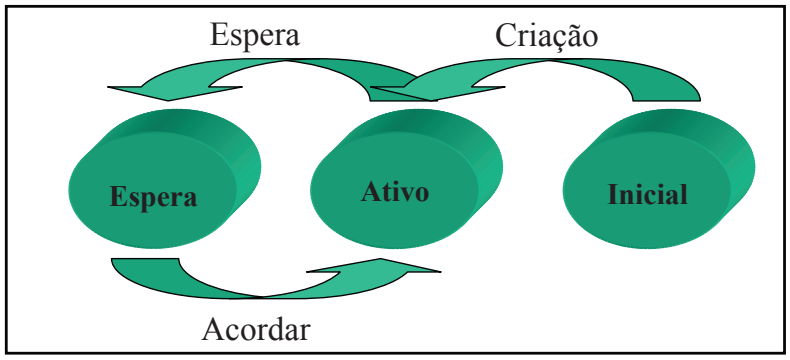

Figura 3.9: Ciclo de Vida de um Agente.

3.3.2 Gerenciamento do Agente

Além dos serviços que são especificados pela FIPA, ela ainda sugere uma arquitetura de referência para plataformas de agentes. Esta arquitetura (Figura 3.10) fornece uma estrutura que descreve como os agentes executam. O modelo de referência (descrito pela FIPA) é elaborado a partir desta estrutura e fornece base para a criação, registro, localização, comunicação e remoção de um agente. O modelo de referência de gerenciamento do agente consiste dos componentes lógicos, descritos a seguir:

i. **Plataforma de Agente (PA)**: Este componente fornece a infra-estrutura física na qual os agentes são implementados. A plataforma de agentes é

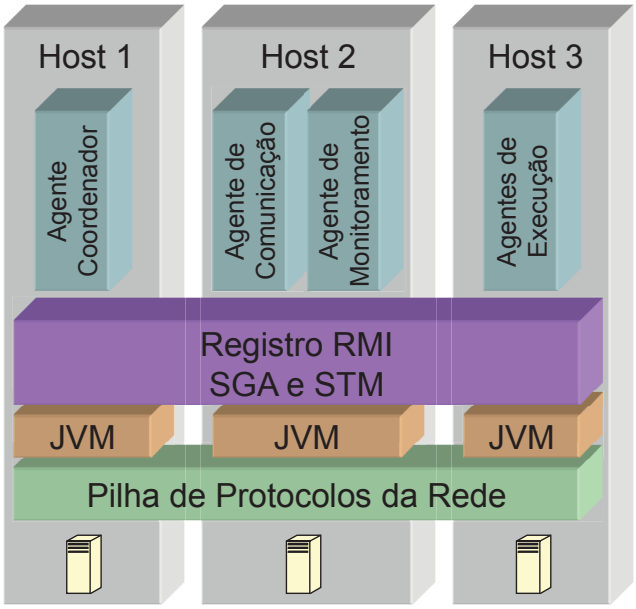

Figura 3.10: Arquitetura de Referência FIPA para Plataformas de Agentes.

constituída dos componentes físicos e lógicos, além dos componentes do modelo de referência FIPA definidos a seguir (STM e SGA) e dos seus agentes.

ii. **Sistema de Gerenciamento do Agente (SGA)**: Este sistema fornece controle sobre o uso e o acesso aos agentes da arquitetura. Há um único SGA por PA. O SGA exerce controle de supervisão sobre o acesso e a utilização da PA. O SGA também mantém um diretório de Identificadores de Agentes (IdA) que contém endereços de transporte (entre outros) para os agentes inscritos na PA. Cada agente deve se registrar no SGA de maneira que possua um IdA válido. O registro é feito pelo RMI. Este registro pertence a um servidor de nomes que a linguagem Java utiliza para armazenar e consultar os objetos pelos nomes. Em outras palavras, a arquitetura usa estes registros para manter uma referência a outros agentes da arquitetura.

iii. **Serviço de Transporte de Mensagem (STM)**: Este é o método de comunicação padrão para agentes em diferentes PAs. A comunicação entre os agentes é garantida pelo RMI e as interfaces de programação da aplicação (APIs) são implementadas na arquitetura.

3.3.3 Mensagem dos Agentes

Os agentes se comunicam através de mensagens que são codificadas na linguagem de comunicação entre eles. Um serviço é definido em termos de um conjunto de ações que o agente suporta. Cada ação define uma interação entre o serviço e o agente que se utiliza dele. A semântica dessas ações é descrita informalmente, para minimizar a complexidade. A FIPA é neutra sobre como esses serviços devem ser apresentados.

Dois aspectos fundamentais da comunicação entre agentes são:

- **Estrutura da Mensagem**; e
- **Transporte da Mensagem**.

Estrutura das Mensagens

A estrutura de uma mensagem (Figura 3.11) é composta pelo emissor, receptor e conteúdo. As mensagens contêm os IdAs do emissor e do receptor, expressos como nomes de agentes. O conteúdo da mensagem é a própria mensagem a ser enviada.

Figura 3.11: Estrutura de uma Mensagem, segundo padrão FIPA.

Transporte das Mensagens

Quando uma mensagem é enviada, esta é codificada em uma informação a ser enviada (*payload*) e incluída num formato chamado "mensagem de transporte". A informação a ser enviada é então codificada para o seu transporte pela rede. A mensagem de transporte é formada pela informação a ser enviada mais o envelope. O envelope inclui informações do emissor e do receptor, além de informações sobre como enviar a mensagem (qual protocolo de transporte é utilizado, qual o endereço de destino). O envelope também pode conter informações adicionais, tais como a codificação utilizada, dados relacionados com a segurança e outros dados para o transporte dessa mensagem até o(s) destinatário(s).

3.4 Redes de Petri

As redes de Petri (RP) (MURATA, 1989) são uma ferramenta matemática e gráfica de uso geral que permite: modelar o comportamento de sistemas dinâmicos a eventos discretos; descrever as relações existentes entre condições e eventos; e visualizar propriedades tais como paralelismo, sincronização e compartilhamento de recursos. Elas foram criadas em 1962 por Carl Adam Petri (PETRI, 1962).

As redes de Petri são utilizadas para representar modelos conceituais de um sistema real. Conforme (HEUSER, 1990), um modelo conceitual é um modelo de uma área de uma organização, o qual não envolve detalhes de implementação e descreve tanto as propriedades estáticas quanto as propriedades dinâmicas do sistema modelado.

Conforme (CARDOSO; VALETTE, 1997), redes de Petri são aplicadas para avaliação de desempenho, análise e verificação formal de sistemas discretos, tais como: protocolos de comunicação, controle de oficinas de fabricação, concepção de software em tempo real e/ou distribuído, sistemas de informação (organização de empresas), sistemas de transporte, logística, gerenciamento de base de dados, interface homem-máquina e multimídia.

Na etapa de modelagem e análise da presente proposta, foram utilizadas:

- **As redes de Petri Coloridas**, usando a ferramenta CPNTools [2];
- **As redes de Petri Estocásticas Generalizadas**, usando a ferramenta GreatSPN [3].

Nesta seção, serão detalhados os conceitos e extensões das redes de Petri.

3.4.1 Extensões às Redes de Petri

O modelo original das Redes de Petri é limitado quando se deseja representar duas importantes características: aspectos funcionais complexos, tais como condições que determinam o fluxo de controle e informação; e os aspectos de temporização. Para enfrentar estas duas limitações, duas classes de extensões às RP foram desenvolvidas:

[2] (http://wiki.daimi.au.dk/cpntools/cpntools.wiki)
[3] (http://www.di.unito.it/~greatspn)

- As RPs de alto-nível (aqui são enfatizadas as Redes de Petri Coloridas, que formam a categoria mais representativa e mais usada das Redes de Petri de alto-nível);
- As RPs com restrições de tempo (dentre elas, as redes de Petri temporizadas: determinísticas e estocásticas generalizadas) (PENHA; FREITAS; MARTINS, 2004).

Redes de Petri Coloridas

As redes de Petri são uma excelente ferramenta para modelagem de sistemas. Entretanto em sistemas complexos, onde existem vários produtos distintos com processos semelhantes (como protocolos de comunicação, por exemplo), elas apresentam algumas limitações. Nestes casos, ao se utilizar Redes de Petri, o projetista tem duas escolhas, como apresentado em (CARDOSO; VALETTE, 1997):

- Modelar o comportamento geral, sem indicar a identidade de cada processo, mas somente seu número; e
- Modelar, individualmente, cada um dos processos que constituem o sistema e modelar a interação existente entre eles, o que consiste, muitas vezes, em desdobrar o modelo que representa o comportamento geral.

No primeiro caso, é evidente a falta de precisão do modelo criado, pois muito do comportamento do sistema é perdido. Já no segundo, o modelo obtido tende a ser demasiadamente grande e complexo para ser entendido. Para resolver tal problema, foram propostas várias extensões às Redes de Petri. Estas abordagens recebem o nome de Redes de Petri de Alto Nível. Tais redes têm sua notação gráfica e definições padronizadas na norma ISO 15909[4].

Uma destas extensões são as Redes de Petri Coloridas (RPC), descritas em (JENSEN; KRISTENSEN; WELLS, 2007). O principal objetivo das RPCs é a redução do tamanho do modelo, permitindo que fichas individualizadas (coloridas) representem diferentes processos ou recursos numa mesma subrede. Elas recebem este nome porque as fichas contêm dados capazes de distingui-las umas das outras em contraponto com às redes de Petri padrão, onde as fichas são indistinguíveis.

[4][ISO/IEC 15909 2002]

Em RPC, as fichas são representadas por estruturas de dados complexas. Desse modo, as fichas podem conter informações. Além disso, cada lugar armazena fichas de um certo tipo definido e arcos realizam operações sobre elas. As transições determinam a dinâmica da RPC e podem apresentar "expressões de guarda". Estas, por sua vez, indicam os tipos de fichas que possibilitam habilitar uma transição. Para que uma transição esteja habilitada em uma RPC, devem existir fichas suficientes nos lugares de entrada da transição. Além disso, estas fichas devem possuir valores que sejam correspondentes às expressões associadas aos arcos que ligam estes lugares à transição.

Uma RPC é composta por três partes: estrutura, inscrições e declarações.

- A estrutura é um grafo direcionado, com dois tipos de nós (lugares e transições), com arcos interconectando nós de tipos diferentes. Graficamente, os lugares são representados por círculos ou elipses, e transições, simbolizadas por retângulos.
- As inscrições são associadas aos lugares, transições e arcos.
- As declarações envolvem tipos, funções, operações e variáveis. Quando a expressão do arco é avaliada, ela gera um multi-conjunto (conjunto com vários tipos) de fichas coloridas.

As RPC suportam hierarquia. Dessa forma, pode-se construir um modelo mais abstrato e, através da hierarquia, refinar o modelo. As RPC usam transições especiais, denominadas transições de substituição, para implementar hierarquias.

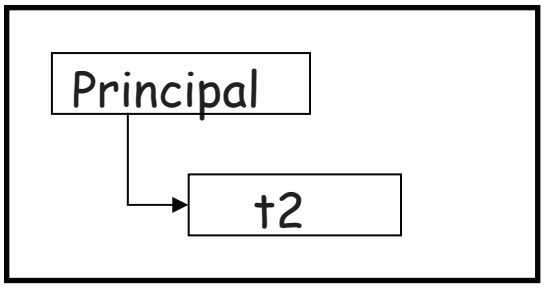

Figura 3.12: Hierarquia em RPC de um sistema de manufatura.

Na Figura 3.12, é apresentada a hierarquia do modelo RPC de um sistema de manufatura apresentado na Figura 3.13. Nessa hierarquia, são apresentadas redes

que compõem o modelo, as quais são: a rede Principal e a sub-rede t2.

Na Figura 3.13, é apresentada a página Principal do modelo e, na Figura 3.14, é apresentada a sub-rede associada à transição de substituição t2.

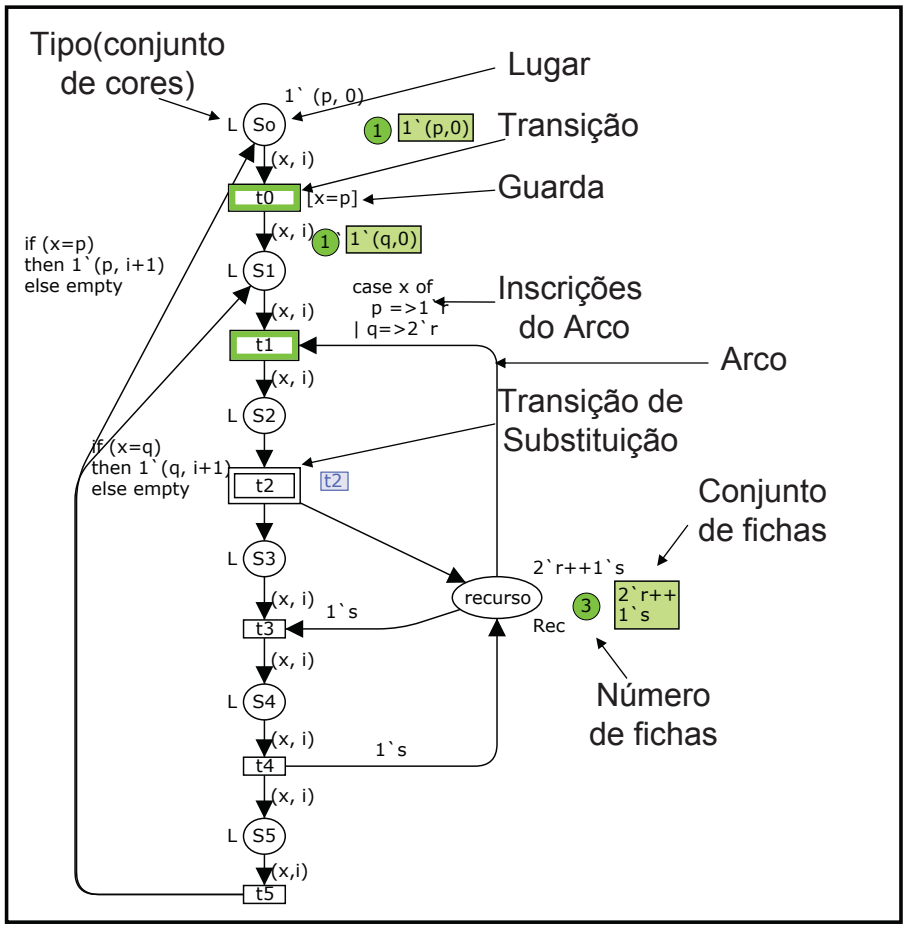

Figura 3.13: Descrição em RPC da página Principal de um sistema de manufatura.

A seguir, são descritos os elementos de uma RPC (Figura 3.13):

- *Lugares*, representados graficamente por um círculo ou uma elipse. Os mesmos contêm as seguintes inscrições:
  - *Nome*: para identificação (por exemplo, o lugar s_0 na RPC apresentada na Figura 3.13);
  - *Tipo* (conjunto de cores): especificando os tipos de fichas que podem residir no lugar (por exemplo, o tipo L associado ao lugar s_0);
  - *Número de fichas*: quantidade de fichas coloridas em um determinado lugar em um dado instante (por exemplo, o lugar *recurso* (Figura 3.13)

possui 3 fichas do tipo *Rec*);

- *Conjunto de fichas*: exemplo de conjunto de cores, onde a ficha 2'r++1's do tipo *Rec* são duas fichas do tipo r e uma do tipo s;
- *Portas de Entrada* (In): lugares de entrada das transições de substituição; e
- *Portas de Saída (Out)*: lugares de saída das transições de substituição.

► *Transições*, representadas graficamente por retângulos ou seguimentos de retas, contêm as seguintes inscrições:

- *Nome*: para identificação (por exemplo, a transição t_0 na RPC apresentada na Figura 3.13); e
- *Expressões de Guarda*: por exemplo, a condição $[x = p]$ associada â transição t_0.

► *Arcos*, ligando transições a lugares e lugares a transições. Os arcos contêm *Inscrições do arco* que definem a habilitação de uma transição e as fichas que serão removidas e adicionadas após o disparo dessa transição (por exemplo, a inscrição *case x of p=>1 ´r | q=>2 ´r* associada ao arco que liga o lugar *recurso* à transição t_1);

► *Transição de substituição*, representada graficamente por um duplo retângulo. Esta transição representa um modelo que deve ser expandido via uma sub-rede associada à essa transição (por exemplo, a transição t_2 da RPC apresentada na Figura 3.13); e

► *Marcação Inicial*, representando a distribuição inicial de fichas nos lugares da rede.

No modelo apresentado na Figura 3.13, os estágios de produção são representados pelos lugares de s_0 a s_5. As fichas de cores (tipos) distintas diferenciam os processos. Os recursos compartilhados são representados por fichas de cores distintas no lugar *recurso*.

A marcação inicial da rede é:

► M(s_0) = 1'(p,0);

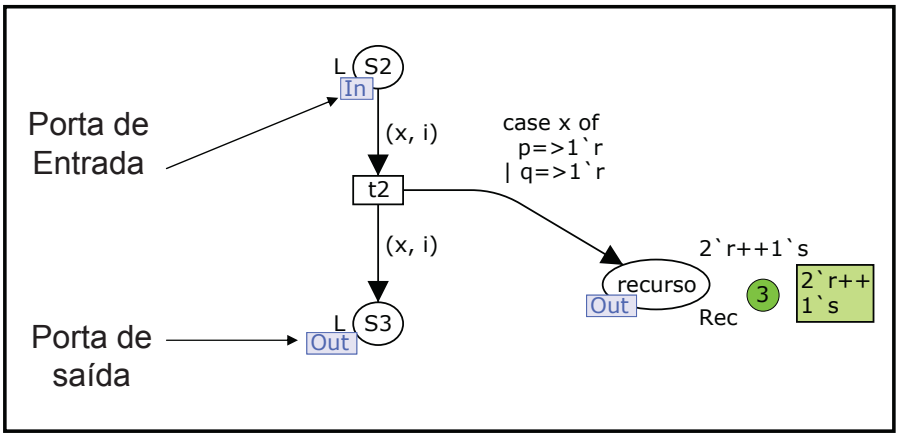

Figura 3.14: Descrição em RPC da página t2 de um sistema de manufatura.

- M(s_1) = 1'(q,0); e
- M(recurso) = 2'r + 1's.

Desse modo, o lugar s_0 possui uma ficha do tipo Proc*I, ou seja, 1'(p,0), onde o tipo do processo é p e o zero corresponde ao número de iterações realizadas. No lugar *recurso*, existem inicialmente duas fichas do tipo r e uma do tipo s.

Inicialmente, as transições t_0 e t_1 estão habilitadas. Para que a transição t_1 seja disparada, a variável x do arco que liga o lugar *recurso* à transição t_1 deve receber uma ficha do tipo q. O disparo da transição t_1 remove uma ficha do tipo (q,0) do lugar s_1 e duas fichas do tipo r do lugar *recurso* e coloca uma ficha do tipo (q,0) em s_2.

Redes de Petri Temporizadas Determinísticas

O conceito de tempo não foi contemplado na definição original das redes de Petri. Contudo, para avaliação de desempenho, é necessária a introdução de retardos de tempo (RAMCHANDANI, 1974). Conforme a proposta de Ramchandani, as formas pelas quais o tempo pode ser especificado são:

- **Associado aos lugares**: onde as fichas armazenadas nos lugares de saída, após o disparo de uma transição, só estarão disponíveis para disparar uma nova transição após um determinado tempo;
- **Associado às fichas**: as fichas possuem uma informação que indica quando esta estará disponível para disparar a transição; e

- **Associado à transição**: que impõe a cada transição um retardo (ou duração) fixo entre o tempo em que ela está habilitada e o tempo de disparo.

Um exemplo de rede de Petri temporizada determinística é apresentado na Figura 3.15.

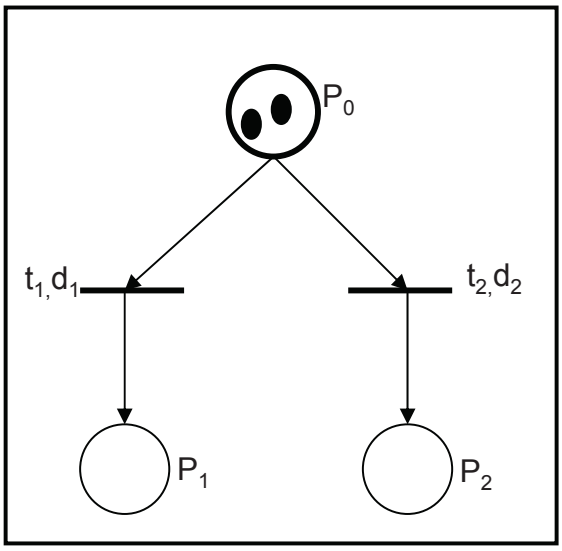

Figura 3.15: Exemplo de Rede de Petri Temporizada.

Nela, as transições t_1 e t_2 possuem tempos associados diferentes (respectivamente, d_1 e d_2). Supondo que $d1 < d2$, então dada a marcação apresentada na Figura 3.15 (2 fichas em P_0), t_1 irá disparar primeiro e uma ficha chegará primeiro ao lugar P_1. Assim, de maneira determinística, pode-se estabelecer a ordem em que os eventos devem ocorrer.

Redes de Petri Estocásticas Generalizadas

As Redes de Petri Estocásticas Generalizadas (GSPN) (CHIOLA et al., 1993) incluem transições temporizadas e não-temporizadas, particionando as transições em dois conjuntos:

- **Transições Imediatas**: uma vez habilitadas, disparam em tempo zero; e
- **Transições Temporizadas**: disparam com tempos aleatórios, descritos por distribuições exponenciais. Nas GSPN, as taxas de disparo estão associadas somente às transições temporizadas.

A definição formal das GSPN é semelhante à das Redes de Petri Estocásticas (BERNARDI; DONATELLI, 2004), diferenciando somente no conjunto das taxas de disparo ou atrasos (λ). Nessa extensão, as expressões de eventos são acrescidas de um intervalo [1, λ], que determina as fronteiras temporais mínima e máxima do disparo da transição. Os parâmetros temporais associados à transição determinam o tempo durante o qual o sistema permanecerá no conjunto de estados de origem, bem como o atraso permitido para que a transição dispare após a fronteira mínima ter sido alcançada. Nas GSPN, esses atrasos são associados somente às transições temporizadas, e não mais a todas as transições, como na redes de Petri Estocásticas.

Segundo Chiola e outros, a diferença fundamental entre as redes de Petri estocásticas e as redes de Petri estocásticas generalizadas está em admitir que as transições também podem ser não-estocásticas, isto é, uma transição também pode ser imediata, como nas redes de Petri. Chiola et al (1993) definiram que as transições imediatas deveriam ter atraso de disparo igual a zero, e que somente as transições estocásticas tinham atrasos associados diferentes de zero.

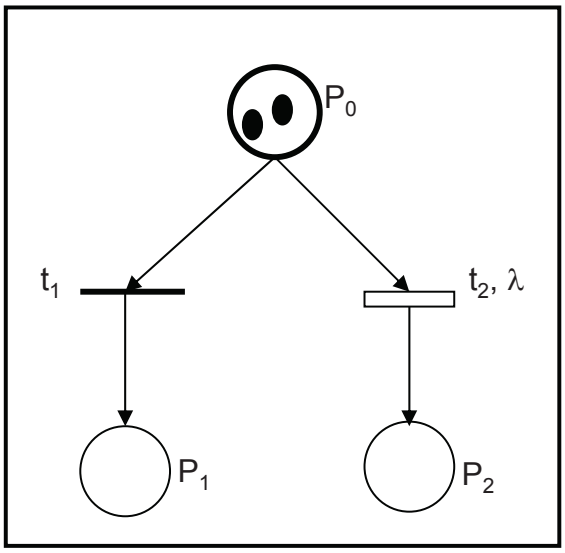

Figura 3.16: Exemplo de Rede de Petri Estocástica Generalizada.

Um exemplo de rede de Petri estocástica generalizada é apresentado na Figura 3.16. Nela, a transição t_1 é imediata e a transição t_2 é temporizada, com atraso de λ.

3.5 Extensão UML-SPT

A *Unified Modeling Language* (UML) é uma linguagem de programação orientada a objetos. Através das extensões da linguagem UML é possível modelar sistemas com restrições de tempo. O modelo *Profile for Schedulability Performance and Time Specification* (SPT ou UML-SPT) estende o padrão UML e tem como objetivo utilizar as funcionalidades da UML para realizar uma análise de desempenho. A grande importância da modelagem é o fato de economizar custos já que erros podem ser eliminados ainda na fase de projeto do sistema. O UML *Profile for Schedulability,Performance and Time Specification* é uma extensão que define uma linguagem para projetar, visualizar, especificar, analisar, construir e documentar os artefatos de testes de um sistema. Através dessa extensão, é possível informar algumas características que descrevem o comportamento de carga e tempo do sistema, como a freqüência esperada de usuários em um dado tempo, qual a probabilidade de uma determinada funcionalidade ocorrer, quanto tempo esta funcionalidade deveria gastar, entre outras características. Este modelo fornece facilidades para:

- Capturar exigências de desempenho dentro do contexto do projeto;
- Associar características de desempenho de QoS (Qualidade de Serviço) com os elementos selecionados de um modelo de UML;
- Especificar os parâmetros da execução que podem ser usados pelo modelo para computar características de desempenho; e
- Apresentar os resultados do desempenho computados pelo modelo ou encontrados nos testes.

O modelo UML-SPT é baseado nos seguintes conceitos: um cenário é uma sequência de etapas (*Steps*), unidas por conectores (*Connectors*). Uma etapa é uma peça sequencial de execução. Os conectores podem incluir ramificações (*branches*), fusões (*merges*), divisões (*forks*) e uniões (*joins*). Os cenários têm os pontos início (*Start*) e fim (*End*), onde eles iniciam e finalizam, respectivamente. Os pontos *Start* estão associados com a carga (*Workload*), que define as chegadas de solicitações e pode ser aberta ou fechada. A carga é aberta quando o número de solicitações atendidas pelo sistema não é fixo, obedecendo-se uma a taxa de chegada, e a carga

é fechada quando se utiliza um número fixo de solicitações. Existem dois tipos de recursos (*Resources*): *Ativos*, que executam etapas, e *Passivos*, que são adquiridos e liberados durante os cenários especiais *ResAcquire* e *ResRelease*.

3.6 Descrição da Linguagem de Programação Java e Java RMI

Java é uma linguagem de programação multiplataforma e possui muitas bibliotecas prontas que facilitam bastante a implementação de um sistema multiagente. A linguagem de comunicação Java RMI (*Remote Method Invocation*) foi utilizada para dar suporte à comunicação entre os agentes, já que esta provê um modelo simples e direto para computação distribuída com objetos Java. A escolha dessa linguagem de programação na implementação dessa arquitetura se deu pelo fato de que a reconfiguração dinâmica de clusters de servidores Web é feito no nível da reconfiguração dinâmica, necessitando ser rápido para não comprometer a velocidade no atendimento das atividades essenciais do cluster.

Aplicações que utilizam RMI são compostas por no mínimo um servidor e um cliente. O servidor implementa os objetos a serem usados remotamente e aguarda que os clientes invoquem métodos destes objetos. O cliente invoca (executa) um ou mais métodos de objetos remotos disponíveis no servidor. Existe ainda o serviço RMI Registry que deve ser executado no mesmo computador onde está o processo servidor; ele mantém uma base de objetos remotos, permitindo ao cliente obter as referências necessárias para que os métodos destes objetos sejam invocados, como mostrado na Figura 3.17. O objeto Remoto é um objeto que pode receber invocações remotas e a para que um objeto possa invocar métodos de um objeto remoto, ele deve ter acesso à referência a esse objeto remoto.

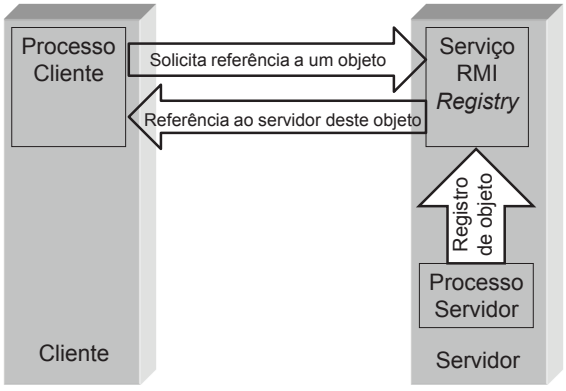

Figura 3.17: Troca de Informações entre o Cliente/Servidor RMI.

A arquitetura Java RMI consiste em três camadas:

- A camada de *stub/skeleton*;
- A camada de referência remota; e
- A camada de transporte.

A camada de *stub/skeleton* é a interface entre a aplicação e as outras camadas que compõem o RMI. Esta camada não trata do transporte em si, mas é responsável por transmitir os dados para a camada de referência remota como um ´´stream" de dados. A camada de referência remota lida com a interface de transporte de baixo nível por meio de um protocolo. Esse protocolo é independente do *stub* do cliente e *skeleton* do servidor. O *stub* e *skeleton* desconhecem os diversos tipos de protocolo que um servidor pode utilizar. A camada de transporte manipula os detalhes da conexão e providencia um canal de comunicação entre as JVMs (*Java Virtual Machine*) executando o código cliente e código servidor. O lado cliente deve possuir a interface do objeto remoto para poder referenciá-lo, enquanto o lado servidor deve possui a implementação deste objeto. Cada camada é definida por uma interface específica e um protocolo. Portanto cada uma delas é independente da outra e pode ser substituída por diferentes tipos de implementação sem afetar as outras camadas no sistema.

3.7 Conclusões do Capítulo 3

Neste capítulo, foram descritos os principais fundamentos utilizados no projeto e implementação da arquitetura DARC: Sistemas Multiagentes, Redes de Petri e suas extensões (Redes de Petri Coloridas, Redes de Petri Temporizadas Determinísticas e Redes de Petri Estocásticas Generalizadas), Extensão UML-SPT e Descrição da linguagem de implementação. Além disso, foi apresentada a plataforma de servidores web à qual a arquitetura DARC foi integrada.

No próximo capítulo será detalhada a arquitetura DARC.

Capítulo 4

Descrição da Arquitetura de Reconfiguração Dinâmica de Recursos

Neste capítulo, é descrita a arquitetura *DARC* (*Dynamic Architecture for Reconfiguration of Web servers Clusters*) proposta neste trabalho. O principal objetivo da arquitetura de realocação dinâmica apresentada é um planejamento do uso dos recursos dos clusters Web de forma dinâmica. Neste trabalho, foi utilizada a plataforma de servidores Web WS-DSAC para integração e validação da proposta, mediante a construção e avaliação de um protótipo integrando a arquitetura DARC à plataforma WS-DSAC.

A arquitetura DARC é composta por um sistema multiagente capaz de adaptar um cluster às constantes mudanças em um ambiente distribuído (e.g., aumentar servidores para uma classe que esteja necessitando de recursos). Os agentes da arquitetura DARC são capazes de reconfigurar os recursos da plataforma de servidores Web através de interações entre elas.

Os agentes são capazes de se adaptar às necessidades atuais de recursos através de experiências passadas para modificar, se necessário, a distribuição dos servidores Web em cada cluster (e.g., alocando um servidor para um cluster que está sobrecarregando).

Este capítulo encontra-se estruturado da seguinte maneira: na Seção 4.1, é apresentada uma visão geral da arquitetura. Na Seção 4.2, são apresentadas as

camadas e níveis da arquitetura. Na Seção 4.3, são apresentados os agentes da arquitetura. Na Seção 4.4, é apresentada a troca de mensagens entre os agentes desta arquitetura. Na Seção 4.5, é apresentada a integração à plataforma WS-DSAC. Na Seção 4.6, é apresentada a descrição dos agentes da arquitetura segundo a especificação FIPA.

4.1 Visão Geral da Arquitetura

Na Figura 4.1, é mostrada uma visão geral da arquitetura de realocação dinâmica de recursos proposta nesta tese. A arquitetura DARC é composta por um conjunto de elementos básicos: Camadas e Níveis, Agentes e Mensagens. As camadas e níveis têm como função organizar as funções dos agentes. Os agentes são responsáveis por realizar a reconfiguração dinâmica dos recursos. As mensagens permitem a troca de informações entre os diversos agentes da arquitetura.

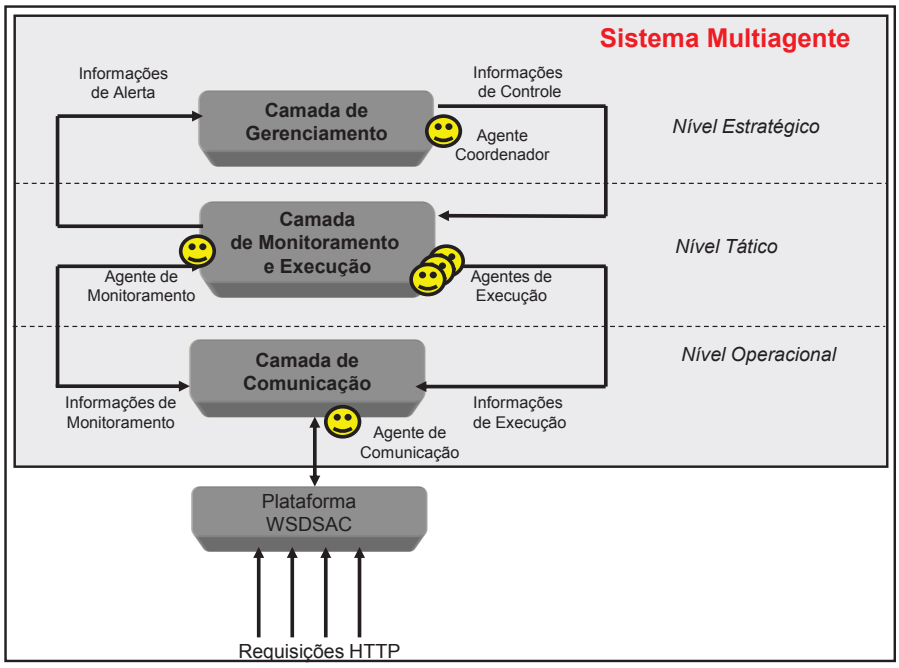

Figura 4.1: Arquitetura DARC em camadas e seus agentes.

A descrição desses elementos é realizada em três etapas. Inicialmente, são descritas as Camadas e Níveis da Arquitetura. Depois, serão descritos os agentes da arquitetura. Por último, descrevem-se as mensagens enviadas e recebidas pelos agentes em cada camada, como apresentados na Figura 4.1.

4.2 As Camadas e Níveis da Arquitetura DARC

Na Figura 4.1 é mostrada a arquitetura de realocação dinâmica de recursos proposta, evidenciando os seus níveis, bem como as suas camadas e seus agentes.

- O primeiro nível é o **Estratégico**. Este nível contém a **Camada de Gerenciamento** e é responsável pela administração das interações entre os vários agentes da arquitetura. Possui o agente-coordenador;
- O segundo nível é o **Tático** e é responsável pelos serviços de monitoramento e execução. Este nível contém a **Camada de Monitoramento e Execução**. Este nível contém os agentes de monitoramento e de execução. Esta camada possui duas funções muito importantes para a arquitetura. A primeira função é de monitoramento. O agente de monitoramento irá monitorar a carga do cluster através da camada de comunicação e, se a carga ultrapassar o limite pré-estabelecido, esse agente envia uma mensagem de alerta ao agente-coordenador. A segunda função da Camada de Monitoramento e Execução é de executar ações. Cada agente de execução executará uma ação diferente;
- O terceiro nível é o **Operacional**. Este nível contém a **Camada de Comunicação** e é responsável por realizar a comunicação entre a Arquitetura DARC e a Plataforma WS-DSAC. Este nível contém o agente de comunicação.

4.3 Os agentes da Arquitetura DARC

Os agentes da arquitetura DARC (Figura 4.1) são instalados em objetos distribuídos e se comunicam usando a abordagem RMI. Eles podem ser replicados em novos objetos distribuídos de acordo com a necessidade.

A seguir, são detalhados os agentes por camada que fazem parte desta arquitetura.

i. **Camada de Gerenciamento**: esta camada contém o agente-coordenador.
   - **Agente Coordenador**: gerencia a interação entre os demais agentes;

ii. **Camada de Comunicação**: esta camada contém o agente de Comunicação.

- **Agente de Comunicação**: realiza a comunicação entre a plataforma WS-DSAC e a arquitetura DARC; e

iii. **Camada de Monitoramento e Execução**: esta camada contém os agentes de monitoramento e execução.

- **Agente de Monitoramento**: realiza o monitoramento constante das cargas dos clusters;
- **Agentes de Execução**: existem três agentes de execução diferentes, apresentados a seguir:
  - **Agente MaximumLoad**: aloca o servidor Web menos carregado para o cluster que está próximo a saturar;
  - **Agente DynamicThreshold**: atualiza os *Thresholds* dos clusters; e
  - **Agente UpdateManager**: controla as atualizações dos *Thresholds* dos clusters. Para isso, verifica as atualizações anteriores feitas pelo *agente DynamicThreshold* para realizar as atualizações posteriores e controlar a quantidade dessas atualizações.

As ações realizadas pelos agentes da arquitetura DARC são mostradas a seguir:

- O *Agente de Comunicação* recebe mensagens geradas pela Plataforma WS-DSAC. A função deste agente é descrita a seguir:
  - O agente de comunicação solicita à Plataforma WS-DSAC qual o valor da variável ρ_{ki} (carga estimada do cluster em um período k) e envia o seu valor através de uma mensagem de monitoramento para o agente de monitoramento.
  - Este agente também solicita à Plataforma WS-DSAC qual é o servidor menos carregado que pode ser alocado de um cluster a outro (caso seja necessário), bem como quais os valores atuais dos *thresholds* de cada cluster.
- O *Agente Coordenador* realiza o planejamento da capacidade de recursos de processamento da plataforma. Este planejamento é realizado quando esse agente administra as interações entre os vários outros agentes da

arquitetura. Por exemplo, verifica as ações a serem tomadas pelos agentes de execução que são acionadas pelas mensagens de alerta enviadas pelo agente de monitoramento. Quando ocorre alguma modificação na carga do cluster o agente-coordenador envia um estímulo ao agente de execução específico.

- O *Agente de Monitoramento* é executado como um *thread* e fica coletando informações do estado de cada cluster, em determinados períodos de tempo, dependendo da carga do computador no qual está instalado. Os intervalos entre os monitoramentos permitirão analisar o comportamento real da carga na plataforma WS-DSAC. O sistema multiagente tomará decisões com base neste comportamento. Estas informações coletadas pelo agente de monitoramento via agente de comunicação servem para ativar as mensagens de controle que são enviadas pelo agente coordenador. A função do agente de monitoramento é descrita a seguir:
  - O *Agente de Monitoramento* monitora a variável ρ_{ki} verificando a média da carga predita para o período de tempo (k+1) dos servidores do cluster da classe i, calculada durante o *k-ésimo* período de tempo.
- Três *Agentes de Execução* estão presentes na arquitetura e realizam tarefas distintas. O algoritmo que explica detalhadamente o funcionamento desse agentes de execução é apresentado na Figura 4.2.
  - O *Agente MaximumLoad* aloca o servidor menos carregado para o cluster que está saturando.
  - O *Agente DynamicThreshold* atualiza os *thresholds* do *cluster* antes que ele sature, atuando apenas caso a ação do *Agente MaximumLoad* não seja suficiente para evitar a saturação do cluster.
  - O *Agente UpdateManager* monitora a atualização feita pelo *Agente DynamicThreshold*.

4.4 As Mensagens Enviadas e Recebidas pelos Agentes da Arquitetura DARC

Nesta seção, são descritas as trocas de mensagens entre os agentes da Arquitetura DARC.

O agente da Camada de Gerenciamento troca as seguintes mensagens:

- **Mensagens de Alerta**: são as mensagens recebidas pelo agente desta camada e enviadas pelo agente de monitoramento quando a carga do cluster ultrapassa o limite pré-determinado pelo administrador da arquitetura de servidores Web; e
- **Mensagens de Controle**: são as mensagens enviadas pelo agente desta camada, ou seja, as ações a serem tomadas pelos agentes de execução.

Os agentes da Camada de Monitoramento e Execução trocam as seguintes mensagens:

- **Mensagens de Controle**: são as mensagens recebidas pelos agentes desta camada que são enviadas pelo agente-coordenador da camada de gerenciamento;
- **Mensagens de Monitoramento**: são as informações solicitadas pelo agente de monitoramento ao agente de comunicação para monitorar a variação da carga dos clusters; e
- **Mensagens de Execução**: são as informações necessárias à execução das ações enviadas pelos agentes de execução.

Os agentes da Camada de Comunicação trocam as seguintes mensagens:

- **Mensagens de Monitoramento**: são as mensagens recebidas pelos agentes desta camada que foram enviadas pelo agente de monitoramento. Estas são as solicitações de informação de carga dos clusters da Plataforma WS-DSAC;
- **Mensagens de Execução**: são as mensagens recebidas pelos agentes desta camada. Estas são as informações necessárias à execução das ações enviadas pelos agentes de execução. Estas também serão solicitações que serão feitas à Plataforma WS-DSAC;
- Os agentes desta camada também recebem as respostas às solicitações feitas à Plataforma WS-DSAC;
- As mensagens enviadas pelos agentes desta camada são as solicitações à Plataforma WS-DSAC.

Um exemplo da troca de mensagens entre os agentes da arquitetura seria:

i. O agente de monitoramento irá monitorar a variável ρ_{ki} (carga do cluster) através da camada de comunicação. Isso é realizado pelo envio de uma solicitação do valor dessa variável pelo agente de monitoramento ao agente de comunicação.

ii. Se esta variável ultrapassar o limite pré-estabelecido definido inicialmente pelo administrador, o agente de monitoramento envia uma mensagem de alerta ao agente-coordenador;

iii. O agente-coordenador recebe a mensagem de alerta e envia uma mensagem de controle ao agente de execução;

iv. O agente de execução recebe a mensagem de execução do agente coordenador e, por exemplo, realoca o servidor menos carregado para o cluster que atingiu o valor máximo da carga permitida pela sua classe.

O algoritmo que controla o funcionamento da arquitetura DARC descreve as ações tomadas pelos Agentes de Execução. Para evitar decisões erradas quando ocorrem picos de carga, usa-se uma variável que armazena a carga média dos clusters (*medLoad*). Para controlar as ações dos agentes de execução, são utilizadas as variáveis de controle. Estas variáveis são compostas dos parâmetros-limite e de atualização. Estas variáveis são descritas a seguir:

- γ_{low} e γ_{high} são parâmetros-limite da carga do cluster, que controlam a atuação dos agentes de execução;
- δ_1 é o parâmetro de atualização dos *thresholds* do cluster.

Os aspectos básicos deste algoritmo são apresentados na Figura 4.2. Adicionalmente, as seguintes situações devem ser consideradas:

- Existem situações que podem comprometer o desempenho da arquitetura de reconfiguração dinâmica, tais como várias atualizações do *threshold* em instantes de tempo muito próximos. Isso poderia acontecer devido à alternância de períodos de sobrecarga e períodos de baixa carga o que levaria o

Se existe um cluster em modo compartilhado então
  WS-DSAC realiza o balanceamento de carga;
senão Se ($medLoad \geq \gamma_{low} * R_{ac}$ e $medLoad < \gamma_{high} * R_{ac}$) então
  o **Agente MaximumLoad** do cluster que está saturando aloca
  o host menos carregado de qualquer cluster para si;
senão se ($medLoad \geq \gamma_{high} * R_{ac}$) então
  o **Agente DynamicThreshold** do cluster em modo saturado
  aumenta o threshold daquele cluster em δ_1.

Figura 4.2: Algoritmo Básico para DARC

Agente DynamicThreshold a modificar o *threshold* constantemente. Para evitar esse problema, o *Agente UpdateManager* irá certificar-se que as atualizações de *thresholds* são adequadas, através de comportamento adaptativo. A adaptação do comportamento é realizada observando-se situações passadas e se o *Agente UpdateManager* detectar várias atualizações de *threshold* em um curto período de tempo (o mesmo utilizado pelo *Agente de Monitoramento*), ele aumentará o *threshold* do *cluster* que está saturando (C1) em δ_1 e diminuirá o threshold do cluster menos carregado (C2) em δ_1 (isto é, C2 alocará recursos próprios na proporção δ_1 para processar requisições para C1);

- De maneira a evitar sobrecarga em um servidor, a atualização dos valores dos thresholds não pode exceder 50% (valor escolhido através de testes) dos valores iniciais destes ($1.5 * R_{ac}$). Isso é utilizado para evitar que novas requições possam ser aceitas, evitando uma sobrecarga; e

- Quando a carga do cluster se estabiliza (isto é, quando o cluster retorna ao modo de trabalho compartilhado), os *thresholds* são reiniciados.

Os valores apropriados para os parâmetros γ_{low}, γ_{high}, e δ_1, foram obtidos experimentalmente. Especificamente, foram iniciados γ_{low} e γ_{high} para 80% e 90% da carga do *cluster*, respectivamente, e o valor de δ_1 para 10%. Estes valores garantiram bons resultados em vários experimentos.

4.5 Integração da Arquitetura DARC à Plataforma WS-DSAC

Na Figura 4.3, é apresentado um diagrama de blocos onde podem ser visualizados:

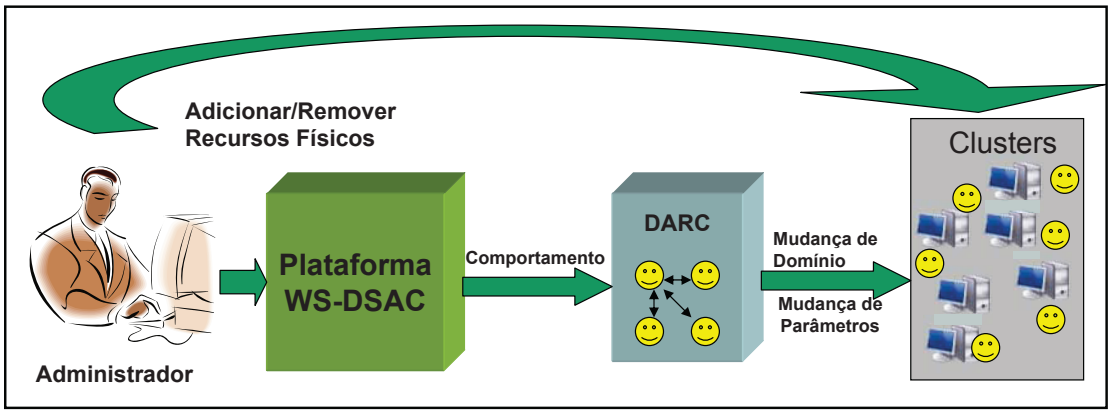

Figura 4.3: Sistema multiagente realizando a reconfiguração dinâmica de recursos.

- O administrador da plataforma, que inicialmente define:
  - Diferentes classes de serviços, bem como as características de cada classe, utilizando o serviço de definição de classes; e
  - Quais clusters e quais servidores estão associados a estas classes.
- O bloco Plataforma WS-DSAC, que representa a arquitetura de servidores Web e a realização do mecanismo de gerência da QoS; e
- O bloco DARC, que representa o sistema multiagente que irá realizar a reconfiguração dinâmica.

As características de cada classe definida pelo administrador estão diretamente relacionadas com o "coeficiente de reatividade" que é requerido para cada requisição de cliente pertencente àquela classe. Quando a plataforma se encontra em regime permanente de funcionamento, estas características mudam constantemente e é necessária a intervenção do administrador para um redimensionamento da mesma. Este trabalho propõe um módulo inteligente que irá interagir com a plataforma e se adaptar a novas situações, auxiliando o trabalho do administrador.

Na Figura 4.4, é apresentada a comunicação da arquitetura DARC com a Plataforma WS-DSAC. Essa comunicação se faz através das solicitações:

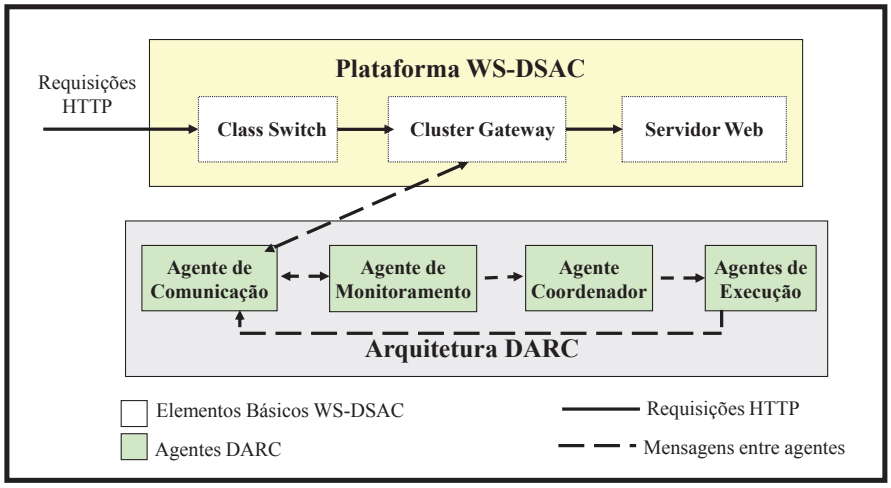

Figura 4.4: Comunicação entre a Arquitetura DARC e a Plataforma WS-DSAC.

i. de cargas feitas pelo *Agente de Comunicação* ao *Cluster Resource Manager* (CRM); e

ii. enviadas pelos *Agentes de Execução* ao *Agente de Comunicação* para realizar atualizações na plataforma WS-DSAC.

Na Figura 4.5 é mostrado como estão distribuídos os agentes da arquitetura DARC. Como a tecnologia de comunicação Java RMI (*Remote Method Invocation*) foi utilizada para dar suporte à comunicação entre os agentes, a localização desses não é fixa, eles poderiam ter sido iniciados em outros nós do ambiente distribuído. Mas utilizou-se essa localização para diminuir a quantidade de mensagens trocadas entre os agentes.

4.6 Especificação dos Agentes DARC segundo a FIPA

Na presente proposta, os agentes foram especificados seguindo algumas das recomendações da FIPA tomando por base os seguintes aspectos:

- A plataforma de agentes é formada pela infraestrutura física da arquitetura DARC que é formada pelos servidores Web que compõem o cluster, além dos agentes. Em cada cluster e em cada servidor, existe uma Máquina Virtual Java

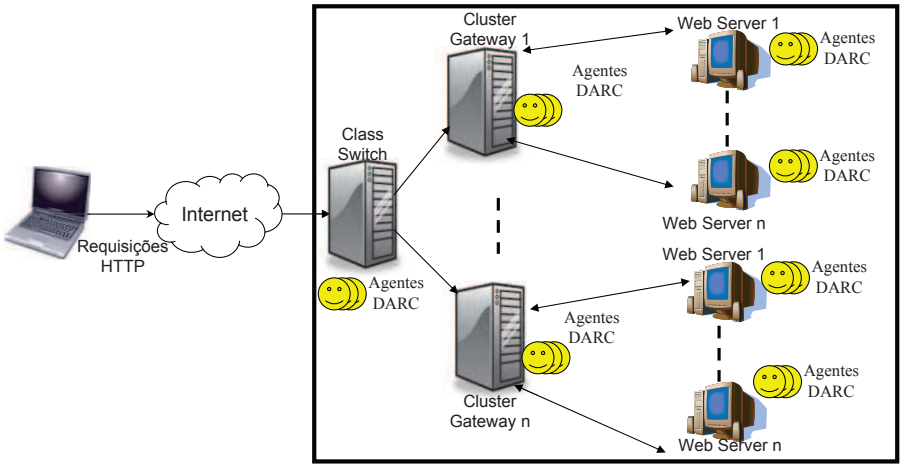

Figura 4.5: Agentes da Arquitetura DARC.

(JVM). Em cada JVM, é criado um ambiente para execução desses agentes, e vários agentes podem executar concorrentemente no mesmo servidor. Isto é, durante a execução da arquitetura, podem co-existir vários agentes em cada JVM. A comunicação entre essas JVMs é realizada pelo RMI (*Remote Method Invocation*) Java.

- O serviço de transporte de mensagens é um dos mais importantes para esta arquitetura. Isso porque é através deste serviço que a arquitetura DARC possibilita a interação dos agentes e a realização da reconfiguração dinâmica;

- Os serviços de gerenciamento e mudança de estado dos agentes são implementados como entidades de software e são acessados através de invocações remotas;

- Os agentes atendem às propriedades essenciais e algumas propriedades desejáveis, tais como: continuidade temporal e capacidade de adaptação;

- A arquitetura de agentes é baseada na troca de mensagens. Os agentes comunicam-se diretamente uns com os outros através da troca de mensagens. Não existe o papel de um intermediário durante o processo de interação, mas existe um agente facilitador de comunicação que é o agente *Coordenador*;

- A linguagem de programação utilizada neste trabalho é a linguagem Java. Além da independência de plataforma e da forte capacidade para trabalhar em ambiente de rede, Java oferece, também, os benefícios da orientação a objetos (OO) e de múltiplas linhas de execução (*multithreading*). A linguagem

também é dinâmica, ou seja, pequenas partes do código Java são montados em tempo de execução (*runtime*) dentro do programa (BICA, 2000).

4.7 Conclusões do Capítulo 4

Neste capítulo foi apresentada uma descrição da arquitetura de realocação dinâmica de recursos, mostrando seus níveis, camadas e agentes. Para validação desta arquitetura foi utilizada a plataforma WS-DSAC para a construção e avaliação de um protótipo do sistema multiagente.

No próximo capítulo será apresentada a Modelagem da arquitetura DARC em redes de Petri Coloridas utilizando a ferramenta CPNTools.

Capítulo 5

Modelagem e Análise da Arquitetura DARC

Neste capítulo, é abordada a modelagem da arquitetura DARC em redes de Petri: Coloridas e Estocásticas Generalizadas. São apresentadas: as descrições dos modelos, os experimentos e a análise dos modelos.

Na Seção 5.1, é apresentada a modelagem em redes de Petri coloridas. Na Seção 5.2, é apresentada a modelagem em redes de Petri estocásticas.

5.1 Modelagem em Redes de Petri Coloridas

São várias as respostas para justificar a modelagem de um sistema computacional. Pode-se afirmar que o uso de um modelo facilita e justifica o prosseguimento de novas etapas de projeto e análise antes da etapa final de avaliação. Além do mais, a medição depende da existência física do sistema ou do protótipo do sistema e que muitas vezes só deve ser realizado após uma fase anterior de validação, antes deste protótipo ser construído. Na modelagem em redes de Petri Coloridas da arquitetura DARC foi realizada uma análise qualitativa com corretude do modelo bem como uma análise de desempenho da arquitetura, utilizando a função monitores da ferramenta CPNTools.

5.1.1 Hierarquia do Modelo

Inicialmente, é apresentada a hierarquia do modelo na Figura 5.1. A hierarquia apresenta uma visão em alto nível do modelo e consiste em um total de quinze

páginas. Neste modelo, os retângulos de onde se originam os arcos representam as páginas e os retângulos onde terminam os arcos denotam as subpáginas. A visão hierárquica foi baseada nas especificações do WS-DSAC (apresentada na Seção 3.1) e da arquitetura DARC (apresentada no Capítulo 4. A modelagem foi feita em redes de Petri Coloridas (JENSEN; KRISTENSEN; WELLS, 2007), utilizando-se a ferramenta CPNTools [1].

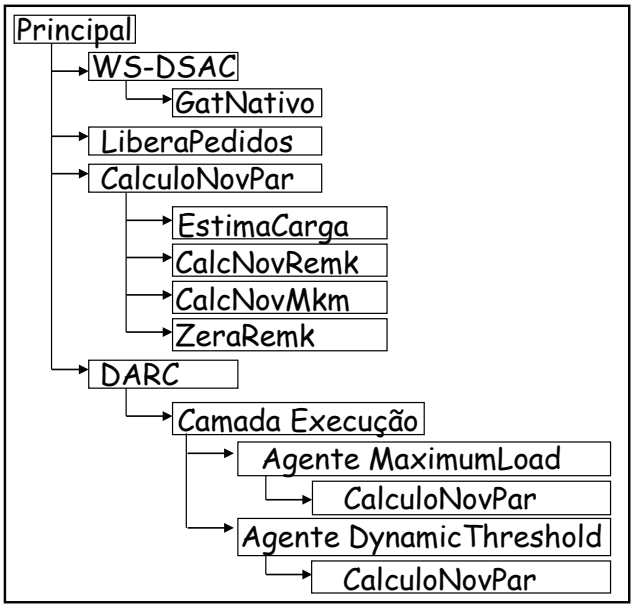

Figura 5.1: Hierarquia de páginas da modelagem em redes de Petri Coloridas.

5.1.2 Páginas do Modelo

A modelagem da arquitetura DARC é baseada na Figura 4.5 (WS-DSAC/DARC), do Capítulo 4. Na Figura 5.2, é apresentada a página Principal da modelagem, em redes de Petri Coloridas. A página Principal representa o funcionamento geral da nova arquitetura e é através dela que são executadas as outras subpáginas (transições de substituição): WS-DSAC, LiberaPedidos, CalculoNovPar e DARC. A modelagem da plataforma WS-DSAC foi realizada numa dissertação de mestrado (ainda em andamento) no Departamento de Teleinformática e a modelagem da arquitetura DARC foi adicionada a esse modelo.

A modelagem da arquitetura DARC foi feita em duas etapas: a primeira

[1] http://wiki.daimi.au.dk/cpntools/cpntools.wiki

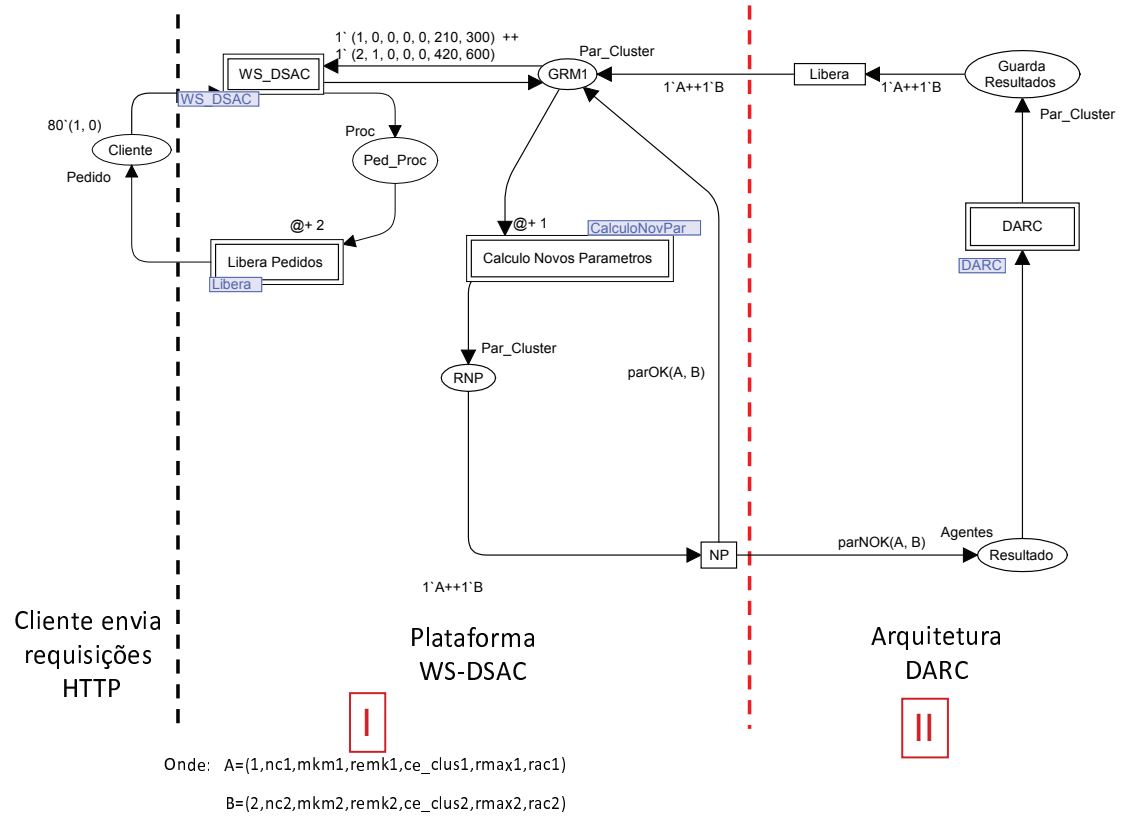

Figura 5.2: Página Principal da modelagem em redes de Petri Coloridas.

modelando a plataforma WS-DSAC (rede I) e a segunda após a inclusão da arquitetura DARC (rede II).

- A subpágina WS-DSAC modela o balanceamento de carga e a diferenciação de serviços realizados pela plataforma WS-DSAC. Esta página é composta pela subpágina GatNativo.
  - A subpágina GatNativo modela as alterações de carga de um servidor.
- A subpágina LiberaPedidos modela a liberação de requisições e cargas dos servidores, após o atendimento das requisições.
- A subpágina CalculoNovPar possui a função de atualização dos parâmetros utilizados pela plataforma WS-DSAC. Esta página é composta pelas subpáginas: EstimaCarga, CalcNovRemk, CalcNovMKM e ZeraRemk.
  - A subpágina EstimaCarga é responsável pelo cálculo de novas cargas estimadas para os servidores pertencentes aos clusters;

- A subpágina CalcNovRemk recalcula o novo R_{emk}, que é o valor do coeficiente de reatividade alcançado pelo cluster de uma classe;
- A subpágina CalcNovMKM recalcula o novo modo de trabalho do cluster;
- A subpágina ZeraRemk terá a função de zerar o R_{emk}, caso seja necessário.

▶ Finalmente, é na página DARC que é modelada a arquitetura DARC. Esta página é composta pela subpágina Camada de Execução. As outras camadas que compõem a arquitetura DARC são modeladas como transições normais (não de substituição) da página DARC.

- A subpágina Camada de Execução modela os agentes de execução que compõem a arquitetura e são descritos no algoritmo apresentado na Figura 4.2. Esta página é composta pelas subpáginas: Agente *MaximumLoad* e *Agente DynamicThreshold*.
  - A subrede Agente *MaximumLoad* modela a atualização dos *thresholds* dos *clusters*. Por isso, essa subpágina chama a subpágina CalcNovPar novamente para uma nova atualização dos parâmetros da plataforma WS-DSAC; e
  - A subrede Agente *DynamicThreshold* modela o controle da atualização dos *thresholds* dos *clusters*.

Na página Principal (Figura 5.2), são representados os lugares ou portas de conexão entre as transições de substituição. Por exemplo, a transição de substituição DARC representa a rede DARC (Figura 5.4), que modela o funcionamento da arquitetura de reconfiguração dinâmica.

Dessa forma, o lugar *Resultado* é o lugar de entrada da transição de substituição DARC na Figura 5.2 que corresponde a porta de entrada *In* para a página DARC no lugar *Controla SMA* da Figura 5.4.

Já o lugar *Guarda Resultados* é o lugar de saída da transição de substituição DARC na Figura 5.2 que corresponde a porta de saída *Out* no lugar *Guarda Resultados* para a página DARC no lugar *Guarda Resultados* da Figura 5.4.

Os lugares da Página Principal do modelo são detalhados a seguir:

- **Cliente**: armazena as requisições a serem processadas; cada ficha nesse lugar é do tipo Pedido, ou seja, cada ficha é do tipo 1'(p, cl), em que p representa o tipo do pedido e cl a sua classe;

- **PedProc**: armazena os pedidos após serem processados; cada ficha nesse lugar é do tipo Proc, ou seja, cada ficha é do tipo 1'(y, s, p, cl), em que:

  - **y**: representa o cluster que atendeu ao pedido, sendo que esse cluster pode ser do tipo "1" ou "2";
  - **s**: representa a situação do pedido após o seu processamento, podendo ser "0", se o pedido foi rejeitado e "1", se o pedido foi atendido.

- **GRM1**: é um lugar que armazena todos os parâmetros de cada cluster que serão necessários ao funcionamento do WS-DSAC e da arquitetura DARC, isto é, ele guarda todos os parâmetros que serão testados, recalculados ou modificados durante todo o processo; cada ficha nesse lugar é do tipo Par_Cluster, ou seja, 1'(y, nc, mkm, remk, ce_clus, rmax, rac), em que:

  - **y**: representa o cluster;
  - **nc**: representa a classe nativa do cluster, podendo ser "0" ou "1";
  - **mkm**: representa o modo de funcionamento do cluster, podendo assumir os valores "0", para o modo compartilhado, "1", para o modo exclusivo. e "2", para o modo saturado;
  - **remk**: é o valor da carga do cluster que é recalculado a cada intervalo de tempo, esta variável pode alcançar o valor máximo da variável rmax;
  - **ce_clus**: representa a carga estimada para o próximo período de tempo referente a cada cluster, e que também é calculada a cada intervalo de tempo;
  - **rmax**: guarda o valor do R_{max} de cada cluster; e
  - **rac**: representa o valor de R_{ac} de cada cluster.

- **RNP**: este lugar recebe os parâmetros de cada cluster após serem recalculados a cada intervalo de tempo; cada ficha nesse lugar é do tipo Par_Cluster, igual ao lugar GRM1 e que já foi detalhado anteriormente;

- **Resultado**: este lugar armazena os parâmetros de cada cluster que deverá ser alterado pela arquitetura DARC;
- **Guarda Resultados**: este lugar recebe os parâmetros de cada cluster após terem sidos processados pela arquitetura DARC; cada ficha nesse lugar é do tipo Par_Cluster, detalhado anteriormente.

As transições da Página Principal do modelo são detalhadas a seguir:

- **WS-DSAC**: é uma transição de substituição que representa o funcionamento do WS-DSAC;
- **Calculo Novos Parametros**: transição de substituição que representa o cálculo dos novos parâmetros de cada cluster feito a cada intervalo de tempo;
- **NP**: esta transição, quando disparada, envia os novos parâmetros, após serem recalculados, para serem testados (função parOK), se devem voltar ao lugar de fusão GRM1 com esse valores, ou se devem ser processados pela arquitetura DARC;
- **DARC**: transição de substituição que representa a execução da arquitetura DARC;
- **Libera**: transição que representa a liberação dos parâmetros de cada cluster pela arquitetura DARC, representados por fichas no lugar de fusão GRM1. Nesse estado os parâmetros podem ser usados pelo WS-DASC; e
- **Libera Pedidos**: transição que representa a liberação de pedidos após todo o processamento.

Quando um pedido chega (representado por fichas no lugar *Cliente* Figura 5.2), a transição de substituição WS-DSAC é executada e é realizado o controle de admissão e o balanceamento de cargas. Ao ser disparada a transição WS-DSAC buscará do lugar de fusão GRM1 todos os parâmetros de cada cluster necessários ao seu funcionamento.

Logo após esta transição ser disparada, são adicionadas duas fichas:

- uma ficha ao lugar *PedProc*. A chegada de uma ficha neste lugar possibilita o disparo da transição de substituição temporizada *Libera Pedidos*, ou seja, existindo pedidos que já foram processados, quer tenham sidos atendidos ou não, a transição Libera Pedidos é habilitada e os pedidos serão liberados, como também as cargas dos clusters que os atenderam, caso eles tenham sidos aceitos; e

- e outra ao lugar GRM1, o que possibilita o disparo da transição de substituição temporizada *CalcNovPar*. Esta transição é disparada em intervalos de tempo pré-definidos e realiza a atualização dos parâmetros usados pela plataforma WS-DSAC.

Após o disparo da transição *CalcNovPar*, esses parâmetros serão testados se devem voltar ao sistema com esses valores ou se devem ser alterados, ou seja, após cada atualização dos parâmetros utilizados pela plataforma WS-DSAC, uma ficha será armazenada no lugar *RNP* (Figura 5.2), o que possibilita o disparo da transição *NP*. No disparo desta transição é realizada uma verificação da necessidade ou não de ser feita uma realocação de recursos no sistema. Este teste é realizado através da função ParOK, associada aos arcos de saída da transição.

Se *ParOK* retornar verdadeiro será armazenada uma ficha no lugar *GRM1* e não será necessária a realização da reconfiguração. Neste caso, os parâmetros do WS-DSAC que foram atualizados voltam novamente para serem usados pela rede e o processamento de pedidos continua. Caso contrário (ParOK falso, representado pela função ParNOK), uma ficha será armazenada no lugar *Resultado*, e então, será possível o disparo da transição de substituição *DARC*, significando que será realizada uma reconfiguração dinâmica dos recursos da plataforma WS-DSAC.

Na rede da Figura 5.3, é apresentada a subrede que detalha a transição de substituição CalculoNovPar. Em intervalos de tempo pré-definidos, a transição de substituição CalculoNovPar é ativada, atualizando os parâmetros usados pela plataforma WS-DSAC.

Os lugares da subrede CalcNovPar são detalhados a seguir:

- **Recalcula**: este lugar representa a porta de entrada (inscrição *In*), por onde chegam os parâmetros dos clusters; cada ficha neste lugar é do tipo Par_Cluster, já detalhado anteriormente;

- **CRM2**: é um lugar de fusão que armazena as cargas dos servidores de cada cluster; cada ficha deste lugar é do tipo Cargas_Est, ou seja, 1'(1,r1k1,r1k2)++1'(2,r2k1,r2k2), em que:
  - **1**: representa cluster 1;
  - **r1k1**: representa a carga do servidor 1 do cluster 1;
  - **r1k2**: representa a carga do servidor 2 do cluster1;
  - **2**: representa cluster2;
  - **r2k1**: representa a carga do servidor 1 do cluster 2;
  - **r2k2**: representa a carga do servidor 2 do cluster 2.
- **GRMaux**: é um lugar que recebe e armazena os parâmetros dos clusters para serem atualizados; cada ficha neste lugar é do tipo Par_Cluster que já foi explicado anteriormente;
- **Inicio CalNovPar**: este lugar armazena as cargas dos servidores de cada cluster; cada ficha neste lugar é do tipo Cargas_Est e já foi explicado anteriormente;
- **MedEst1Cluster**: este lugar armazena as novas médias de cargas estimadas dos clusters para o próximo período de tempo; cada ficha neste lugar é do tipo CE_CLUSTERS, ou seja 1'(Ce_Clus1,Ce_Clus2), em que:
  - Ce_Clus1 representa a carga estimada para o cluster 1; e
  - Ce_Clus2 representa a carga estimada para o cluster 2.
- **Inicia_Remk**: este lugar armazena os parâmetros de cada cluster, já com as suas novas cargas médias estimadas para o próximo período de tempo; cada ficha neste lugar é do tipo Par_Cluster que já foi detalhado anteriormente;
- **GuardaPar**: armazena as cargas estimadas dos clusters para serem usadas posteriormente; cada ficha neste lugar é do tipo Cargas_Est, que já foi explicado anteriormente;
- **Modo Trabalho**: armazena os parâmetros de cada cluster após o cálculo dos novos valores de carga (Remks), ou seja, aqui os parâmetros dos clusters já representam as novas cargas estimadas e os novos Remks; cada ficha neste lugar é do tipo Par_Cluster, já detalhado anteriormente;

- **Testa Limite**: armazena os parâmetros dos clusters após o cálculo do modo de trabalho de cada cluster, ou seja, aqui os parâmetros dos clusters já estão com as novas cargas estimadas, com os novos valores de carga e com seus novos modos de trabalho; cada ficha neste lugar é do tipo Par_Cluster, já detalhado anteriormente;

- **RecOut**: armazena as cargas dos servidores dos clusters, que foram usadas para o cálculo das novas médias de cargas estimadas, para voltarem novamente para serem usadas pelo sistema; e

- **EnviaNovosParametros**: armazena os novos parâmetros dos clusters que foram recalculados e que serão devolvidos para o sistema; é a porta de saída da transição de substituição CalculoNovPar (inscrição *Out*); cada ficha neste lugar é do tipo Par_Cluster e já foi explicado anteriormente.

As transições da Página CalcNovPar são detalhadas a seguir:

- **Processa_NovoCalc**: esta transição tem a função de receber os parâmetros dos clusters da porta de entrada Recalcula para enviá-los para o lugar GRMaux, como também receber as cargas atuais dos servidores de cada cluster para enviá-las para o lugar InicioCalNovPar;

- **Estimacao Carga**: transição de substituição que representa os cálculos das novas médias de cargas estimadas dos clusters;

- **NovoCe_Clus**: tem a função de substituir as novas cargas médias estimadas dos clusters nas fichas que contêm os parâmetros dos mesmos;

- **Novo_REMK**: transição de substituição que representa o cálculo dos novos Remks dos clusters;

- **Novo_MKM**: transição de substituição que representa o cálculo dos novos modos de trabalhos dos clusters;

- **Zera_Remk**: transição de substituição que testa se a variável Remk de cada cluster precisa ser zerada ou não. Se "sim", os parâmetros de cada cluster são atualizados e podem ser utilizados pelo sistema; e

- **RetCRM**: retorna os valores de cargas dos servidores dos clusters ao lugar de fusão CRM2, para serem utilizados pelo sistema novamente.

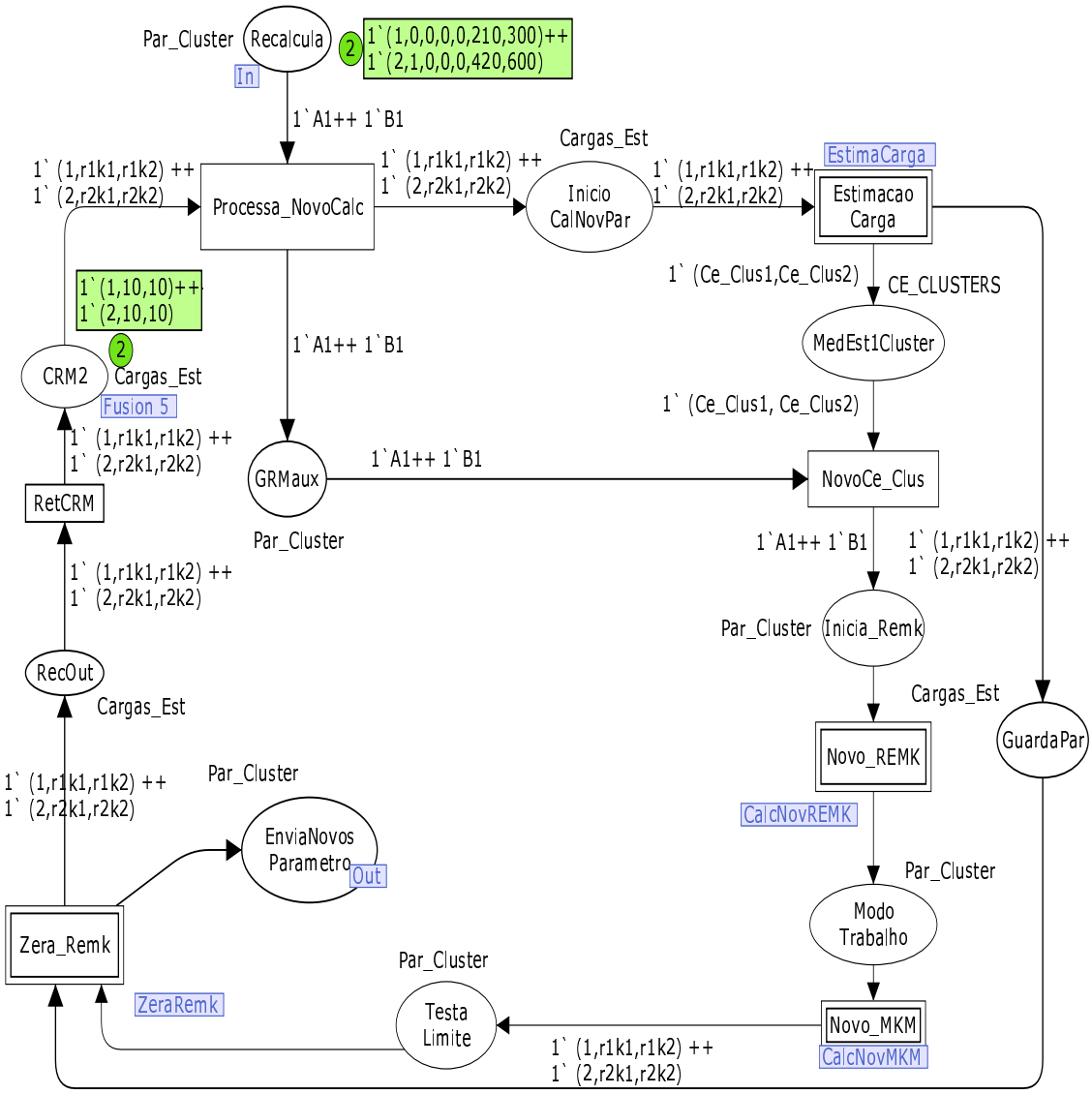

Figura 5.3: Página CalculoNovPar.

Em intervalos de tempo (pré-definidos no modelo pela inscrição @+1) são retiradas do sistema as fichas do lugar GRM1 e armazenadas no lugar *Recalcula* (que é um lugar de entrada, ou seja, de comunicação com a rede *Principal*). Essas fichas contêm os parâmetros dos clusters. Também são retiradas as fichas do lugar de fusão CRM2, que contêm as cargas atuais dos servidores dos clusters, para que se dê início ao cálculo dos novos parâmetros.

Com fichas no lugar *Recalcula*, a transição *Processa_NovoCalc* está habilitada e pode disparar. Após o seu disparo, serão armazenadas fichas nos lugares:

- *GRMaux*: armazena os parâmetros e os guardará para serem usados depois; e

- *Inicio CalcNovPar*: armazena as cargas dos servidores.

Com ficha no lugar *Inicio CalcNovPar* (porta de entrada da subpágina), a transição de substituição *Estimacao Carga* pode disparar para calcular a estimativa de carga dos clusters para o próximo período k. Após o seu disparo, serão armazenadas fichas nos lugares:

- *MedEst1Cluster*: armazena os resultados do cálculo das médias das cargas; e
- *GuardaPar*: armazena os valores atuais das cargas dos clusters.

Após a chegada de uma ficha no lugar *MedEst1Cluster*, a transição $NovoCe\_Clus$ irá disparar e armazenar uma ficha no lugar $Inicia\_Remk$. A chegada de fichas nesse lugar irá disparar a transição de substituição $Novo\_REMK$ para realizar o cálculo do novo valor da variável R_{emk}. Após o disparo desta transição, será armazenada uma ficha no lugar *Modo Trabalho*, o que possibilitará o disparo da transição de substituição $Novo\_MKM$ para verificar se o modo de trabalho mudou.

Após o disparo desta transição, será armazenada uma ficha no lugar *Testa Limite*, o que possibilitará o disparo da transição de substituição $Zera\_Remk$ para verificar se a variável R_{emk} necessitará ser zerada.

Após o seu disparo, serão armazenadas fichas nos lugares:

- *Envia Novos ParametrosOut*, que é um lugar de saída da subrede; e
- *RecOut*. A chegada de fichas nesse lugar permite o disparo da transição *RetCRM* e o seu disparo armazena fichas no lugar *CRM2*, que é o lugar de fusão com a subrede *GatNativo*.

Na Figura 5.4, é apresentada a rede que detalha a transição de substituição DARC.

Os lugares da subrede DARC são detalhados a seguir:

- **Controla SMA**: este lugar representa a porta de entrada (inscrição *In*) dessa subpágina, por onde chegam os parâmetros dos clusters; cada ficha neste lugar é do tipo Agentes, ou seja, cada ficha é do tipo 1‘ (Par_Cluster, *av*), em que *av* contém a informação de carga dos clusters;

- **Envia Inf. Gerenciamento**: este lugar armazena as informações de gerenciamento (detalhadas no Capítulo 4) a serem enviadas aos agentes de execução;
- **Envia Inf. Execução**: este lugar armazena as informações de execução (detalhadas no Capítulo 4) enviadas pelo agente coordenador;
- **Guarda Resultados**: este é o lugar de saída da subrede.

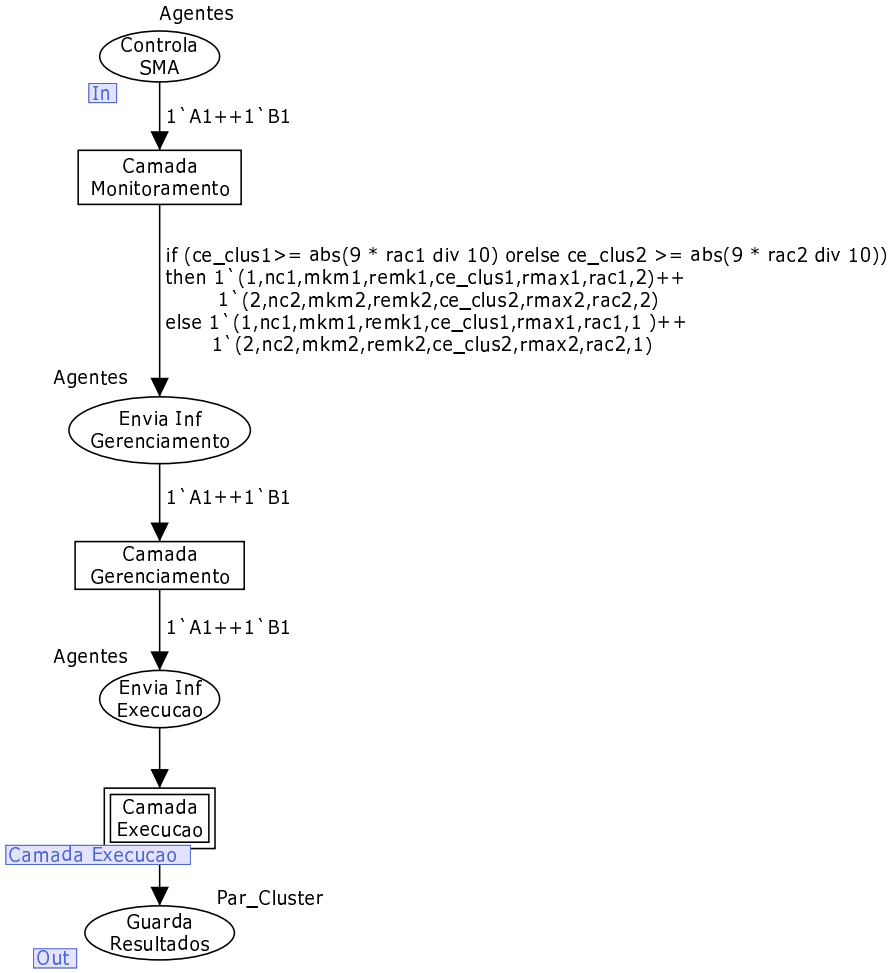

Figura 5.4: Página DARC.

As transições da subrede DARC são detalhadas a seguir:

- **Camada Monitoramento**: esta transição, quando disparada, envia a informação de monitoramento (detalhada no Capítulo 4) à camada de gerenciamento; e

- **Camada Gerenciamento**: esta transição quando disparada envia a informação de execução à camada de execução;
- **Camada Execução**: esta é uma transição de substituição que representa o funcionamento da camada de execução.

As condições para a ocorrência desta transição podem ser visualizadas na Figura 5.2. Através da inscrição do arco de entrada no lugar Resultado (Figura 5.2), se a carga do cluster em questão não estiver de acordo com as condições impostas, isto é, se não estiver dentro do limite estabelecido para a sua classe nativa (definido pelo administrador), o agente de monitoramento enviará a informação de disparo ao agente coordenador e este ativará o agente de execução (transição Camada Execução apresentada na Figura 5.4 e detalhada através da subrede da Figura 5.5). O disparo dessas transições possibilita a verificação das propriedades essenciais dos agentes da arquitetura DARC, por exemplo, a autonomia e a habilidade social.

Na Figura 5.5, é apresentada a rede que detalha a transição de substituição *Camada de Execução*. É nesta rede que é realizado o teste para verificar qual Agente de Execução irá atuar.

Os lugares da subrede *Camada de Execução* são detalhados a seguir:

- **Envia Inf. Execução**: este lugar representa a porta de entrada (inscrição *In*), por onde chegam os parâmetros dos clusters; cada ficha neste lugar é do tipo Agentes; e
- **Guarda Resultados**: este é o lugar de saída (inscrição *Out*) da subrede, sendo que as fichas nesse lugar são do tipo Par_Cluster.

As transições da subrede *Camada de Execução* são detalhadas a seguir:

- **Agente MaximumLoad**: esta transição permite a execução do *agente MaximumLoad* e a alocação do servidor menos carregado para o cluster que está prestes a saturar; e
- **Agente DynamicThreshold**: esta transição permite a execução do *agente DynamicThreshold* e a atualização dos *thresholds* dos clusters.

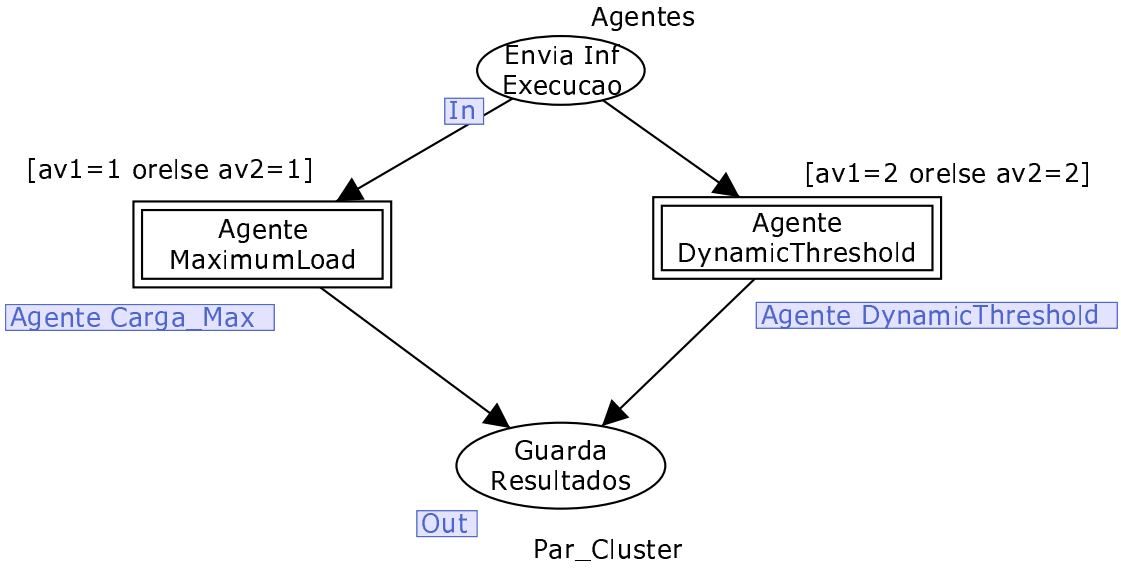

Figura 5.5: Página Camada de Execução.

A chegada de fichas no lugar *Envia Inf Execução* possibilita a verificação da propriedades essencial dos agentes da arquitetura DARC, por exemplo, a capacidade de reação, ou seja, se chegar fichas do tipo Agentes com *av1=1 ou av2=1* o Agente *MaximumLoad* irá executar, caso contrário, será o Agente *DynamicThreshold* que irá executar .

Na Figura 5.6, é apresentada a rede que detalha a transição de substituição *AgenteMaximumLoad*.

Os lugares da subrede *Agente MaximumLoad* são detalhados a seguir:

- **Envia Inf. Execução**: este lugar representa a porta de entrada (inscrição *In*), por onde chegam os parâmetros dos clusters; cada ficha neste lugar é do tipo Agentes;
- **Recalcula**: este lugar armazena fichas com os valores dos thresholds dos clusters atualizados e suas fichas são do tipo Par_Cluster; e
- **Envia**: este é o lugar de saída (inscrição *Out*) da subrede, sendo que as fichas nesse lugar são do tipo Par_Cluster.

As transições da subrede *Agente MaximumLoad* são detalhadas a seguir:

- **Atualiza Rac**: esta transição realiza a atualização dos *thresholds* dos clusters; e

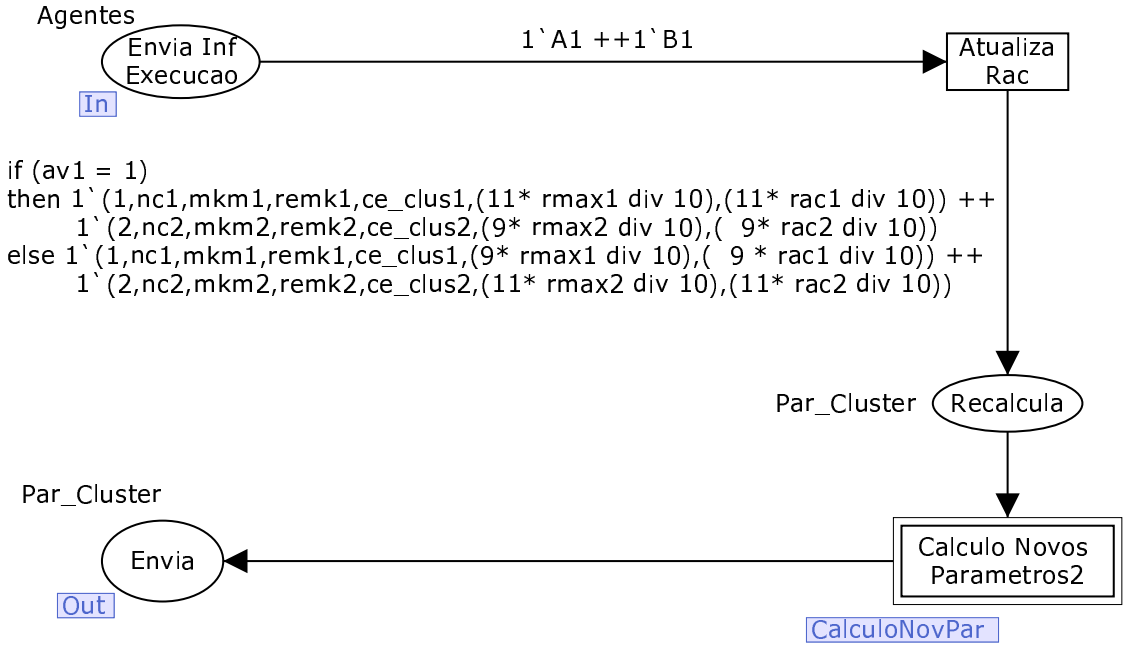

Figura 5.6: Página Agente MaximumLoad.

- **Calculo Novos Parametros2**: esta é uma transição de substituição que representa a execução da transição CalculoNovPar, ou seja, a necessidade de uma atualização dos parâmetros após a alteração dos *thresholds* dos clusters.

É nesta rede que é realizada a atualização dos parâmetros dos clusters caso a carga do cluster ultrapasse o valor-limite de 80% da variável R_{ac}. Esta rede realiza uma nova chamada à rede CalculoNovPar(2), como é mostrado na Figura 5.1. Após a execução dessa transição, os cálculos dos parâmetros usados pelo WS-DSAC são refeitos, visto que recursos foram realocados e a configuração dos clusters provavelmente mudou.

Na Figura 5.7, é apresentada a rede que detalha a transição de substituição AgenteDynamicThreshold. É nesta rede que é realizada a atualização dos parâmetros dos clusters caso a carga do cluster ultrapasse o valor-limite de 90% da variável R_{ac}.

Os lugares da subrede *Agente DynamicThreshold* são detalhados a seguir:

- **Envia Inf. Execução**: este lugar representa a porta de entrada (inscrição *In*), por onde chegam os parâmetros dos clusters; cada ficha neste lugar é do tipo Agentes;

- **Recalcula**: este lugar armazena fichas com os valores dos *thresholds* dos clusters atualizados e suas fichas são do tipo Par_Cluster; e
- **Guarda Resultados**: este é o lugar de saída (inscrição *Out*) da subrede, sendo que as fichas nesse lugar são do tipo Par_Cluster.

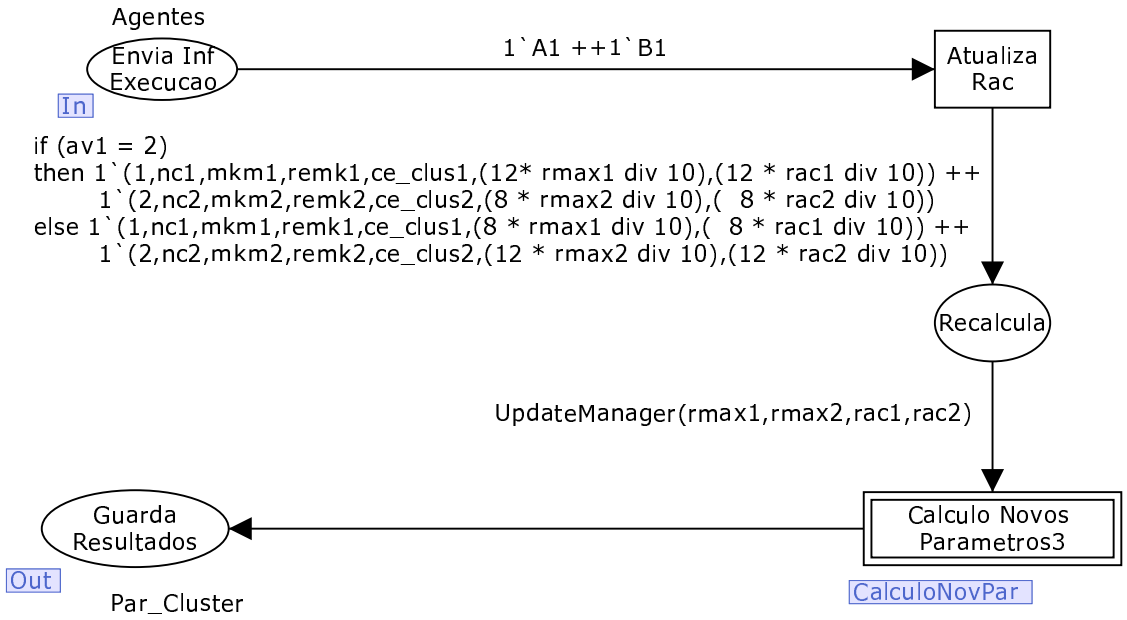

Figura 5.7: Página Agente Dynamic Threshold.

As transições da subrede *Agente DynamicThreshold* são detalhadas a seguir:

- **Atualiza Rac**: esta transição realiza a atualização dos *thresholds* dos clusters; e
- **Calculo Novos Parametros3**: esta é uma transição de substituição que representa a execução da transição CalculoNovPar, ou seja, a necessidade de uma atualização dos parâmetros após a alteração dos *thresholds* dos clusters.

A partir dessa modelagem apresentada, foram realizados vários experimentos simulando diferentes cenários de funcionamento do sistema, com a finalidade de constatar a capacidade da arquitetura DARC em garantir a QoS contratada para cada classe de serviços em momento de utilização crítica dos recursos. Esses experimentos foram realizados utilizando-se a ferramenta CPNTools e serão apresentados a seguir.

Tabela 5.1: Parâmetros das Classes de Serviços.

	Classe 0	Classe 1
R_{ac}	300	600
R_{max}	210	420

5.1.3 Simulação do Modelo e Análise dos Resultados

Os experimentos foram realizados em duas etapas:

i. Na primeira etapa, as simulações foram realizadas sem realocação dinâmica de clusters, ou seja, foi simulada apenas a plataforma WS-DSAC usando apenas a rede I (Figura 5.2);

ii. Na segunda etapa, a arquitetura DARC foi adicionada ao WS-DSAC, e a rede II (Figura 5.2) foi adicionada à rede I, da mesma figura.

Para as simulações, dois domínios diferentes de classes de pedidos foram definidos na plataforma (classes 0 e 1) cujos parâmetros são apresentados na Tabela 5.1 e são descritos a seguir. Essa quantidade de classes, de clusters e de servidores por clusters foi escolhida para realizar uma comparação com os testes realizados da plataforma WS-DSAC.

- R_{max} é o valor-limite que o coeficiente de reatividade do cluster da classe pode assumir para que o mesmo trabalhe no modo compartilhado. Quando o coeficiente de reatividade estiver acima deste valor, o cluster trabalha em "modo exclusivo"; e
- R_{ac} é o valor-limite que o coeficiente de reatividade do cluster da classe pode assumir para trabalhar no modo exclusivo. Quando o coeficiente de reatividade estiver acima deste valor, o cluster trabalha em "modo saturado".

Estes valores iniciais foram determinados através de vários experimentos realizados dentro do contexto da plataforma WS-DSAC (SERRA et al., 2005): estes valores apresentaram bons resultados em vários experimentos realizados.

Foi usada na simulação uma arquitetura contendo dois clusters (clusters 0 e 1), cada um possuindo dois servidores, sendo atribuído a cada cluster uma classe nativa

de serviços a serem atendidos, ficando o cluster 0 associado à classe 0 e o cluster 1 à classe 1.

Na simulação usando a arquitetura DARC, para representar o aumento de recursos em um cluster, ou seja, a retirada de um servidor de um cluster e a sua adição ao outro cluster, aumentaram-se os parâmetros de uma classe de serviços, na mesma proporção em que reduziram-se os parâmetros da outra.

Após as simulações serem feitas, os resultados foram comparados e serão apresentados nas seções seguintes.

Momentos Críticos

No primeiro experimento, foi simulada uma situação em que um cliente envia requisições pertencentes à classe "0" (uma a cada 1ms) ao mesmo tempo em que um outro cliente envia requisições pertencentes à classe "1" (uma a cada 1ms), sendo as quantidades de requisições iguais para as duas classes.

Inicialmente, foram realizados experimentos com a plataforma WS-DSAC sem a arquitetura DARC. Este experimento é chamado *no-DARC1-sim*. Depois, foram realizados experimentos com reconfiguração dinâmica, ou seja, foi avaliada a plataforma WS-DSAC integrada à arquitetura DARC. Este experimento é chamado *DARC1-sim*.

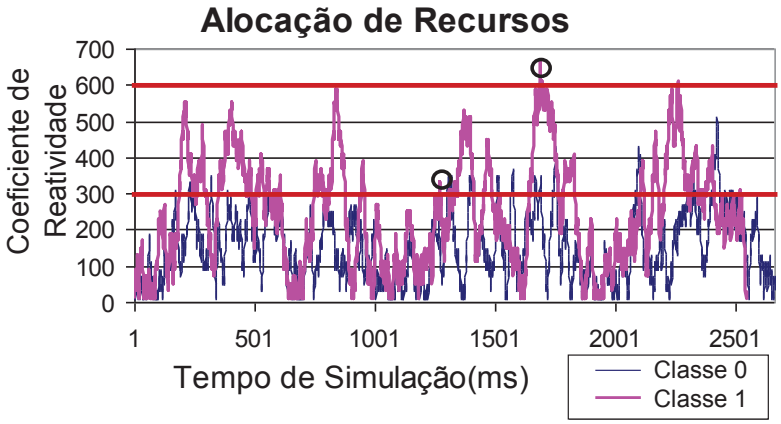

Figura 5.8: Alocação de Recursos com carga nativa das duas classes sem a Arquitetura DARC (no-DARC1-sim).

Na Figura 5.8, é apresentada a distribuição de cargas entre os dois clusters ao longo do tempo, usando *no-DARC1-sim*. Observando o comportamento dos

clusters neste experimento, pode-se notar três situações particulares (representadas por círculos):

- No primeiro círculo: As simulações mostram as primeiras rejeições da classe 0 (Coeficiente de Reatividade - *Reactivity Coefficient* (RC) RC > 300) no instante t = 168 ms;
- No segundo círculo: No instante 732 ms, o cluster 0 muda o seu modo de funcionamento para o modo exclusivo (RC=210). Neste instante, o cluster 1 passa a atender as requisições em modo compartilhado, visto que seu RC é ainda menor que 420 (limite para que ele passe ao modo exclusivo), ou seja, além de atender as requisições da sua classe nativa, ele passa a atender também as requisições da classe 0, sem saturar. Esta distribuição de cargas não impede o sistema de rejeitar requisições, pois algumas vezes os clusters ultrapassaram os seus limites máximos de regime compartilhado; e
- No terceiro círculo: Nesse instante, os cluster 0 e 1 estão no modo compartilhado.

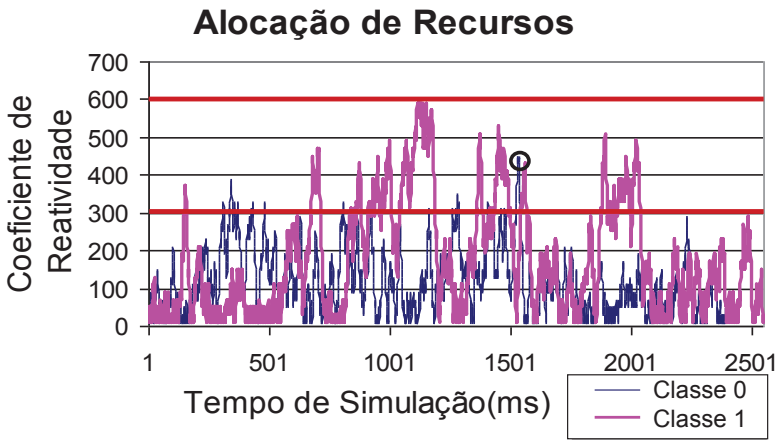

Figura 5.9: Alocação de Recursos com carga nativa das duas classes com a Arquitetura DARC (DARC1-sim).

Na Figura 5.9, são apresentados os benefícios da reconfiguração dinâmica propiciada pela arquitetura DARC, usando *DARC1-sim*. Observando o comportamento dos clusters neste experimento, pode-se notar uma situação (representada pelo círculo particular):

- A distribuição de cargas entre os dois clusters ao longo do tempo, é mostrada na Figura 5.9, usando o experimento *DARC1-sim*. Esse experimento foi realizado com a mesma quantidade de requisições enviadas no experimento *no-DARC1-sim*. Pode-se observar pela Figura 5.9 que o cluster 1 não saturou e que o cluster 0 chegou em menor número de vezes aos seus limites de carga;
- Comparando-se com os resultados da simulação da Figura 5.8, na Figura 5.9 o cluster 1 se manteve sempre com o RC abaixo de 600, ao contrário do que aconteceu na Figura 5.8, onde ele ultrapassa este limite causando uma maior taxa de rejeição de requisições; e
- Pode-se constatar também que, na Figura 5.9, o cluster 0 obteve o valor máximo de coeficiente de reatividade (RC=450) contra o valor máximo (RC=510) na Figura 5.8.

Conclui-se portanto, que o uso da arquitetura DARC realizou uma reconfiguração dinâmica de recursos entre os dois clusters, o que levou consequentemente, à diminuição do índice de saturação dos mesmos. Em ambos os casos (Figuras 5.8 e 5.9), foram feitas várias simulações com o objetivo de calcular a taxa média de rejeições, sendo observado que, no primeiro caso, a taxa média de rejeição das requisições foi de 3,93% enquanto para o segundo foi de 0,04%, comprovando uma melhor utilização dos recursos do sistema.

Momentos de Saturação

Com o objetivo de forçar uma saturação no sistema, foram feitas simulações em que apenas são enviadas requisições pertencentes à classe 0 (Figuras 5.10, 5.11, 5.12 e 5.13). Dessa forma, o cluster pertencente à classe 0 irá saturar logo, porque esse cluster possui os menores limites de carga. Essa simulação foi realizada para se analisar o comportamento dos agentes de monitoramento, ou seja, verificar se eles tomariam uma atitude de reconfiguração de forma mais rapidamente.

A Figura 5.10 se refere aos resultados da simulação sem o uso da arquitetura DARC (denominada no-DARC2-sim), ao contrário das Figuras 5.11, 5.12 e 5.13 em que a arquitetura DARC foi aplicada. Estes experimentos são chamados *DARC2-sim*, *DARC3-sim* e *DARC4-sim* respectivamente. É importante citar que o percentual monitorado de carga dos clusters, pelos agentes, foi modificado em diferentes níveis, sendo de 80% no primeiro caso (Figura 5.11), 85% no segundo

(Figura 5.12) e 90% no último (Figura 5.13), e que em todos os três casos foram deslocados, quando necessários, 10% de recursos do cluster 1 para o cluster 0. Isso quer dizer que, por exemplo, referindo-se à simulação da Figura 5.11, quando a carga do cluster 0 atingiu 80% da sua carga limite, 10% dos recursos do cluster 1 foram deslocados para ele.

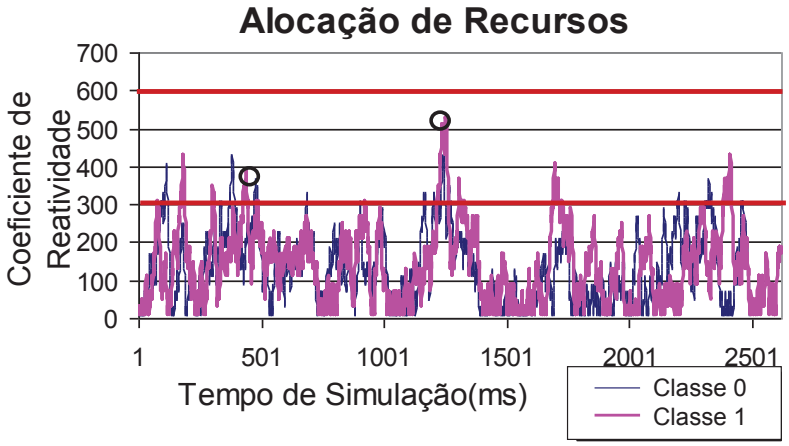

Figura 5.10: Alocação de Recursos com carga nativa da classe 0 sem a Arquitetura DARC (no-DARC2-sim).

Na Figura 5.10, é apresentada a distribuição de carga entre os dois clusters ao longo do tempo, usando *no-DARC2-sim*. Observando o comportamento dos clusters neste experimento, pode-se notar a seguinte situação particular:

- Apesar de haver distribuição de cargas, ocorre rejeição de requisições em vários momentos devido à saturação do cluster 0, pois em vários momentos ele ultrapassa o seu limite de carga (RC>300).

Na Figura 5.11, é apresentada a distribuição de carga entre os dois clusters ao longo do tempo, usando *DARC2-sim*. Observando o comportamento dos clusters neste experimento, pode-se notar a seguinte situação particular:

- Observa-se que, com a realocação dinâmica de recursos, o cluster 1, apesar de ceder recursos para o cluster 0, não saturou e manteve o seu nível de carga médio mais baixo que o da Figura 5.10. Isso pode ser comprovado através da verificação dos pontos de carga máxima do cluster 1 nas duas simulações citadas.

Na Figura 5.10, observa-se que o cluster 1 atinge um valor máximo de carga (RC=530) no instante 1254. Na Figura 5.11, o seu ponto máximo de carga (RC=350) ocorreu no instante 656 e trabalhou o restante do tempo de simulação sempre abaixo disso.

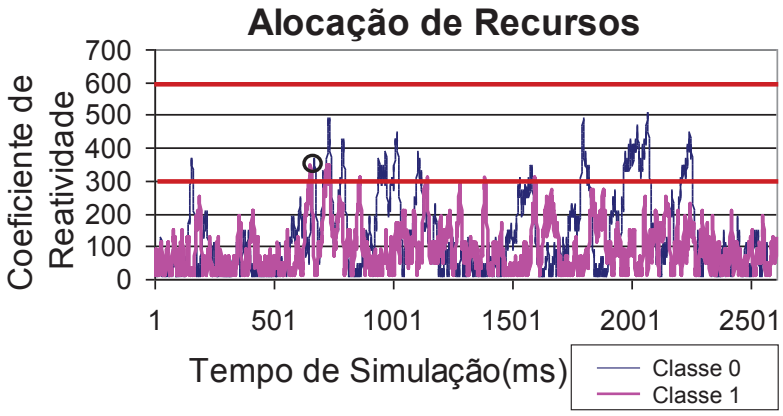

Figura 5.11: Alocação de Recursos com carga nativa da classe 0 com a Arquitetura DARC, com percentual de monitoramento de 80% (DARC2-sim).

Como nas duas situações anteriores (*no-DARC1-sim* e *DARC1-sim*), várias simulações foram feitas com o objetivo de se calcular as taxas de rejeição. Elas mostram que, no primeiro caso, a taxa média de rejeição foi de 9,8% (Figura 5.10) enquanto no segundo foi de 0% (Figura 5.11). Assim, o cluster 1 não mudou o seu modo de trabalho para saturado, embora seus recursos tenham sido deslocados para o cluster 0. O sistema conseguiu atender a todas as requisições enviadas e, consequentemente, melhorou consideravelmente a utilização dos recursos pelos usuários.

Na simulação cujos resultados são apresentados na Figura 5.12, o *agente de monitoramento* deve enviar uma informação de alerta ao *agente coordenador* quando a carga do cluster alcançar 85% do seu limite. Na Figura 5.13, isso deve ocorrer quando a carga alcança 90%. A primeira situação é chamada de *DARC3-sim* e a segunda situação é chamada de *DARC4-sim*.

É importante destacar que, comparando o comportamento das simulações representadas nas Figuras 5.11, 5.12 e 5.13, têm-se:

- Na Figura 5.11, existem 13 picos de carga referentes ao cluster 0, contra 15 do mesmo cluster na Figura 5.12, e 17 na Figura 5.13, indicando que, quando a realocação de recurso é feita com 80% do limite de carga do cluster, o

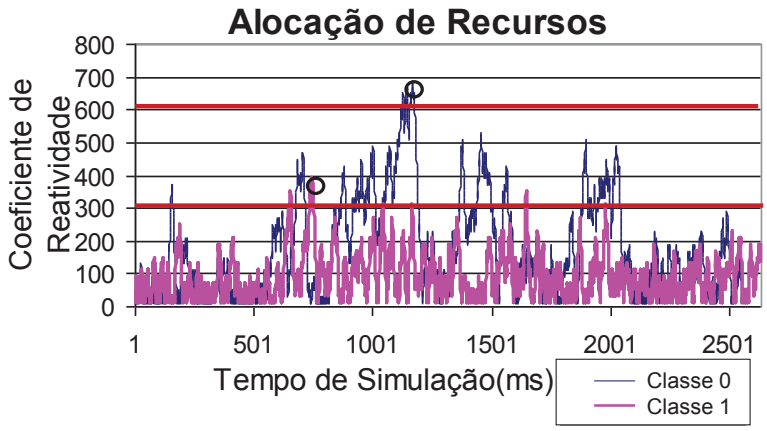

Figura 5.12: Alocação de Recursos com carga nativa da classe 0 com a Arquitetura DARC, com percentual de monitoramento de 85% (DARC3-sim).

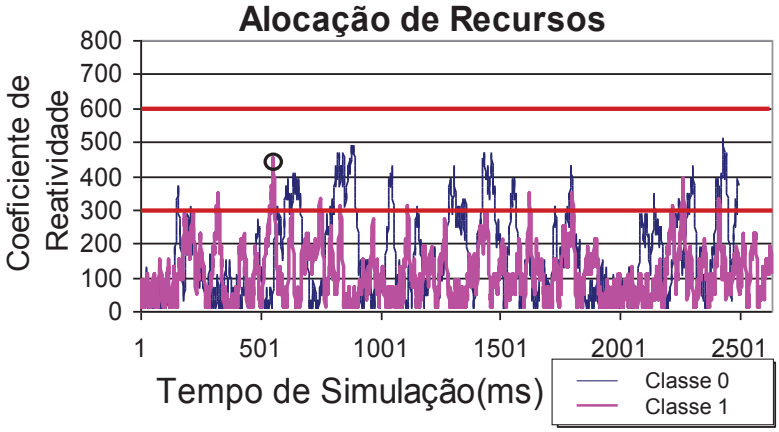

Figura 5.13: Alocação de Recursos com carga nativa da classe 0 com a Arquitetura DARC, com percentual de monitoramento de 90% (DARC4-sim.

sistema necessitará de menos reajustes do que quando as atitudes necessárias são tomadas quando o cluster monitorado está mais perto dos seus limites;

- Na Figura 5.11, o cluster 1 se manteve sempre com nível de carga baixo, a maior parte da simulação com RC<300, enquanto este nível foi maior nas outras duas simulações, e também foi onde se obteve o menor tempo de resposta ao usuário. Isto pode ser visto pela Figura 5.13, já que a maior quantidade de cargas é atendida nos instantes iniciais do experimento;

- Na simulação da Figura 5.12, os clusters se comportaram mais irregularmente do que na Figura 5.11, alcançando valores de carga próximos aos limites estabelecidos, como pode ser observado no instante 1164 ms para o cluster

0, e no instante 748 ms para o cluster 1; e

- Na Figura 5.13, o cluster 1 atingiu um valor de carga que ultrapassou o seu limite no instante 555 ms, como também em vários outros momentos da simulação seus valores foram bastante altos, maiores do que os das outras simulações (Figuras 5.11 e 5.12).

Diante dos resultados apresentados, pode-se concluir que, dentre as simulações apresentadas, a que se comportou melhor foi a que fez a realocação de recursos com monitoramento de 80% da carga do cluster 0 (Figura 5.11), esta se mantendo com maior coerência com os objetivos propostos neste trabalho. Assim, pode-se concluir que a demora na tomada de atitudes para realocação de recursos nos clusters provoca uma sobrecarga nos seus servidores e consequentemente, a rejeição de requisições.

Portanto, como foi mostrado nesta seção, em todos os experimentos realizados e apresentados, pode-se verificar que, com a utilização da arquitetura proposta, ou seja, com a adição da arquitetura DARC ao WS-DSAC, conseguiu-se realizar a realocação dinâmica dos recursos como desejado, promovendo assim uma maior equidade na utilização dos recursos (*fairness*), eliminação do número de rejeições e, consequentemente, um melhor desempenho do sistema.

5.1.4 Análise de Desempenho da Arquitetura DARC em Redes de Petri Coloridas

A análise de desempenho da arquitetura DARC foi realizada em redes de Petri coloridas aproveitando o modelo que havia sido construído para analisar a sua dinâmica de funcionamento. Esta análise consiste em utilizar a função monitores da ferramenta CPNTools. Essa análise de desempenho é calculada através de várias simulações realizadas com o modelo, variando apenas a quantidade de passos em que essa é realizada.

O tempo de resposta para este modelo em redes de Petri Coloridas foi calculado para a plataforma WS-DSAC sem DARC e WS-DSAC/DARC. O cálculo foi realizado através de várias simulações e calculado a média aritmética dos tempos obtidos. Este tempo para a plataforma WS-DSAC sem DARC foi de 2,69% e para a plataforma WS-DSAC/DARC foi de 2,74%.

O tempo de resposta do sistema WS-DSAC/DARC, calculado utilizando este modelo em redes de Petri coloridas, é um pouco maior do que o da plataforma

WS-DSAC, 1,35%. Contudo, a arquitetura DARC melhora o atendimento das requisições por parte da arquitetura DARC e, assim, a taxa de rejeição é menor, o que prova a efetividade da solução.

5.2 Modelagem em Redes de Petri Estocásticas Generalizadas

Nesta seção, é realizada uma análise formal de desempenho da arquitetura DARC seguindo as técnicas de Engenharia de Performance de Software (SPE) (SMITH; WILLIAMS, 2003). Esta análise de desempenho a partir de uma especificação de software na linguagem *Unified Modeling Language* (UML),[2] pode auxiliar uma equipe de projeto na avaliação de desempenho e na tomada de decisões de projetos complexos.

A modelagem em redes de Petri estocásticas foi utilizada para complementar a modelagem em redes de Petri coloridas, já que essa permite verificar a dinâmica do modelo bem como realizar uma análise qualitativa com possibilidade de corretude do modelo. Já o formalismo de redes de Petri estocásticas generalizada é muito útil para obter-se uma avaliação quantitativa de desempenho de sistema. Existem vários tipos de técnicas para análise de desempenho, tais como: teoria das filas e redes de Petri coloridas. Porém as redes de Petri estocásticas generalizadas são mais adequadas para análise de desempenho, já que se baseiam em cadeias de Markov e possuem a precisão necessária para a descrição de sistemas complexos. A análise de desempenho realizada neste capítulo foi realizada com redes de Petri estocásticas generalizadas e permite comparar o desempenho da plataforma WS-DSAC sem e com a utilização da arquitetura DARC. As anotações nos modelos são utilizadas para fornecer informações necessárias de desempenho (como os parâmetros de carga).

Na subseção 5.2.1, é apresentada a modelagem em UML da Plataforma WS-DSAC e da arquitetura DARC. Na subseção 5.2.2, são construídos os modelos de desempenho usando GSPN. Na subseção 5.2.6, são apresentados os resultados da avaliação de desempenho.

5.2.1 Modelagem em UML da Plataforma WS-DSAC e da Arquitetura DARC

O diagrama de sequência apresentado na Figura 5.14 modela a interação entre os agentes da arquitetura DARC. O *Agente de Monitoramento* solicita informação de

[2] http://www.uml.org

carga dos clusters ao *Agente de Comunicação*. Para obter essa informação de carga dos clusters da plataforma de servidores Web, o *Agente de Comunicação* solicita essa informação ao *Cluster Gateway* (da plataforma WS-DSAC) usando o método *getLoadInf*, e comunica este valor ao *Agente de Monitoramento*.

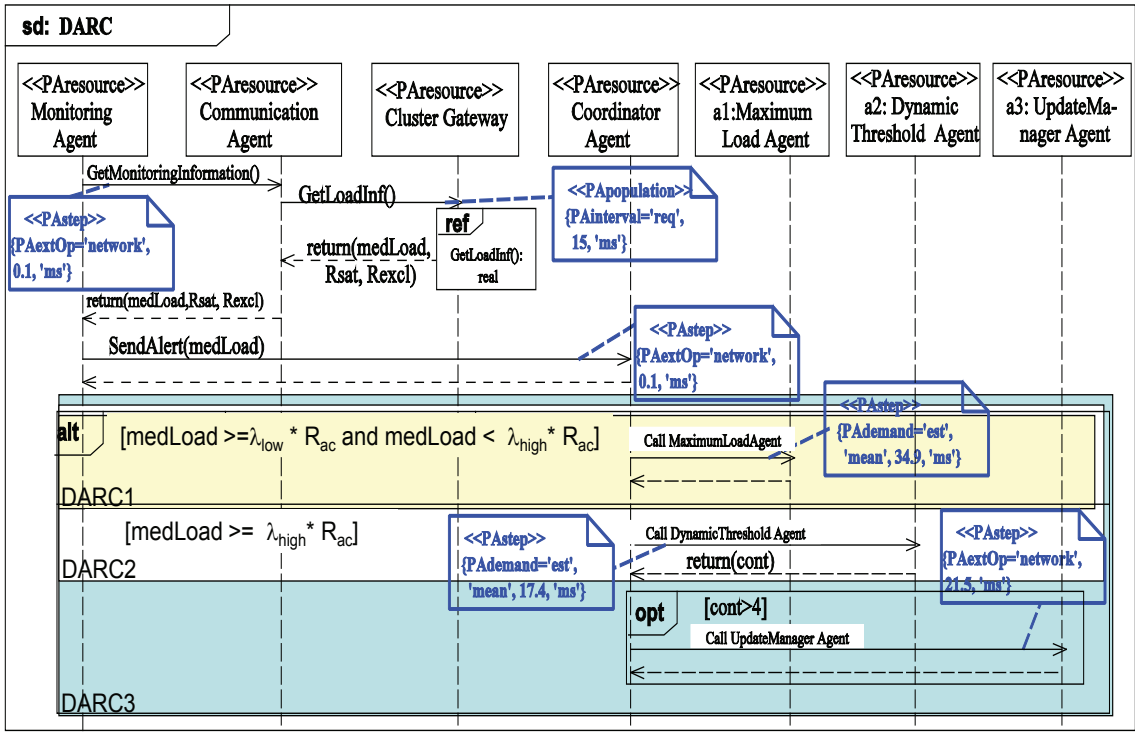

Figura 5.14: Diagrama de Sequência DARC.

Em seguida, o *Agente de Monitoramento* testa as condições a seguir para decidir como proceder:

- Se as expressões $(medLoad_k \geq \gamma_{low} {}^* R_{ack})$ e $(medLoad_k < \gamma_{high} * R_{ack})$ são verdadeiras, então o *Agente de Execução a1:MaximumLoad Agent* executará. Este agente aloca o servidor menos carregado para o cluster que está saturando.
- Caso contrário, se $medLoad_k \geq \gamma_{high} * R_{ack}$ é verdade, então o *Agente de Execução a2:DynamicThreshold Agent* executará. Este agente tem como função atualizar os thresholds do cluster antes que este sature, atuando apenas no caso em que a ação do *Agente de Execução a1:MaximumLoad Agent* não seja suficiente para evitar a saturação do cluster.
- Porém, se várias atualizações (dez ou mais) de *thresholds* ocorrerem em um curto período de tempo, então o *Agente UpdateManager* executará e este

aumentará o *threshold* do cluster (C1) em δ_1 e diminuirá o *threshold* do cluster menos carregado (C2) em δ_1 (isto é, C2 alocará δ_1 de seus recursos para processar requisições para C1).

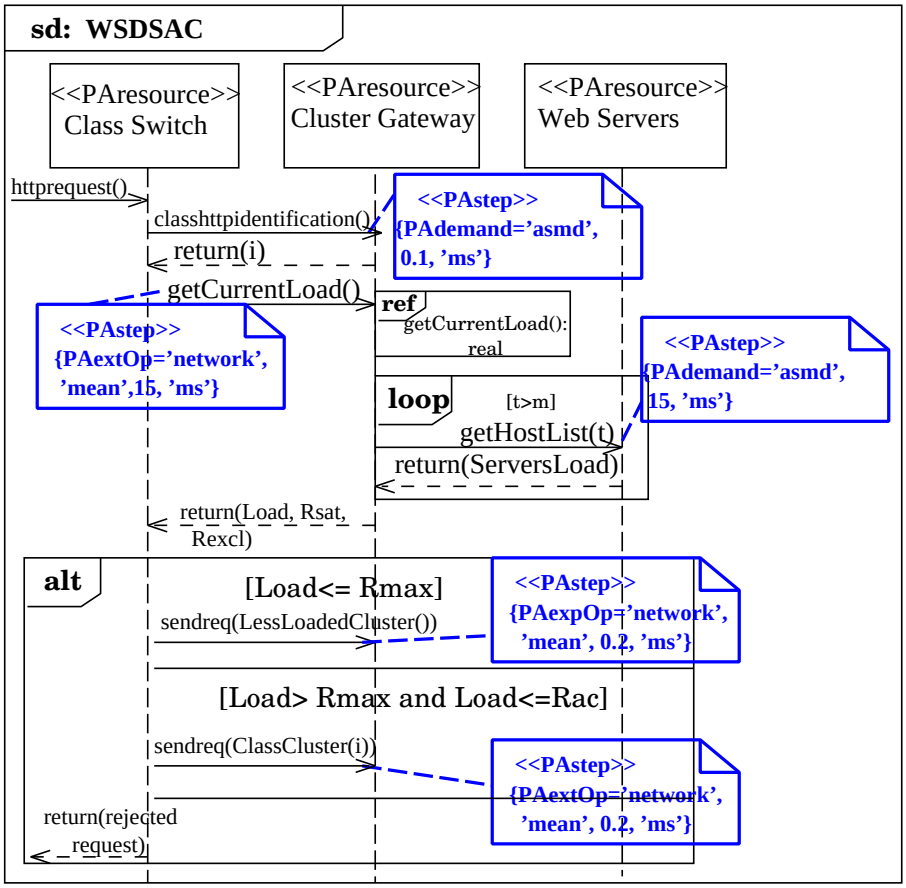

Figura 5.15: Diagrama de Sequência da Plataforma WS-DSAC.

O diagrama de sequência apresentado na Figura 5.15 modela o funcionamento da Plataforma *WS-DSAC*. Esta plataforma recebe requisições HTTP, identifica a classe de cada requisição usando o método *httpidentification*, e as envia para o *Cluster Gateway* menos carregado. O *Cluster Gateway* verifica a carga atual dos clusters, calculando a variável *medLoad*, e retorna a informação solicitada. Em seguida, o *Cluster Gateway* testa se a condição ($Load_k \leq R_{maxk}$) é verdadeira. Se sim a requisição é enviada ao cluster menos carregado. Caso contrário, se $Load_k > R_{maxk}$ e $Load_k \leq R_{ack}$ são ambas verdadeiras a requisição é enviada para o cluster da classe nativa desta. Se nenhuma destas duas condições for verdadeira então a requisição será rejeitada.

Para concluir esta seção, o *Diagrama de Implantação* (*DD*) é apresentado na Figura 5.16. Este diagrama representa como é realizada a distribuição do sistema através de nós de hardware, componentes e dependências de software e as suas devidas relações de comunicação. Na Figura 5.16 é apresentada a modelagem da estrutura física da arquitetura DARC, a localização dos agentes da arquitetura DARC e sua comunicação com os módulos da plataforma WS-DSAC. Na modelagem utilizando redes de Petri estocásticas é necessário modelar a infra-estrutura física do sistema, como processador e redes de comunicação e o relacionamento destes com o software.

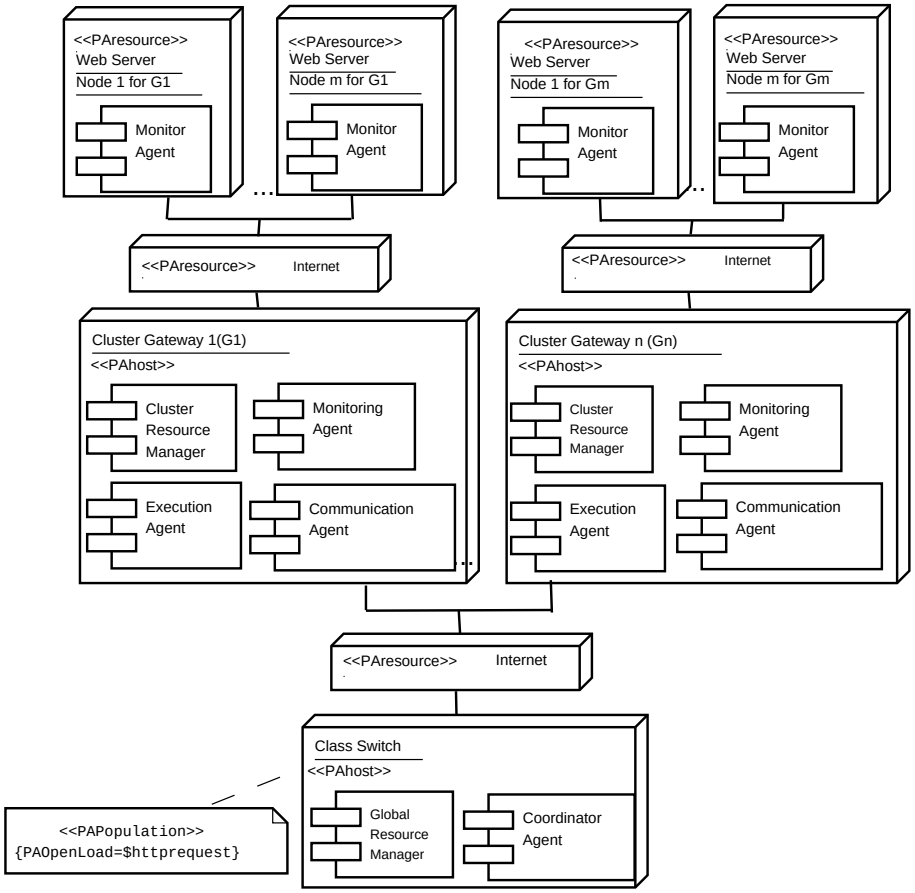

Figura 5.16: Diagrama de Implantação da Arquitetura DARC.

5.2.2 Modelos de Desempenho

Nesta seção, é apresentada a construção dos modelos de desempenho, utilizando Redes de Petri Estocásticas Generalizadas (GSPN). Inicialmente, modela-se a arquitetura DARC em Diagramas UML. As características de desempenho do sistema são anotadas nestes diagramas, vide Subseção 5.2.3. Para a conversão

dos diagramas UML anotados para GSPN utilizou-se uma ferramenta para automatização desse processo. Esta ferramenta é chamada *Corel Scenario Model* (CSM) (PETRIU; WOODSIDE, 2007). Os modelos em GSPN serão utilizados para a análise de desempenho. Veja Figura 5.17.

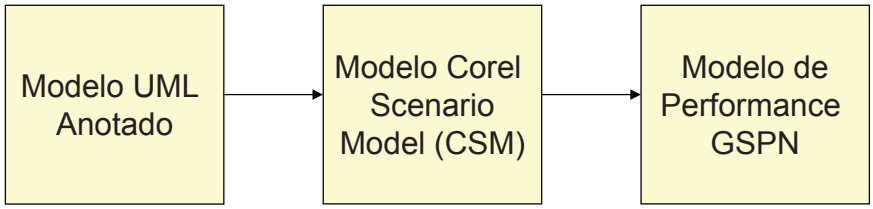

Figura 5.17: Construção do Modelo de Desempenho.

5.2.3 Modelos UML Anotados

Foi utilizado o modelo *Standard UML Profile for Schedulability, Performance and Time Specification* (UML-SPT[3]) para introduzir os valores de desempenho iniciais da arquitetura DARC e também as métricas que irão caracterizar esta análise de desempenho. O modelo UML-SPT estende o padrão UML definindo estereótipos (extensão do vocabulário UML que permite a criação de um tipo básico novo que é específico ao problema que está sendo resolvido, exemplo «PAstep») e anotações (são as informações de desempenho, exemplo PAdemand) que podem ser aplicados aos elementos do modelo. Observa-se nas Figuras 5.14, 5.15 e 5.16 que os estereótipos e anotações usados introduzem na arquitetura DARC características intrínsecas de desempenho. Estas anotações aparecem anexadas como notas nos diagramas UML, como propõe o modelo SPT. O modelo SPT baseia-se nos modelos de recursos e de desempenho, que serão a base do modelo intermediário CSM descrito na Subseção 5.2.4.

Os parâmetros de entrada da arquitetura DARC serão a carga do sistema, a duração das ações tomadas pelos agentes e os atrasos no envio de mensagens. Eles foram coletados através de testes experimentais usando a arquitetura DARC. Ações e mensagens são representadas pelo estereótipo <<PAstep>>, em que a anotação *PAdemand* especifica o tempo de execução ou espera correspondente, como uma variável aleatória distribuída exponencialmente (ver Tabela 5.2). A carga do sistema é representada pelo estereótipo <<PApopulation>>, ver Figura 5.16. A anotação

[3]http://www.uml.org

Tabela 5.2: Tempo médio de execução de operações básicas.

Operação	Realizada pela	Tempo Médio de Execução(ms)
GetMonitoringInformation	Arquitetura DARC	0.1
GetLoadInf	Arquitetura DARC	15
SendAlert	Arquitetura DARC	0.1
CallMaximumLoadAgent	Arquitetura DARC	34.9
CallDynamicThresholdAgent	Arquitetura DARC	17.4
CallUpdateThresholdAgent	Arquitetura DARC	21.5
GetCurrentLoad	Plataforma WS-DSAC	15
httprequest	Plataforma WS-DSAC	0.01
classhttpidentification	Plataforma WS-DSAC	0.1
GetHostList	Plataforma WS-DSAC	15
sendreq	Plataforma WS-DSAC	0.2

PAinterval indica que a carga utilizada neste modelo é do tipo aberta (explicada na Seção 5.2.4) e que esta muda a cada milissegundo. A anotação *PAextOp* indica o tempo que uma operação externa ao modelo influencia no tempo, por exemplo o tempo de envio de uma mensagem através da rede.

As operações apresentadas na Tabela 5.2 são as ações necessárias na reconfiguração dinâmica realizada pela arquitetura DARC bem como, no balanceamento de carga realizado pela plataforma WS-DSAC. As operações são descritas a seguir:

i. A operação *GetMonitorinInformation* solicita informações de monitoramento ao *Agente de Comunicação*;

ii. A operação *GetLoadInf* solicita informações de carga ao *Cluster Gateway*;

iii. A operação *SendAlert* envia uma informação de alerta ao *Agente Coordenador*;

iv. A operação *Call MaximumLoadAgent* realiza uma chamada ao *Agente de Execução MaximumLoad*;

v. A operação *Call DynamicThresholdAgent* realiza uma chamada ao *Agente de Execução DynamicThreshold*;

vi. A operação *Call UpdateManagerAgent* realiza uma chamada ao *Agente de Execução UpdateManager*;

vii. A operação *httprequest* realiza o recebimento de requisições HTTP pelo *Class Switch*;

viii. A operação *classhttpidentify* identifica a classe a qual a requisição HTTP pertence;

ix. A operação *getCurrentLoad* solicita informações de carga dos *Clusters*;

x. A operação *getHostList* solicita informações de carga dos *Servidores Web*; e

xi. A operação *sendreq* envia a requisição a ser atendida ao *Cluster*.

5.2.4 Diagramas Core Scenario Model

O *Core Scenario Model* (CSM) (PETRIU; WOODSIDE, 2007) é um modelo intermediário de desempenho. Os benefícios da utilização de um modelo intermediário são discutidos em (PETRIU; WOODSIDE, 2007), e estes modelos podem ser transformados em diferentes modelos formais, tais como as redes de Petri ou teoria das filas. O CSM tem como foco a descrição de cenários de desempenho UML-SPT (*Scenarios*).

É importante ressaltar que existe uma tradução automática de modelos UML-SPT anotados em modelos CSM (PETRIU; WOODSIDE, 2007). Isso significa que os modelos CSM para a arquitetura DARC, apresentados nas Figuras 5.18 e 5.20, podem ser obtidos dos modelos UML-SPT, apresentados nas Figuras 5.14 e 5.15, respectivamente. Embora exista uma tradução automática, é simples verificar algumas correspondências entre os modelos. Por exemplo, a mensagem *GetMonitoringInformation* na Figura 5.14 é convertida em uma etapa CSM na Figura 5.18. As etapas *ResAcquire* e *ResRelease* são utilizadas para alocar e desalocar os recursos necessários, respectivamente. Isso é apresentado na Figura 5.18.

5.2.5 Modelo GSPN da Arquitetura DARC

Para realizar a análise de desempenho da arquitetura DARC, utilizou-se uma extensão às redes de Petri, a classe chamada redes de Petri Estocásticas Generalizadas - *Generalized Stochastic Petri Nets* [4] (GSPN).

[4] http://www.di.unito.it/~greatspn

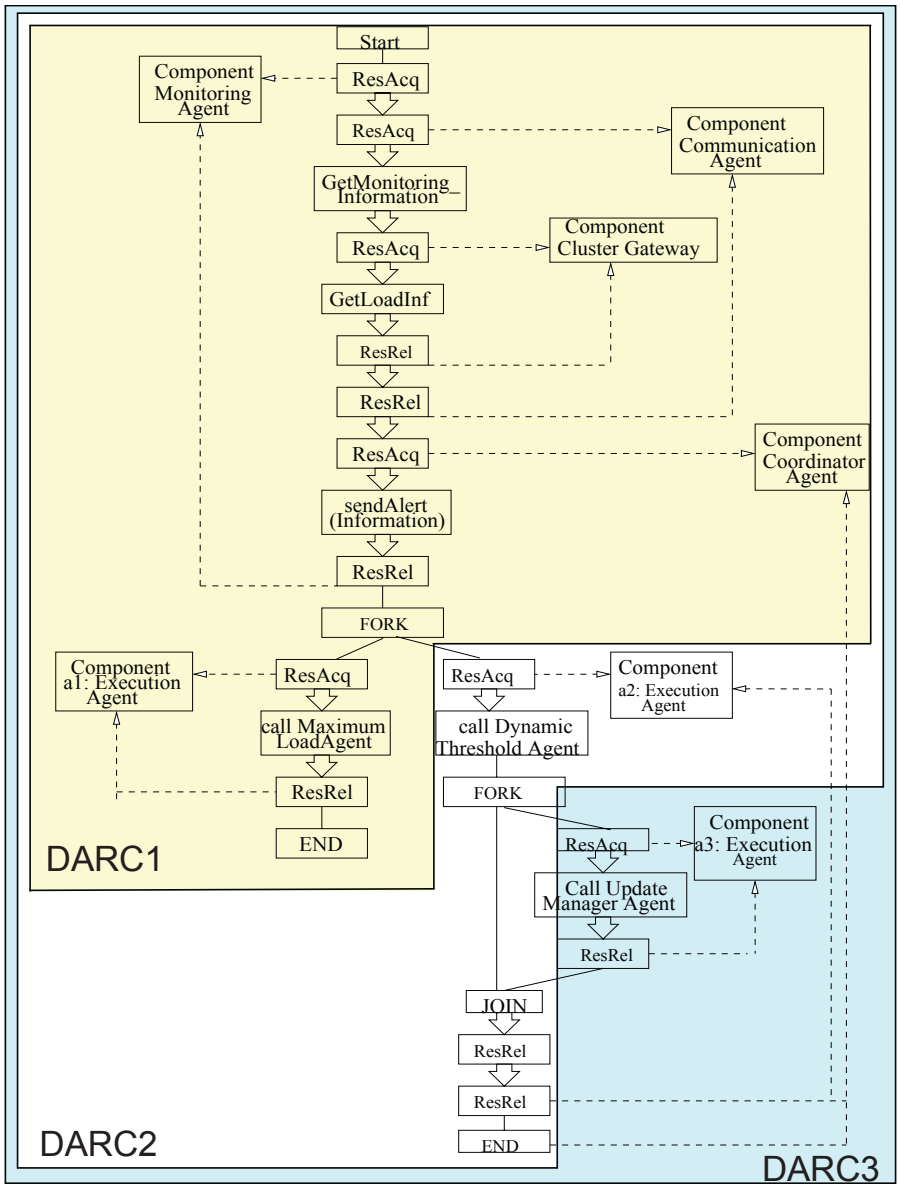

Figura 5.18: CSM para DARC-1, DARC-2 and DARC-3.

Existe uma tradução automática dos modelos CSM para GSPN[5]. As Figuras 5.19 e 5.21 descrevem as GSPNs que foram obtidas dos modelos CSMs nas Figuras 5.18 e 5.20.

Uma composição GSPN é baseada na união de lugares que representam elementos comuns nos modelos CSM, como recursos ou componentes. A tradução entre um modelo CSM para uma GSPN será explicada a seguir detalhando o passo *GetCurrentLoad* no CSM da Figura 5.18.

[5]http://webdiis.unizar.es/~jmerse/csm2pn.html

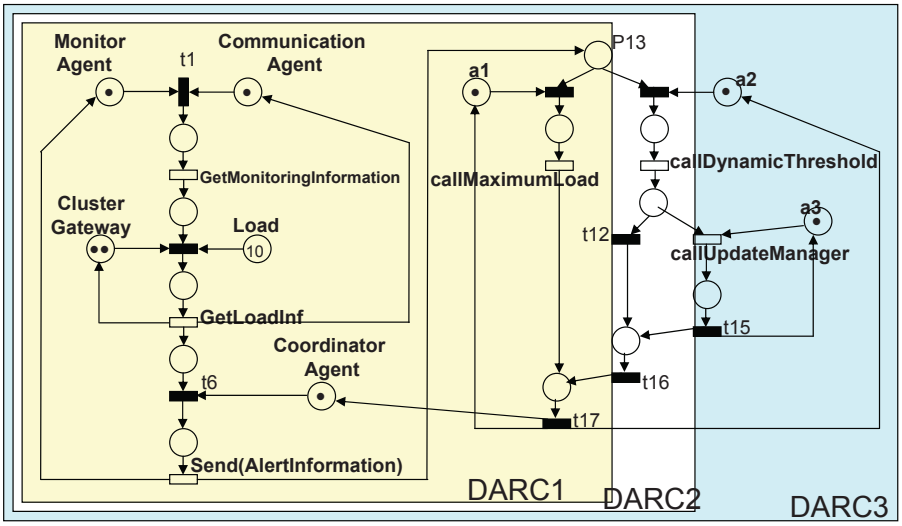

Figura 5.19: GSPN para DARC-1, DARC-2 e DARC-3.

i. O mapeamento é realizado transformando o passo em uma transição temporizada (ver Figura 5.19), sendo o atraso definido como o atributo do tempo de espera na execução desta;

ii. Anteriormente, o recurso WS-DSAC foi adquirido (ver transição t3, Figura 5.21) representando a solicitação de carga feita ao Cluster Gateway da plataforma WS-DSAC; e

iii. Os lugares representam recursos, tais como servidores ou componentes de software, e são iniciados com a quantidade de fichas especificada pela correspondente anotação *PApopulation*.

As GSPNs apresentadas nas Figuras 5.19 e 5.21 foram construídas de modo a obter um modelo de desempenho, ou seja, uma GSPN que modela o comportamento da Arquitetura DARC e da Plataforma WS-DSAC.

5.2.6 Avaliação de Desempenho

O modelo GSPN apresentado na Figura 5.19 é usado para avaliar o desempenho da arquitetura DARC. Foram realizados os seguintes experimentos:

- **DARC-1**: este experimento utiliza apenas o agente *MaximumLoad*, ou seja, realiza a realocação do servidor menos carregado para o cluster que está saturando;

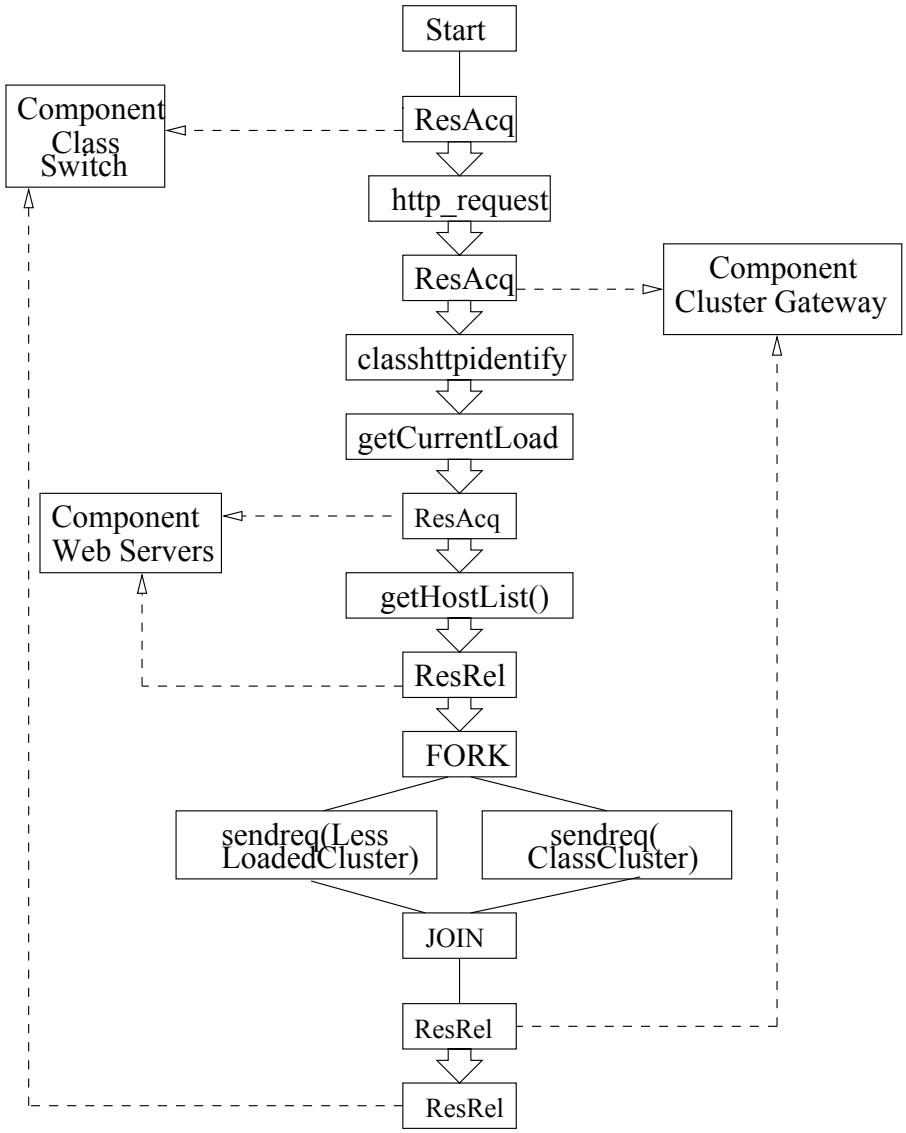

Figura 5.20: CSM da Plataforma WS-DSAC.

- **DARC-2**: este experimento utiliza o *DARC-1* com a adição do agente *DynamicThreshold* para atualização dos *thresholds* quando não é mais possível a realocação de recursos; e
- **DARC-3**: este experimento utiliza o *DARC-2* com a adição do agente *UpdateManager*, que apresenta um comportamento adaptativo. Este agente se adapta através das observações de comportamentos passados da distribuição dos recursos da plataforma de servidores Web.

Os diferentes experimentos realizados na arquitetura DARC estão identificados no modelo CSM da Figura 5.18, sendo que a estratégia DARC-3 é representada

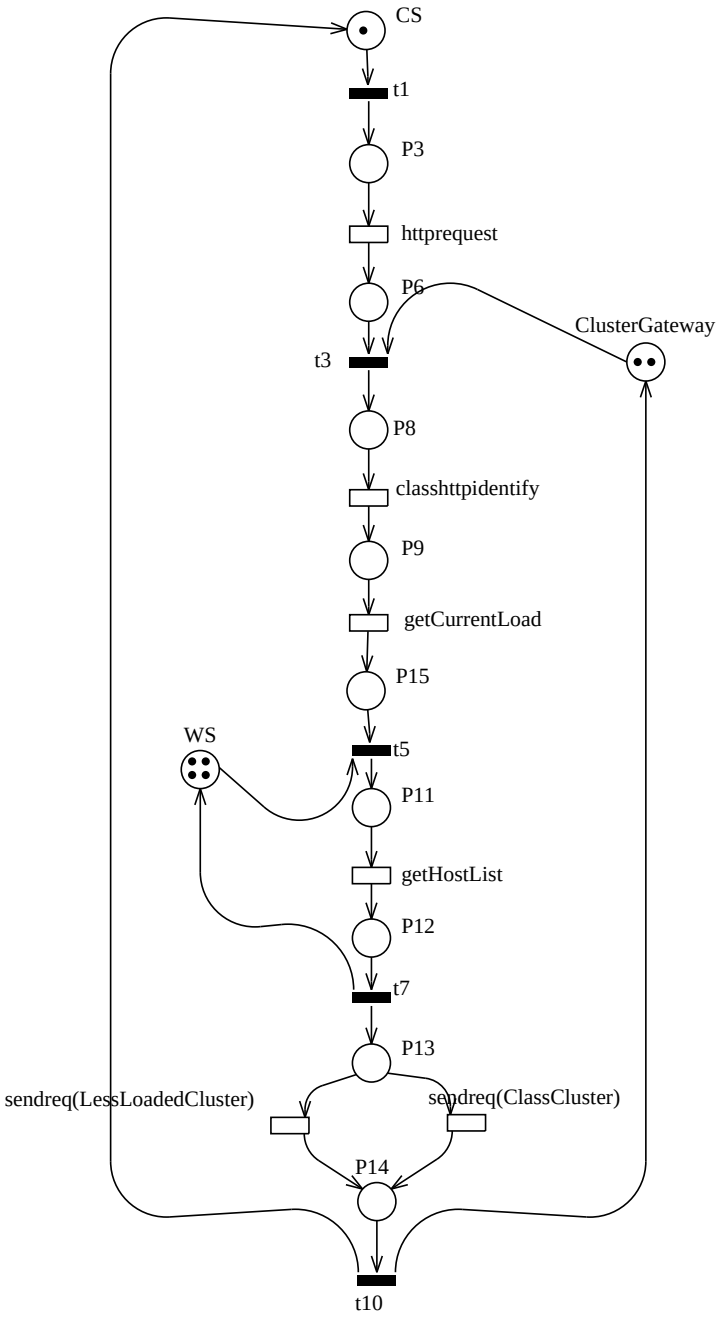

Figura 5.21: GSPN da Plataforma WS-DSAC.

por todo o modelo CSM apresentado nesta figura, ao passo que as outras duas estratégias (DARC-1, DARC-2) são apenas subconjuntos de lugares e transições de DARC-3. Portanto, a tradução desses três modelos CSM (DARC-1, DARC-2 e DARC-3) constituem as três GSPNs identificadas na Figura 5.19. Os experimentos

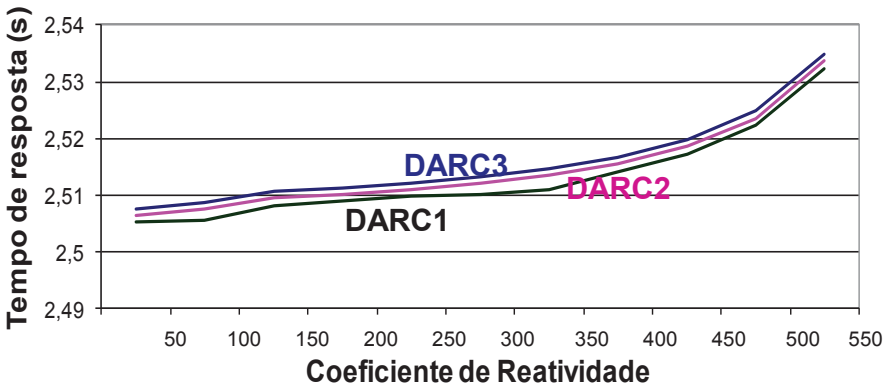

Figura 5.22: Tempo de Resposta: DARC-[1-3].

foram realizados utilizando a ferramenta GreatSPN [6] a qual se baseia na técnica de análise estrutural GSPN. Esta análise é baseada no método da "Regeneração Natural" (LAVENBERG., 1980) para prover um ponto de estimativa do número médio de fichas em cada lugar juntamente com o atraso atribuído às transições temporizadas.

Os resultados da análise do modelo GSPN para esses três experimentos são apresentados na Figura 5.22. Assim como nas simulações realizadas com os modelos de Redes de Petri Coloridas (apresentados no Capítulo 5) e na implementação (apresentada no Capítulo 6), foram enviadas a mesma quantidade de requisições HTTP nos experimentos GSPN, ou seja, a mesma carga gerada nesses testes experimentais.

Os resultados apresentados na Figura 5.22 mostram o tempo de resposta do sistema obtido da análise dos modelos GSPN. Esses tempos foram comparados com os tempos encontrados nos experimentos realizados na arquitetura e que serão apresentados no Capítulo 6.

Os resultados para os três experimentos (*DARC-1, DARC-2* e *DARC-3*) são muito semelhantes em relação aos tempos de resposta obtidos. O tempo de resposta da Plataforma WS-DSAC aumenta à medida que o coeficiente de reatividade aumenta, ou seja, à medida que a carga dos clusters aumenta. Porém, em compensação, o número de pedidos recusados diminui (Figura 5.24), o que comprova a eficácia da solução. A carga da Plataforma WS-DSAC (variação entre 50 e 550) foi parametrizada no modelo UML através da anotação *PApopulation* e esta carga é atualizada no lugar *http request* do modelo GSPN. O tempo de resposta foi calculado

[6] http://www.di.unito.it/~greatspn

na GSPN como o inverso do *throughput* dado pela transição $t17$ (Figura 5.19).

Além dos experimentos descritos e apresentados nesta seção, foi realizado um novo experimento aumentando o número de clusters e servidores Web na Plataforma WS-DSAC. A carga dos clusters foi aumentada na mesma proporção. Estes experimentos apresentam uma comparação entre os tempos de resposta dados pelos modelos de desempenho para WS-DSAC e DARC-3 em diferentes situações e estão apresentados na Figura 5.23.

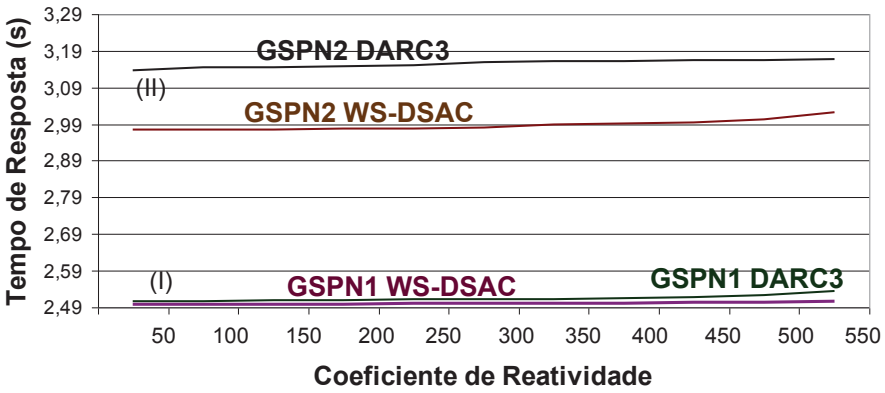

Figura 5.23: Tempo de Resposta: DARC-3 e WS-DSAC.

Pode-se observar na Figura 5.23 que o tempo de resposta da plataforma WS-DSAC usando a arquitetura DARC é um pouco maior do que o tempo de resposta da plataforma sem o uso da arquitetura DARC. Contudo, a arquitetura DARC possibilita à plataforma WS-DSAC atender mais requisições e que a taxa de rejeição dessas é menor utilizando essa arquitetura (Figura 5.24), o que comprova a eficácia da solução.

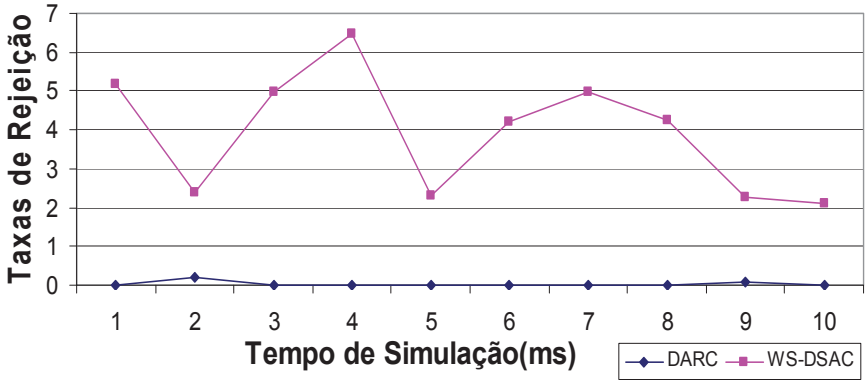

Figura 5.24: Comparação entre as Taxas de Rejeição.

Na Figura 5.24, é apresentada a taxa de rejeição dada pelo modelo de desempenho GSPN da plataforma WS-DSAC integrada à arquitetura DARC. A taxa de rejeição da plataforma WS-DSAC sem a arquitetura DARC é de 6,46%. Contudo, a taxa de rejeição desta plataforma com a arquitetura DARC é de apenas de 0,02%. Esta figura evidencia a vantagem de se utilizar uma estratégia de balanceamento de cargas com reconfiguração dinâmica sobre uma estratégia sem reconfiguração dinâmica.

5.3 Conclusões do Capítulo 5

Este capítulo descreveu a modelagem e análise da arquitetura DARC em redes de Petri: Coloridas e Estocásticas Generalizadas. Foram apresentadas: as descrições dos modelos, as simulações realizadas no modelo e a análise dos resultados obtidos.

No póximo capítulo será apresentada a implementação da Arquitetura DARC usando a linguagem de programação Java e Java RMI.

Capítulo 6

Implementação da Arquitetura de Realocação Dinâmica de Recursos

Neste capítulo, é apresentado a implementação da arquitetura DARC utilizando a linguagem de programação Java.

Na Seção 6.1, é detalhada a implementação dos agentes da arquitetura DARC. Na Seção 6.2, é detalhado o cenário de testes dos agentes da arquitetura DARC. Na Seção 6.3, são apresentadas os experimentos realizados. Na Seção 6.4, são comparados os tempos de resposta entre diversos experimentos. Na Seção 6.5, é apresentada uma proposta de integração da arquitetura DARC a plataforma G-DSAC.

6.1 Implementação dos Agentes

Os agentes adaptativos de software da arquitetura DARC foram implementados utilizando-se a linguagem de programação Java. Essa linguagem foi escolhida para facilitar a comunicação com a plataforma WS-DSAC, já que esta foi implementada nessa linguagem. No protótipo implementado estão os agentes DARC, cada um deles com uma função específica, como explicado no Capítulo 4 e repetida na Figura 6.1 para melhor detalhar os agentes da arquitetura.

Esses agentes se comunicam utilizando RMI. O agente de comunicação funciona como servidor RMI, já que atende às solicitações dos agentes de monitoramento, e funciona também como um cliente RMI, acionando métodos remotos, quando

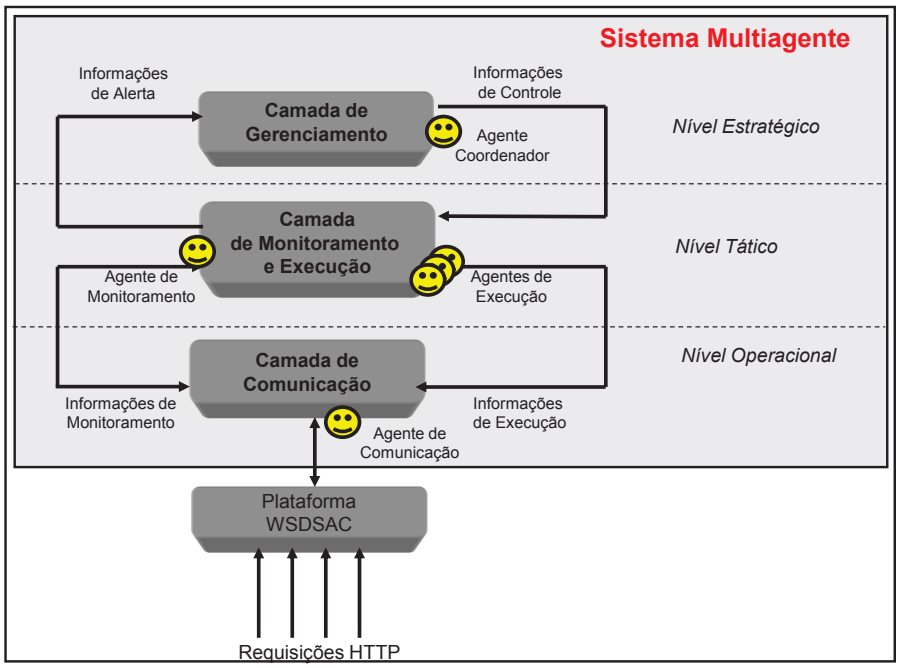

Figura 6.1: Agentes da Arquitetura DARC.

necessário. Já os agentes de monitoramento funcionam como clientes RMI porque acionam métodos remotos para solicitar informações ao agente de comunicação e enviar informações de alerta ao agente Coordenador. O agente Coordenador funciona como cliente/servidor JAVA RMI. Ele age como servidor, atendendo às invocações remotas dos agentes de monitoramento, e como cliente, acionando métodos remotos dos agentes de execução.

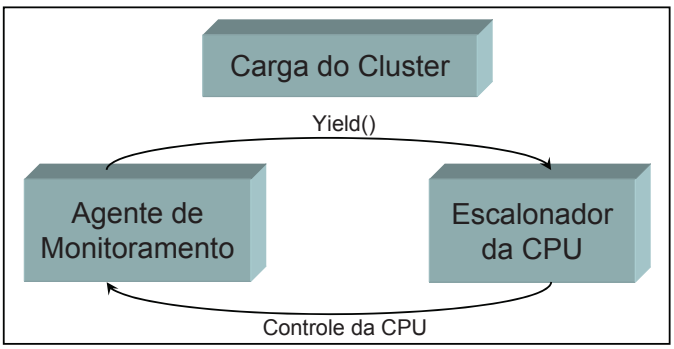

Figura 6.2: Execução da Thread de Monitoramento.

Os agentes de monitoramento são implementados como *threads* Java, ver Figura 6.2. Uma *thread* é um fluxo sequencial de controle, ou linha de execução, dentro de um processo ou programa. Um processo pode, assim, ter diversas *threads* executando concorrentemente, mas todas partilhando o mesmo espaço

de endereçamento. Como não há necessidade de trocar de contexto, as *threads* representam uma forma mais leve de processamento concorrente. O monitoramento feito pelo agente de monitoramento obtém o controle da CPU, executa por um determinado tempo e depois devolve o controle para outro processo. O tempo de execução da *thread* é aleatório, e dependendo da carga da CPU pode executar mais ou menos vezes. O método *Yield()* é utilizado para interromper a execução da *thread* de monitoramento e ceder o seu tempo de processamento para os módulos da plataforma WS-DSAC continuarem atendendo as requisições e realizando o balanceamento de carga.

Na Figura 6.3, são evidenciadas as trocas de mensagens entre os agentes.

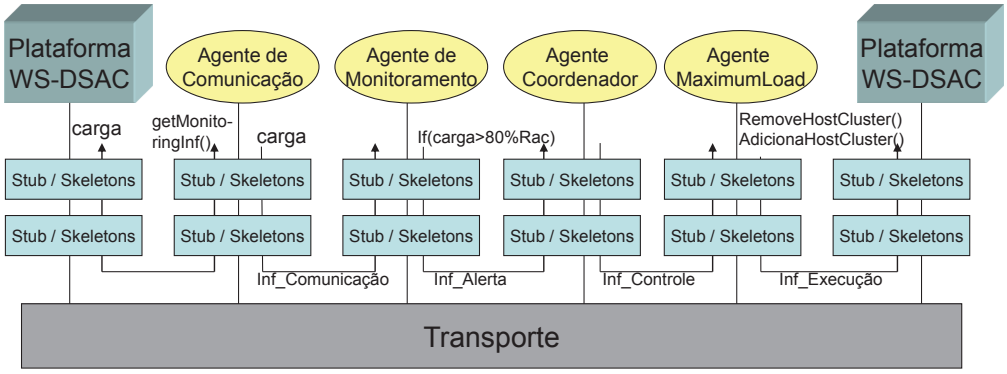

Figura 6.3: Troca de mensagens entre os agentes da arquitetura de realocação dinâmica.

O *agente de monitoramento* realiza o monitoramento da carga dos clusters através de invocações remotas ao *agente de comunicação*. Quando um parâmetro monitorado pelo *agente de monitoramento* ultrapassa um dos limites pré-estabelecidos, é enviada uma mensagem de alerta ao *agente coordenador*. O *agente coordenador*, logo após analisar a mensagem, enviará uma mensagem para o *agente de execução maximumLoad*. Ao receber a informação de controle do agente coordenador, o *agente de execução maximumLoad* aciona os métodos necessários para descobrir qual é o servidor menos carregado. Se este servidor não pertence à classe que está próxima de atingir sua carga máxima, ele será removido do cluster a que pertence e passará para a outra classe mais carregada.

6.2 Cenário de Testes da Arquitetura DARC

Na Figura 6.4 é apresentado o modelo em redes de Petri coloridas da plataforma

WS-DSAC integrado à arquitetura DARC (WS-DSAC/DARC). Essa Figura é formada pelos componentes:

i. Cliente envia requisições HTTP: representa as requisições HTTP enviadas pelos clientes;

ii. Plataforma WS-DSAC: contém os elementos básicos da plataforma de servidores Web que realizam o balanceamento de cargas;

iii. Arquitetura DARC: contém os agentes da arquitetura que realizam a reconfiguração dinâmica dos recursos na plataforma de servidores Web.

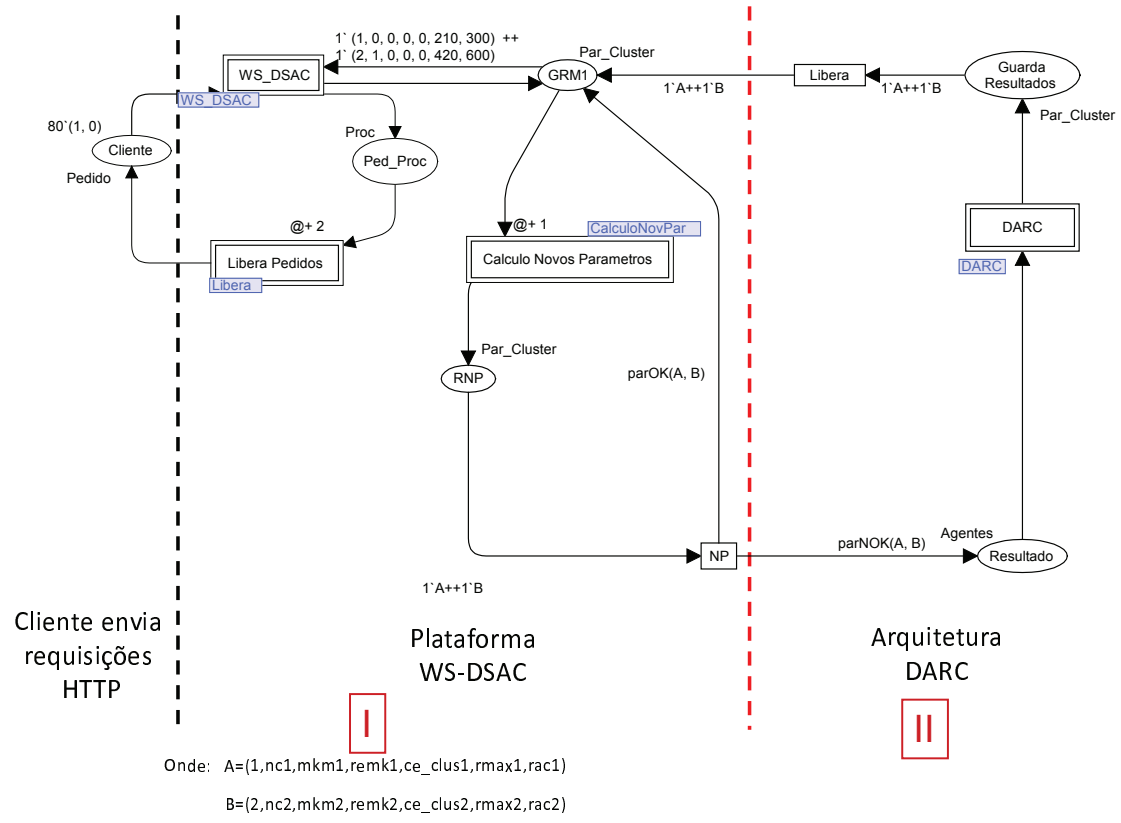

Figura 6.4: Modelo em redes de Petri coloridas.

Toda a implementação realizada nesse capítulo foi baseada no modelo em redes de Petri coloridas apresentado na Figura 6.4 e estes podem ser visualizados na Figura 6.5.

Para avaliar a arquitetura proposta, foram realizados vários experimentos em um ambiente real de testes [1]. Como é mostrado na Figura 6.5, um cenário heterogêneo (com sistemas operacionais e configurações de hardware diferentes) e dois clusters (*Cluster 0* e *Cluster 1*) foram considerados para a avaliação dos experimentos:

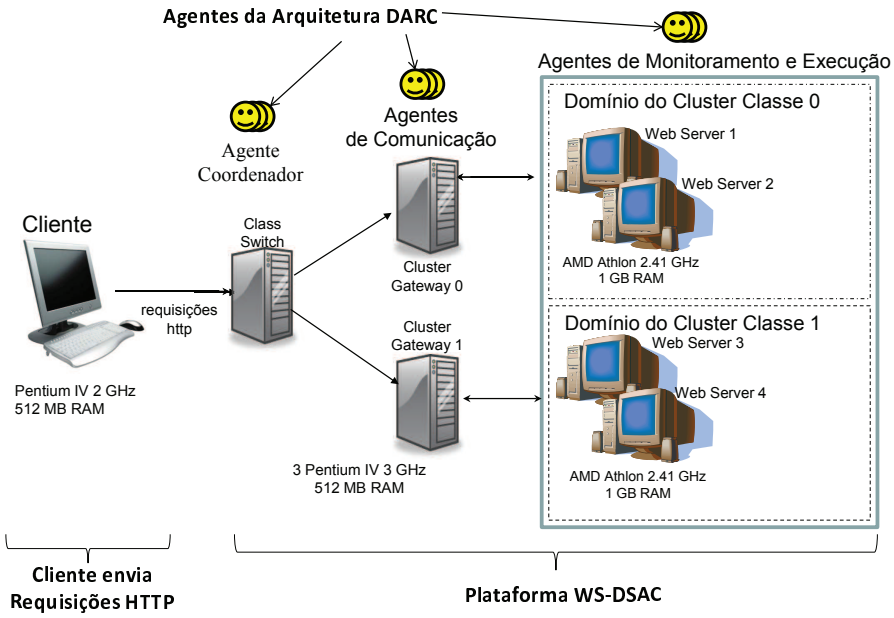

Figura 6.5: Hardware utilizado nos experimentos.

- Os agentes Coordenador e de Comunicação foram instalados na máquina que contém o Class Switch e Cluster Gateways;
- Os agentes de Monitoramento e de Execução foram instalados nos servidores Web;
- Como servidores Web 1, 2, 3 e 4 foram utilizados PCs genéricos, com um Processador AMD Athlon 64 3800, CPU 2.41 GHz, 1 GB RAM, e uma placa de rede full duplex Ethernet 100 Mb. Nestes nós, o *Apache Tomcat 5 Servlet/JSP Container* [2] está instalado;
- Como Class Switch e Cluster Gateway foram utilizados PCs genéricos com processador Intel Pentium IV, CPU 3.06 GHz, 512 MB RAM, e uma placa de rede 100 Mb full duplex Ethernet; e

[1] Cada experimento é repetido várias vezes e a média aritmética desses experimentos é calculada.
[2] `(http://tomcat.apache.org)`

Tabela 6.1: Parâmetros limites dos Clusters.

	Classe 0	Classe 1
R_{ac}	300	600
R_{max}	210	420

- Como nó-cliente foi utilizado um PC genérico, com processador Intel Pentium IV, CPU 2 GHz, 512 MB RAM, e uma placa de rede full duplex Ethernet 100 Mb. Foi utilizado *Webserver Stress Tools 7 Professional Edition* [3] para emular o envio de requisições HTTP, sendo que o Class Switch recebe uma requisição a cada milissegundo.

O administrador da plataforma associa um valor máximo de carga que pode ser alcançado pelos clusters pertencentes a cada classe de serviço através do módulo de definição de Classe.

Nestes experimentos, dois domínios diferentes de classes de pedidos foram definidos na plataforma (classes 0 e 1). Essas classes estão associadas aos clusters 0 e 1, respectivamente. Os parâmetros limites para cada cluster são apresentados na Tabela 6.1 (isto é, o Cluster 1 tem mais recursos alocados porque as requisições da Classe 1 têm prioridade maior). Estes valores iniciais foram determinados através de vários experimentos realizados dentro do contexto da plataforma WS-DSAC: os mesmos apresentaram bons resultados em vários experimentos realizados. Os demais valores utilizados levavam a uma saturação mais rápida e um aumento na taxa de rejeição de requisições.

6.3 Experimentos e Análise dos Resultados da Arquitetura Implementada

No primeiro experimento realizado, foi avaliada a plataforma WS-DSAC sem reconfiguração dinâmica, ou seja, sem a arquitetura DARC. Este experimento é chamado *no-DARC*. Na Figura 6.6, é mostrada a distribuição de carga entre os dois clusters ao longo do tempo, para um cenário com uma carga inicial baixa que aumenta consideravelmente nos momentos finais da simulação. Embora o Cluster 0 exceda o valor de R_{ac} nos instantes 120 ms e 270 ms, nenhuma requisição foi

[3] (`http://www.paessler.com/webstress`)

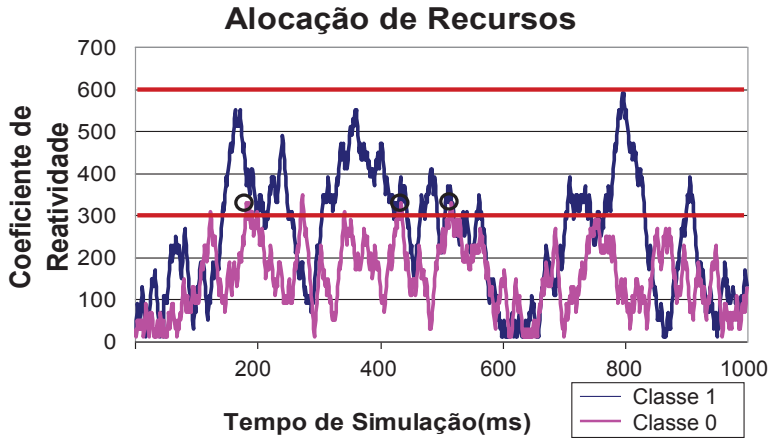

Figura 6.6: WS-DSAC sem DARC.

rejeitada porque o Cluster 1 se encontrava no modo compartilhado e atendeu às requisições nativas de sua classe e da classe 0. Contudo, nos instantes de tempo 182 ms, 430 ms, e 514 ms, o Cluster 0 excedeu seu limite R_{ac}, o que levou à rejeição das requisições da classe atribuída ao cluster 0, já que o Cluster 1 estava no modo exclusivo. O Cluster 1 chegou muito próximo a exceder o valor de R_{ac} no final da simulação, alcançando o valor de 598 ms. Nesse mesmo instante da simulação, o Cluster 0 estava no modo exclusivo.

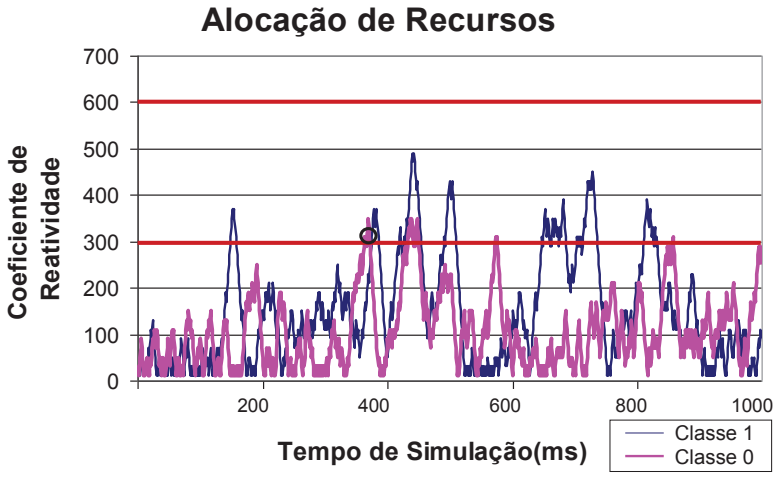

Figura 6.7: WS-DSAC com DARC-1.

No segundo experimento realizado, foram avaliados os benefícios da reconfiguração dinâmica com a arquitetura DARC, com a adição do *Agente MaximumLoad*. Este experimento é chamado *DARC-1*. Na Figura 6.7, o Cluster 0

excede o valor de R_{ac} no instante de tempo 364 ms; como o cluster 1 está no modo exclusivo, nesse momento, então as requisições nativas da classe atribuída ao cluster 0 serão rejeitadas. No entanto, em relação ao experimento anterior, em que não é utilizado DARC (cujos resultados são mostrados na Figura 6.6), o percentual de requisições rejeitadas diminui.

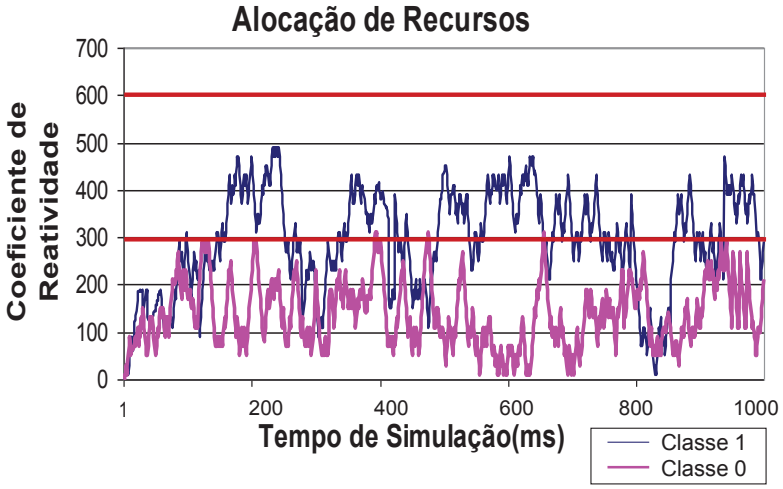

Figura 6.8: WS-DSAC com DARC-2.

No terceiro experimento realizado, foram avaliados os benefícios do experimento DARC-1 com a adição do *agente DynamicThreshold* para melhorar a utilização dos recursos. Este experimento é chamado *DARC-2*. Os resultados são apresentados na Figura 6.8. Nos instantes de tempo 390 ms e 930 ms, o Cluster 0 excedeu o limite e não foi possível alocar qualquer *host* do cluster 1 (que estava em modo exclusivo) para o Cluster 0. Para resolver este problema, o *agente DynamicThreshold* atualizou os limites do cluster algumas vezes (3-5 vezes nas diferentes repetições dos experimentos).

Finalmente, foram realizados experimentos com a melhor estratégia de reconfiguração dinâmica, em que o *agente UpdateManager* (que possibilita uma adaptação da arquitetura baseada em comportamentos passados) é adicionado ao *DARC-2*. Este experimento é chamado *DARC-3*. Os resultados são mostrados na Figura 6.9. Neste experimento, não houve pico de saturação para os clusters 0 e 1. Com a utilização da arquitetura DARC foi possível resolver os principais problemas da arquitetura WS-DSAC apresentados no atendimento das requisições. Portanto, com a adição do *agente UpdateManager* foi possível reduzir os picos de saturação do

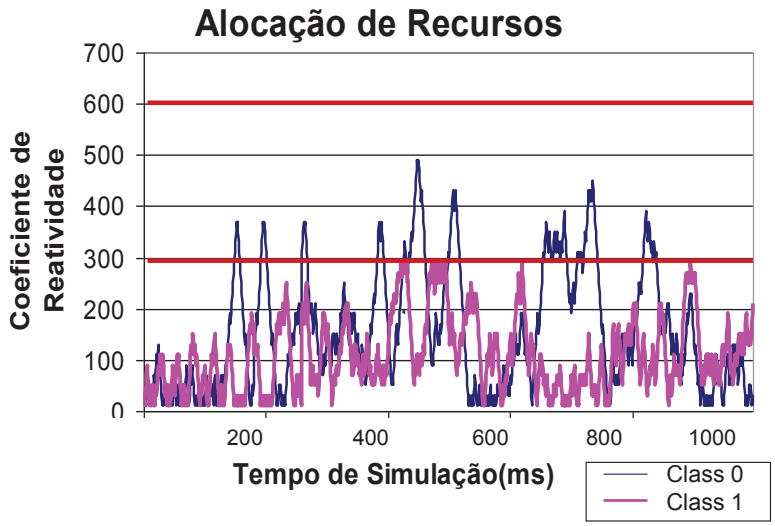

Figura 6.9: WS-DSAC com DARC-3.

sistema e como consequência o número de requisições rejeitadas.

Uma comparação final entre as estratégias analisadas (*no-DARC*, *DARC-1*, *DARC-2*, e *DARC-3*) é apresentada nas Figuras 6.10 e 6.11.

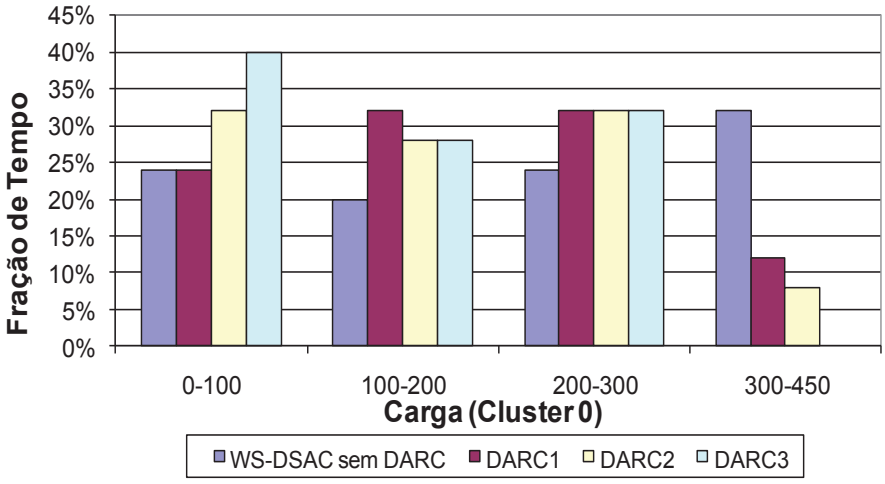

Figura 6.10: Comparação de Estratégias: Variação de Carga no Cluster 0.

Na Figura 6.10, pode-se ver a fração de tempo em que o Cluster 0 está com uma determinada carga (para diferentes intervalos de coeficientes de reatividade: 0-100, 100-200, etc). Observa-se que com a utilização da estratégia DARC-3 a carga do cluster é a mais baixa na maioria das vezes, já que esta estratégia distribui os

recursos disponíveis de forma mais eficiente. Assim, com a estratégia DARC-3 a carga do Cluster 0 está:

- entre 0-100 durante 40% do tempo total do experimento;
- entre 100-200 durante 28% do tempo total do experimento;
- entre 200-300 durante 32% do tempo total do experimento.

Na estratégia DARC-3, a carga do cluster 0 não alcançou limites acima de 300.

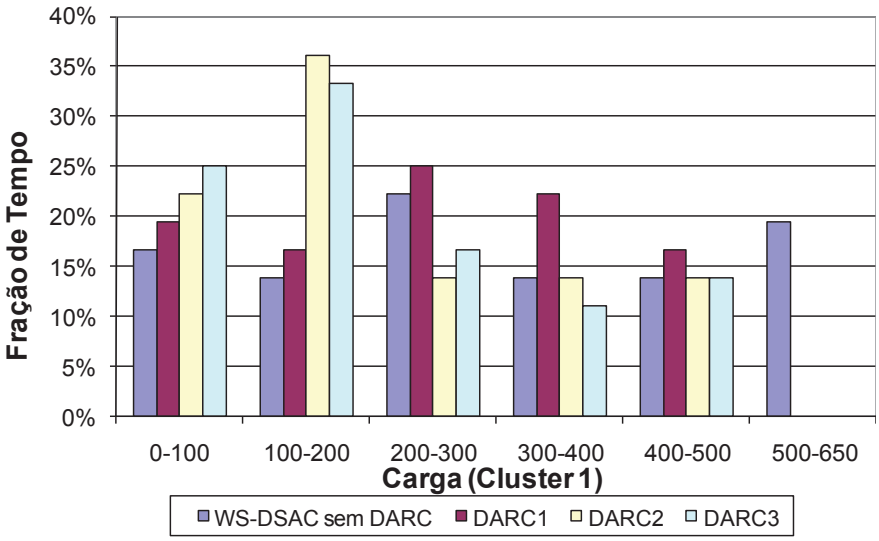

Figura 6.11: Comparação de Estratégias: Variação de Carga no Cluster 1.

Já na Figura 6.11, pode-se ver a fração de tempo em que o Cluster 1 está com uma determinada carga (também para diferentes intervalos de coeficientes de reatividade: 0-100, 100-200, etc). Observa-se também que com a utilização da estratégia DARC-3 a carga do cluster 1 é a mais baixa na maioria das vezes. Isso é possível já que esta estratégia distribui os recursos disponíveis de forma mais eficiente. Assim, com a estratégia DARC-3, a carga do Cluster 1 está:

- entre 0-100 durante 25% do tempo total do experimento;
- entre 100-200 durante 33% do tempo total do experimento;
- entre 200-300 durante 17% do tempo total do experimento;

- entre 300-400 durante 12% do tempo total do experimento;
- entre 400-500 durante 13% do tempo total do experimento;
- entre 500-650 durante 0% do tempo total do experimento.

Apenas a estratégia no-DARC apresentou carga nos limites entre 500-650 durante 19% do tempo total do experimento.

6.4 Comparação entre os Tempos de Resposta

Nesta seção, serão discutidos os tempos de resposta dos experimentos apresentados neste capítulo e os experimentos realizados com os modelos GSPN (Capítulo 5, Seção 5.2).

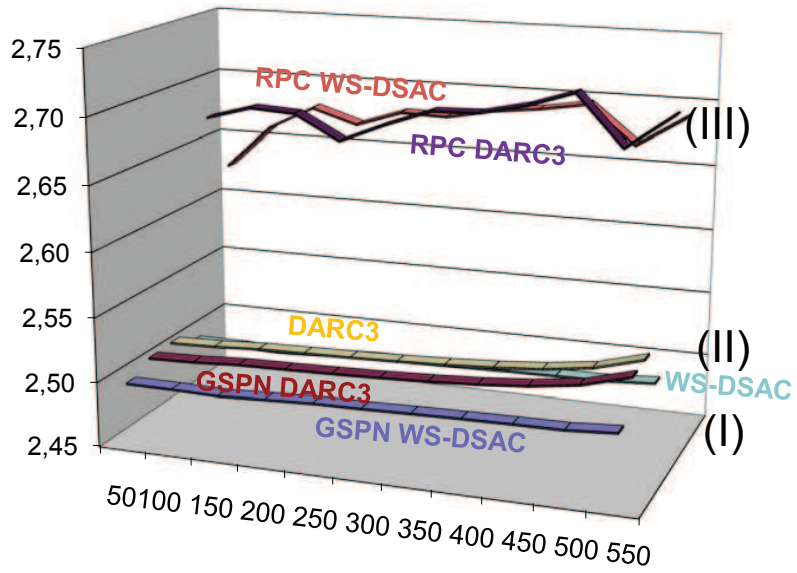

Figura 6.12: Tempo de Resposta: DARC-3 e WS-DSAC.

i. **Experimento (I)**: apresenta resultados obtidos dos modelos GSPN;

ii. **Experimento (II)**: apresenta resultados obtidos no ambiente real; e

iii. **Experimento (III)**: apresenta resultados obtidos dos modelos RPC.

Estes experimentos apresentam uma comparação entre os tempos de resposta dados pelos modelos de desempenho para WS-DSAC e DARC-3 em diferentes situações e estão apresentado na Figura 6.12.

Note que os tempos de resposta do protótipo implementado são praticamente os mesmos dos tempos obtidos nos testes dos modelos GSPN mostrando que o sistema implementado é fiel ao modelo. Existe uma diferença de aproximadamente 6,5% entre os tempos de resposta obtidos com o modelo GSPN e o modelo em redes de Petri coloridas.

6.5 Proposta de Integração da Arquitetura DARC com outras plataformas

A Arquitetura DARC é capaz de realizar a reconfiguração dinâmica em outras plataformas de servidores Web. Como exemplo, foi utilizada a plataforma G-DSAC *Grid DiffServ Admission Control* (CEDRO D. M. ; SOUZA, 2009) (ver Figura 6.13).

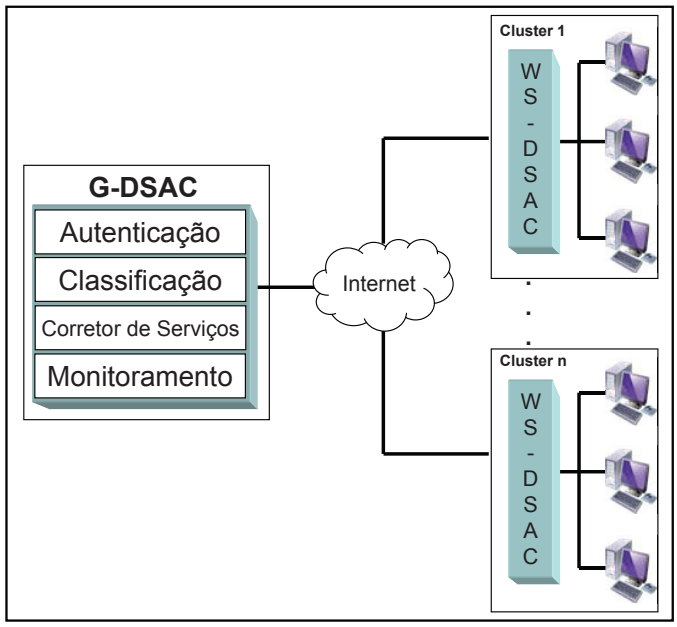

Figura 6.13: Plataforma G-DSAC.

Esta plataforma inclui paradigmas de computação distribuída, notadamente as arquiteturas em grade orientadas a serviços. Esta plataforma compreende a concepção de uma solução voltada para grades computacionais que seja capaz de garantir os SLAs (*Service Level Agreements*) estabelecidos com os usuários utilizando de forma otimizada os recursos de processamento disponibilizados na grade.

Para que a reconfiguração dinâmica, proposta nessa tese, possa ser realizada na plataforma G-DSAC é necessário que os agentes de comunicação da Arquitetura DARC possam obter informações de carga dos servidores Web, e após o

processamento de realocação dinâmica retornem a nova configuração da plataforma (ver Figura 6.14).

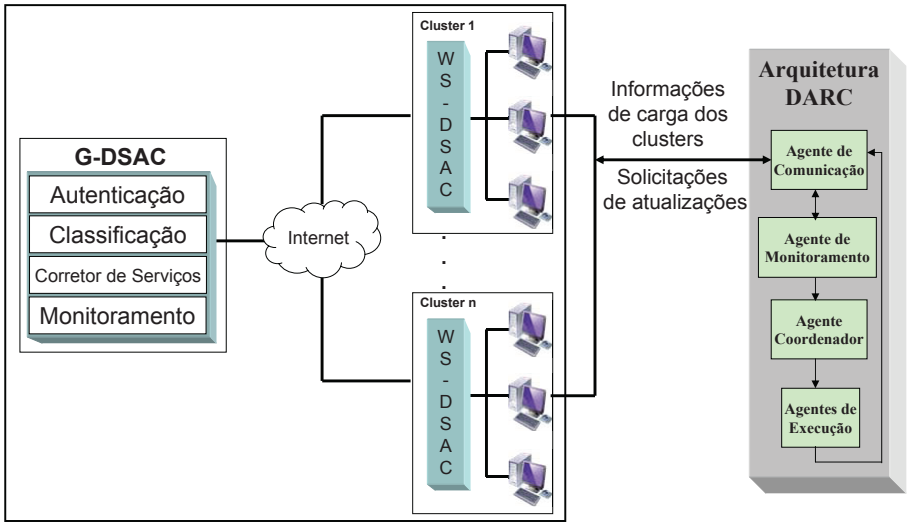

Figura 6.14: Comunicação da Plataforma G-DSAC com a Arquitetura DARC.

6.6 Conclusões do Capítulo 6

Neste capítulo, foram apresentados: a implementação da arquitetura DARC utilizando a linguagem Java, o cenário de testes da arquitetura DARC, os experimentos realizados e a análise de seus resultados,bem como uma comparação entre os tempos de resposta de diversos experimentos. No final do capítulo, apresentou-se uma proposta de integração desta arquitetura a uma outra plataforma de servidores Web.

No próximo capítulo serão apresentadas as Conclusões obtidas neste trabalho.

Capítulo 7

Conclusão

Nesta tese, foram apresentadas a concepção, a especificação, a modelagem e a implementação da arquitetura DARC (*Dynamic Architecture for Reconfiguration of Web servers Clusters*). Esta arquitetura realiza a reconfiguração dinâmica de clusters de servidores Web, utilizando sistemas multiagentes. O objetivo desta arquitetura é auxiliar a tarefa do administrador de clusters através da reconfiguração dinâmica dos recursos de um cluster.

Para validação e testes dessa arquitetura, foi utilizada como estudo de caso a plataforma de servidores Web WS-DSAC. Esta plataforma foi escolhida por oferecer benefícios de balanceamento de carga em um ambiente distribuído, e também oferecer diferentes níveis de QoS baseando-se na diferenciação de serviços.

As vantagens de utilizar a arquitetura DARC integrada a uma plataforma de servidores Web são:

- A nova arquitetura é composta por dois níveis: Nível de Balanceamento de Carga e Nível de Reconfiguração Dinâmica. A reconfiguração dinâmica de recursos é realizada em um nível acima do nível em que é realizado o balanceamento de carga. O trabalho em um nível acima permite que agentes observem o comportamento da arquitetura, se adaptem as novas necessidades de recursos e depois executem ações sobre a plataforma de servidores Web (Figura 7.1);
- Sem a reconfiguração dinâmica, se os clusters estão em modos de saturação e exclusivos, todas as requisições direcionadas aos clusters que se encontram

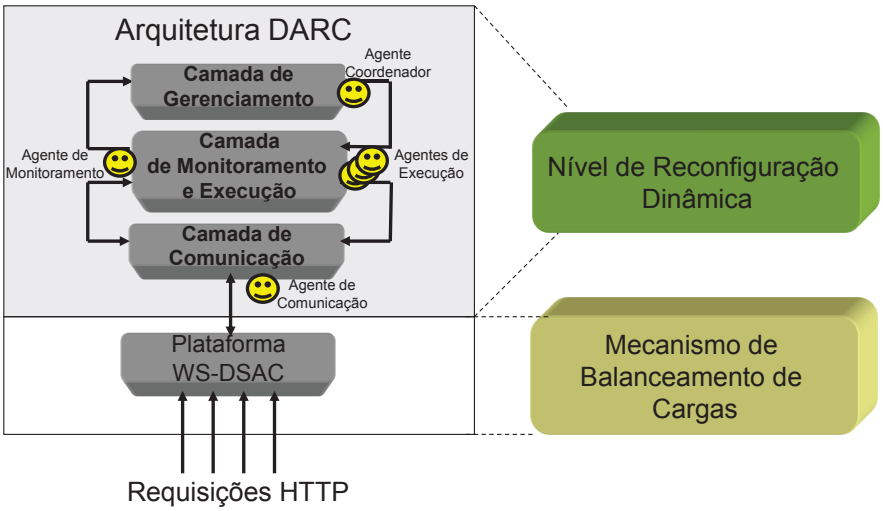

Figura 7.1: Plataforma WS-DSAC integrado a Arquitetura DARC.

em modo de saturação serão rejeitadas. Com a reconfiguração dinâmica, será possível ainda utilizar parte dos recursos dos clusters que estão em modo exclusivo para aceitar requisições daqueles que se encontram em modo saturado;

- Realocação dinâmica de servidores. Isso foi assim implementado após a realização de testes utilizando-se realocação de servidores e de requisições. Este trabalho foi realizado numa dissertação de mestrado (ainda em andamento) no Departamento de Teleinformática em que chegou-se a conclusão de que, à medida que aumentava a quantidade de requisições a serem realocadas, o tempo de resposta da plataforma aumentava. Em contrapartida, sendo feita a realocação de servidores, o tempo de resposta permanece praticamente o mesmo e a taxa de rejeição das requisições diminui consideravelmente;

- Modificação dos limites máximos de cada modo de trabalho da plataforma. Isso foi implementado utilizando-se agentes adaptativos que, através da observação dos comportamentos passados da plataforma, se adptam às novas necessidades de recursos bem como detectam a necessidade de atualizações dos limites de carga do cluster, evitando saturação e rejeição de requisições; e

- Possibilidade de integração com outras plataformas de servidores Web.

Os principais objetivos buscados e alcançados nesta tese foram:

- Realizar a tarefa de reconfiguração dinâmica de recursos através das interações de agentes reativos ou cognitivos com a plataforma de servidores Web:

  - A arquitetura DARC é composta por um sistema multiagente capaz de adaptar um cluster às constantes mudanças em um ambiente distribuído (e.x., aumentar servidores para uma classe que esteja necessitando de recursos). Os agentes da arquitetura são capazes de reconfigurar os recursos da plataforma interagindo com a mesma. A tarefa de reconfiguração dinâmica foi alcançada através das interações entre os agentes da Arquitetura DARC e a plataforma de servidores Web e foi possível utilizar parte dos recursos dos clusters que estão em modo exclusivo para aceitar requisições daqueles que se encontram em modo saturado;

- Melhorar a eficiência e a utilização dos recursos disponíveis no cluster através das ações que os agentes exercerão sobre o mecanismo de gestão da QoS:

  - Os resultados obtidos após as modelagens e a implementação da Arquitetura DARC mostraram uma melhoria do desempenho da plataforma WS-DSAC quando integrada a arquitetura em questão;

- Tomar decisões dinâmicas a respeito da taxa de utilização dos recursos do sistema baseado em informações de carga dos clusters da plataforma de servidores Web que são colhidas pelos agentes da arquitetura DARC:

  - A arquitetura DARC realiza a atualização do *threshold* do cluster, através do agente *UpdateManager*, baseando-se em situações passadas e estas atualizações se adaptam às variações de cargas dos clusters bem como das atualizações anteriores realizadas pelo agente *DynamicThreshold*;

- Minimizar o trabalho do administrador do cluster e reduzir a possibilidade de erros em períodos de picos de carga da Internet:

  - A arquitetura DARC minimizou a tarefa do administrador de clusters já que reduziu a possibilidade de erros na distribuição dos seus recursos em períodos de picos de carga já que um cluster, por ser um ambiente distribuído, possui características inconstantes e imprevisibilidade da carga de trabalho dos servidores Web; e

- Validar o sistema através de modelagem, simulação e experimentos e comparar a estratégia de balanceamento de cargas sem reconfiguração dinâmica com a estratégia com reconfiguração dinâmica:
  - Todas as simulações realizadas nesta tese compararam as duas estratégias citadas e provaram que a estratégia de balanceamento de cargas com reconfiguração dinâmica diminuiu a taxa de rejeição sem impactar no desempenho.

Nesta tese, foram realizadas a modelagem do sistema WS-DSAC/DARC em redes de Petri Coloridas (usando a ferramente CPNTools) e modelagem em redes de Petri Estocásticas Generalizadas / UML (usando a ferramenta GreatSPN). Estas modelagens foram realizadas porque elas abragem aspectos diferentes. O modelo em redes de Petri colorida possibilita analisar a estrutura e a dinâmica do sistema, bem como é de fácil visualização e entendimento. Já o modelo em redes de Petri estocástica explicita os tempos de comunicação entre os agentes, a troca de informações entre os agentes e a plataforma de servidores Web, bem como o tempo de execução das tarefas dessa plataforma.

Utilizando os modelos de redes de Petri coloridas, foram realizadas diferentes simulações, quais sejam:

- Utilização da plataforma WS-DSAC sem reconfiguração dinâmica;
- Utilização da plataforma WS-DSAC com reconfiguração dinâmica, ou seja, a plataforma WS-DSAC integrada a arquitetura DARC.

Os resultados das simulações mostraram uma melhoria do desempenho da plataforma quando integrada à arquitetura DARC. Foram realizadas simulações modificando-se o limite de carga máxima monitorado pelo agente de monitoramento (80%, 85% e 90% da carga máxima). Verificou-se que, quando a arquitetura DARC tomava atitude de reconfiguração, o cluster monitorado atingia 80% de sua carga máxima permitida, reduzindo portanto a taxa de rejeição de requisições.

Para a obtenção dos modelos em redes de Petri estocásticas o sistema WS-DSAC/DARC foi modelado antes em UML para a inclusão das anotações de desempenho. Para a transformação dos modelos UML em GSPN, utilizou-se uma ferramenta intermediária chamada *Corel Scenario Model*. Utilizando os modelos de redes de Petri estocásticas, foram realizadas diferentes simulações, quais sejam:

- Utilização da plataforma WS-DSAC sem reconfiguração dinâmica; e
- Utilização da plataforma WS-DSAC com reconfiguração dinâmica, ou seja, a plataforma WS-DSAC integrada à arquitetura DARC.

Os resultados das simulações mostraram que a utilização da arquitetura DARC melhora o atendimento ao cliente diminuindo as taxas de rejeição, sem impactar no tempo de resposta da plataforma WS-DSAC.

Além das modelagens discutidas acima, foi implementado um ambiente real de testes usando-se uma plataforma de servidores Web com dois clusters e dois servidores por cluster. Foram realizados experimentos nesta plataforma com e sem reconfiguração dinâmica.

Os resultados foram compatíveis com os resultados obtidos nos modelos, mostrando que a solução proposta melhora o desempenho do sistema.

7.1 Trabalhos Futuros

Os trabalhos futuros estão direcionados à comunicação da Arquitetura DARC com outras plataformas. Para tanto, é necessário o desenvolvimento de um padrão de comunicação da arquitetura de reconfiguração dinâmica. Isso é importante para que outras plataformas possam interagir com a arquitetura proposta neste trabalho de forma transparente, ou seja, sem a necessidade de alterações em códigos destas plataformas. Esta arquitetura pode pertencer também a outros domínios de conhecimento, diferentes do contexto de servidores Web. Por exemplo a arquitetura poderia realizar a reconfiguração de recursos na área financeira.

A quantidade de recursos disponíveis em um cluster é um fator de limitação para a plataforma WS-DSAC. Porém a arquitetura DARC realiza a reconfiguração destes no seu ambiente e, se não existir recursos disponíveis para serem reconfigurados a arquitetura não poderá exercer sua função. A fim de resolver esse problema propõe-se a utilização de agentes móveis na arquitetura DARC. O papel desse agente móvel seria o de negociar a entrada de novos recursos web de uma grade quando fosse solicitado ou pelo administrador ou pelos agentes de execução (ver figura 7.2). Esta negociação verificaria a disponibilidade destes novos recursos em pertencerem aos clusters da plataforma de servidores Web.

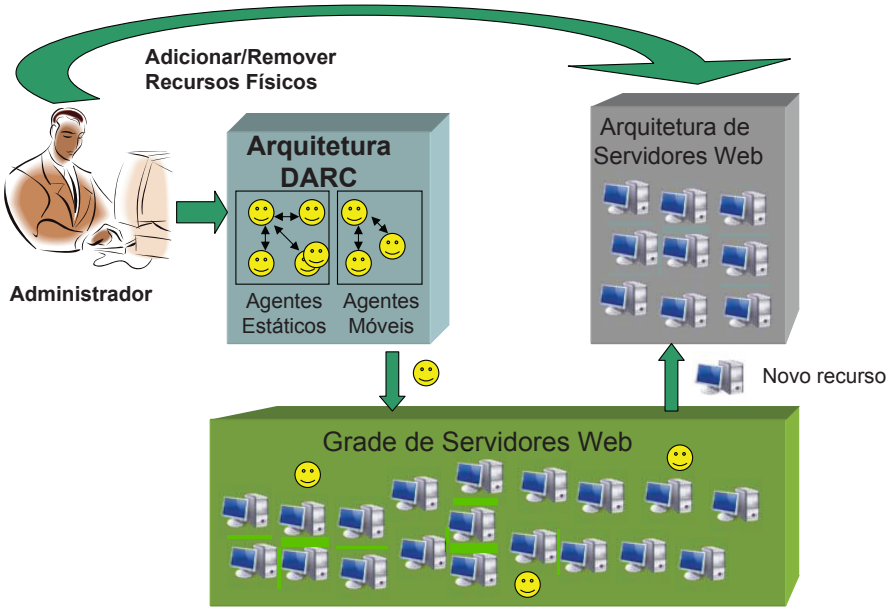

Figura 7.2: Utilização de Agente Móvel Negociador.

Capítulo 8

Publicações

8.1 Publicações Nacionais

MARQUES, Carla Katarina de Monteiro ; BARROSO, G. C. ; Marinho, I. C. ; AGUILAR, M. F. ; Serra, A. B. . Uma Solução para Aumentar a Disponibilidade de Serviços em Clusters de Servidores Web Baseada em Sistemas Multiagentes. In: Simpósio Brasileiro de Sistemas de Informação, 2009, Brasília.

8.2 Publicações Internacionais

C.K.M. Marques, S. Ilarri and G.C. Barroso, "DARC: A Dynamic Architecture for Reconfiguration of Web Servers Clusters Using Multiagent Systems", Fifth International Conference on Networking and Services (ICNS'09), Valencia (Spain), IEEE Computer Society, ISBN 978-0-7695-3586-9, pp. 169-174, April 2009.

Marques, C.; Serra, A.; Oliveira, I.; Barroso, G.; Fiallos, M., "The Simulation of DARC (Dynamic Architecture for Reconfiguration of Web Servers Clusters) in Petri Nets", First International Conference on Advances in System Simulation, 2009. (SIMUL'09), Porto (Portugal), IEEE Computer Society, pp. 112-118, September 2009.

C.K.M. Marques, S. Ilarri, J. Merseguer and G.C. Barroso, "Performance Analysis of a Dynamic Architecture for Reconfiguration of Web Servers Clusters", Sixth International Conference on Networking and Services (ICNS'10), Cancun (Mexico), IEEE Computer Society, ISBN 978-0-7695-3969-0, pp. 224-229, March 2010.

Referências Bibliográficas

ABDELLATIF, T.; KORNAS, J.; STEFANI, J.-B. Reengineering j2ee servers for automated management in distributed environments. *IEEE Distributed Systems Online*, IEEE Computer Society, Los Alamitos, CA, USA, v. 8, n. 11, 2007. ISSN 1541-4922.

ABDELZAHER, T.; SHIN, K. G.; BHATTI, N. Performance guarantees for web server end-systems: A control-theoretical approach. *IEEE Transactions on Parallel and Distributed Systems*, v. 13, p. 2002, 2001.

ADAM, C.; STADLER, R. Adaptable server clusters with QoS objectives. *International Symposium on Integrated Network Management*, p. 149–162, May 2005.

BERNARDI, S.; DONATELLI, S. Stochastic petri nets and inheritance for dependability modelling. In: *PRDC '04: Proceedings of the 10th IEEE Pacific Rim International Symposium on Dependable Computing (PRDC'04)*. [S.l.: s.n.], 2004. p. 363–372.

BICA, F. *Eletrotutor III: Uma abordagem multiagente para o Ensino à distância.* Dissertação (Mestrado) — Universidade Federal do Rio Grande do Sul, Janeiro 2000.

CARDOSO, J.; VALETTE, R. *Redes de Petri.* [S.l.]: Ed. da UFSC, 1997.

CEDRO D. M. ; SOUZA, J. N. . S. A. B. Gestão da qos em arquiteturas de grelhas computacionais orientadas a serviços. *9ª Conferência da Associação Portuguesa de Sistemas de Informação, CAPSI 2009*, Viseu, Portugal, 2009.

CHERKASOVA, L.; PHAAL, P. Session-based admission control: A mechanism for peak load management of commercial web sites. *IEEE Trans. Comput.*, v. 51, n. 6, p. 669–685, 2002.

CHIOLA, G. et al. Generalized stochastic petri nets: A definition at the net level and its implications. *IEEE Transactions on Software Engineering*, v. 19, n. 2, p. 89–107, 1993.

CUI, J. F.; CHAE, H. S. Agent-based design of load balancing system for rfid middlewares. *International Workshop on Future Trends of Distributed Computing Systems*, p. 21–30, March 2007.

FIPA. *FIPA Agent Communication Language Overview, Foundation for Intelligent Physical Agents*. [S.l.], 1999. Disponível em: <Disponível em http://www.fipa.org [Consultado em 19/11/2009]>.

HERRERO, P. et al. Amble: An awareness model for balancing the load in collaborative grid environments. *International Conference on Grid Computing*, p. 246–253, September 2006.

HERRERO, P. et al. An agents-based cooperative awareness model to cover load balancing delivery in grid environments. *Lecture Notes in Computer Science*, p. 64–74, November 2007.

HEUSER, C. A. *Modelagem Conceitual de Sistemas: Redes de Petri*. [S.l.]: Material de apoio utilizado no curso de pós-graduação em Ciência da Computação da UFRGS., 1990.

JENSEN, K.; KRISTENSEN, L.; WELLS, L. Coloured petri nets and cpn tools for modelling and validation of concurrent systems. *International Journal on Software Tools for Technology Transfer (STTT)*, v. 9, n. 3, p. 213–254, June 2007. Disponível em: <http://dx.doi.org/10.1007/s10009-007-0038-x>.

LAVENBERG., S. *Statistical Analysis of Simulation Outputs*. [S.l.], 1980.

LI, Z. et al. A novel approach for dynamic reconfiguration of the distribution network via multi-agent system. In: *Electric Utility Deregulation and Restructuring and Power Technologies, 2008. DRPT 2008. Third International Conference on*. [S.l.: s.n.], 2008. p. 1305–1311.

MOHEUDDIN, S.; NOORE, A.; CHOUDHRY, M. A novel approach for dynamic reconfiguration of the distribution network via multi-agent system. In: *International Journal of Computational Intelligence*. [S.l.: s.n.], 2009. v. 5, n. 1, p. 60–71.

MORRISON, R. S. *Cluster Computing - Architectures, Operating Systems, Parallel Processing and Programming Languages*. [S.l.]: Distributed under the GNU General Public Licence, 2003.

MURATA, T. Petri nets: Properties, analysis and applications. In: *Proceedings of the IEEE*. [S.l.: s.n.], 1989. p. 541–580. Proceedings of the IEEE, volume 77, number 4.

NEHRA, N.; PATEL, R. B. Towards dynamic load balancing in heterogeneous cluster using mobile agent. *International Conference on Computational Intelligence and Multimedia Applications*, p. 15–21, December 2007.

NEHRA, N.; PATEL, R. B.; BHAT, V. K. A multi-agent system for distributed dynamic load balancing on cluster. *International Conference on Advanced Computing and Communications*, p. 135–138, August 2006.

NETTO, M. A. S. et al. Transparent resource allocation to exploit idle cluster nodes in computational grids. In: *E-SCIENCE '05: Proceedings of the First International Conference on e-Science and Grid Computing*. Washington, DC, USA: IEEE Computer Society, 2005. p. 238–245. ISBN 0-7695-2448-6.

OLEJNIK, R.; BOUCHI, A.; TOURSEL, B. An object observation for a java adaptative distributed application platform. *International Conference on Parallel Computing in Electrical Engineering*, p. 171–176, September 2002.

PENHA, D. O.; FREITAS, H. C.; MARTINS, C. A. Modelagem de sistemas computacionais usando redes de petri: aplicação em projeto, análise e avaliação. IV Escola Regional de Informática RJ/ES, Novembro 2004.

PETRI, C. A. *Kommunikation mit Automaten*. Tese (Doutorado) — Bonn: Institut für Instrumentelle Mathematik, Schriften des IIM Nr. 2, 1962. Second Edition:, New York: Griffiss Air Force Base, Technical Report RADC-TR-65–377, Vol.1, 1966, Pages: Suppl. 1.

PETRIU, D.; WOODSIDE, M. An intermediate metamodel with scenarios and resources for generating performance models from UML designs. *Software and Systems Modeling*, Springer, v. 6, n. 2, p. 163–184, 2007. ISSN 1619-1366.

PONCI, F.; DESHMUKH, A. A mobile agent for measurements in distributed power electronic systems. In: *Instrumentation and Measurement Technology Conference Proceedings, 2008. IMTC 2008. IEEE.* [S.l.: s.n.], 2009. v. 58, n. 5, p. 1657–1668.

RAMCHANDANI, C. *Analysis of asynchronous concurrent systems by timed Petri nets*. Cambridge, MA, USA, 1974.

REIS, L. P. *Coordenação em Sistemas Multi-Agente: Aplicações na Gestão Universitária e Futebol Robótico*. Tese (Ph. D. Thesis) — Faculdade de Engenharia da Universidade do Porto, Portugal, June 2003.

SERRA, A. et al. Controle de admissão e diferenciação de serviços em clusters de servidores web. *Anais do SBRC-SBC, 2005*, s.n, p. 93–105, 2005.

SHI, Y. et al. (Ed.). *Computational Science - ICCS 2007, 7th International Conference, Beijing, China, May 27 - 30, 2007, Proceedings, Part II*, v. 4488 de *Lecture Notes in Computer Science*, (Lecture Notes in Computer Science, v. 4488). [S.l.]: Springer, 2007. ISBN 978-3-540-72585-5.

SMITH, C. U.; WILLIAMS, L. G. Software performance engineering. Kluwer Academic Publishers, Norwell, MA, USA, p. 343–365, 2003.

SUGAWARA, T. et al. Total performance by local agent selection strategies in multi-agent systems. In: *AAMAS '06: Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems*. New York, NY, USA: ACM, 2006. p. 601–608. ISBN 1-59593-303-4.

SUNG, H. et al. Dynamic clustering model for high service availability. *International Symposium on Autonomous Decentralized Systems*, p. 311–317, March 2007.

WANG, J. et al. Agent based load balancing model for service based grid applications. *International Conference on Computational Intelligence and Security*, p. 486–491, November 2006.

WOOLDRIDGE, M. *An Introduction to MultiAgent Systems*. [S.l.]: John Willey and Sons Ltd, 2002.

WOOLDRIDGE, M.; JENNINGS, N. R. Intelligent agents: Theory and practice. *Knowledge Engineering Review*, v. 10, p. 115–152, 1995.

ZHANG, Z. H. et al. Easy and reliable cluster management: The self-management experience of Fire Phoenix. *International Parallel and Distributed Processing Symposium*, April 2006.