UNIVERSIDADE FEDERAL RURAL DO SEMI-ÁRIDO
UNIVERSIDADE DO ESTADO DO RIO GRANDE DO NORTE
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

HUGO NATHAN BARBOSA RÉGIS

DETECÇÃO DINÂMICA DE ANTIPADRÕES EM SISTEMAS BASEADOS EM SERVIÇOS UTILIZANDO UM SISTEMA MULTIAGENTE

MOSSORÓ – RN

2017

HUGO NATHAN BARBOSA RÉGIS

DETECÇÃO DINÂMICA DE ANTIPADRÕES EM SISTEMAS BASEADOS EM SERVIÇOS UTILIZANDO UM SISTEMA MULTIAGENTE

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação – Associação ampla entre a Universidade Federal Rural do Semi-Árido e a Universidade do Estado do Rio Grande do Norte, como requisito para obtenção do título de Mestre em Ciência da Computação.

Orientadora: Profª. Drª. Carla Katarina de Monteiro Marques – UERN.

MOSSORÓ – RN

2017

Ficha catalográfica gerada pelo Sistema Integrado de Bibliotecas
e Diretoria de Informatização (DINF) - UERN,
com os dados fornecidos pelo(a) autor(a)

B337d Barbosa Régis, Hugo Nathan.
DETECÇÃO DINÂMICA DE ANTIPADRÕES EM SISTEMAS BASEADOS EM SERVIÇOS UTILIZANDO UM SISTEMA MULTIAGENTE / Hugo Nathan Barbosa Régis - 2017.
58 p.

Orientadora: Carla Katarina de Monteiro Marques.
Coorientadora: .
Dissertação (Mestrado) - Universidade do Estado do Rio Grande do Norte, Pós-graduação em Ciência da Computação, 2017.

1. Antipadrões. 2. Detecção Dinâmica. 3. Sistemas Baseados em Serviços. 4. Sistema Multiagente. I. de Monteiro Marques, Carla Katarina, orient. II. Título.

HUGO NATHAN BARBOSA RÉGIS

Detecção Dinâmica de Antipadrões em Sistemas Baseados em Serviços Utilizando um Sistema Multiagente.

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação para a obtenção do título de Mestre em Ciência da Computação.

APROVADA EM: _____/_____/_____

Profa. Dra. Carla Katarina de Monteiro Marques
Orientadora e Presidente

Profa. Dra. Francisca Aparecida Prado Pinto
Universidade do Estado do Rio Grande do Norte

Prof. Dr. Henrique Jorge Amorim Holanda
Universidade do Estado do Rio Grande do Norte

Prof. Dr. Giovanni Cordeiro Barroso
Universidade Federal do Ceara

Àqueles que sempre acreditaram em mim:
Meus pais, Francisco Valderi e Maria José.

AGRADECIMENTOS

A minha família por todo apoio desde o início do meu processo pessoal e acadêmico, por me fornecer amparos emocionais e me incentivar em todo o momento a não desistir e persistir nessa caminhada árdua e de grande significado;

A Professora orientadora Carla Katarina, que não mediu esforços para me acompanhar, mesmo que longe sempre tentou estar presente, tirar as dúvidas possíveis. Obrigado por me incentivar sempre a continuar e procurar fornecer melhorias para o meu trabalho;

A minha namorada Milena por toda a sua paciência comigo nos meus momentos de stress, por sempre me incentivar a ser o melhor que posso ser e a apoiar minhas decisões;

Aos Professores da banca: Henrique, Geovanni e Aparecida, pela participação e sugestão de melhorias para este trabalho;

Aos Professores do PPGCC pelo conhecimento compartilhado durante as disciplinas;

Aos meus amigos de laboratório: Isaias, Kennedy, Samuel e Alefy pelo apoio, diálogos, pela troca de experiências e aos meus colegas de turma e laboratório;

As minhas amigas Tayana e Sângila que sempre estiveram me dando o apoio necessário, me amparando em momentos de preocupações e me mostrando que tudo daria certo;

Aos meus amigos Paulo e Wallace pelas conversas descontraídas nos fins de semana e lazeres proporcionados;

A meu primo Reuber que dividiu apartamento comigo desde a graduação até o mestrado e pelas dicas fornecidas no processo de formação acadêmica.

“O tempo amadurece todas as coisas.
Nenhum homem nasce sábio.”
Cervantes.

RESUMO

Durante o desenvolvimento de Sistemas Baseados em Serviços (SBS), soluções ruins, erros de projeto ou implementação podem conduzir ao surgimento de antipadrões, que em oposição aos padrões, são especificações ruins para problemas recorrentes. Antipadrões acarretam em desvios de funcionalidade durante a execução do sistema, não satisfazendo adequadamente aos seus requisitos, o que pode degradar a Qualidade de Serviço (QoS) de um SBS. Sua detecção e correção são de extrema importância pois o surgimento de antipadrões pode dificultar a manutenção e a futura evolução do SBS. Como método de detecção, este trabalho propõe o uso de um Sistema Multiagente (SMA), realizando buscas em tempo de execução e notificando ao administrador do sistema sobre quedas de QoS no SBS causadas por antipadrões. A detecção desses antipadrões permite a atuação dos agentes na correção do projeto, usando a solução recomendada pelo SMA para o problema detectado. Os resultados mostraram que, nos piores casos, o surgimento de antipadrões chegou a extrapolar em até o dobro de tempo de resposta dos serviços em um Web Service. O SMA foi capaz de detectar, em tempo de execução, os 3 antipadrões estudados no trabalho. Quando adotadas, as soluções propostas pelo SMA também normalizaram o tempo de resposta.

Palavras-Chave: Antipadrões, Detecção Dinâmica, Sistemas Baseados em Serviços, Sistema Multiagente.

ABSTRACT

During development of Service-Based Systems (SBS), bad solutions, design or implementation errors can lead to antipatterns that, as opposed to patterns, are bad specifications for recurring problems. Antipatterns result in deviations of functionality during the execution of the system, not adequately satisfying its requirements, which can degrade the Quality of Service (QoS) of the SBS. Its detection and correction are of extreme importance, the emergence of antipatterns may hinder future maintenance and evolution of SBS. As detection method, this work proposes the use of a Multiagent System (MAS), executing search at runtime and notifying the system administrator of QoS drops on the SBS caused by antipatterns. The detection of these antipatterns allows the agents to act in correction the project using the solution recommended by the MAS for the detected problem. The results show that, in the worst cases, the emergence of antipatterns reached extrapolate up to twice the response time of services in a Web Service. The MAS was able to detect, in execution time, the 3 antipatterns studied at work. When adopted, the solutions proposed by the MAS also normalized the response time.

Keywords: Antipatterns, Dynamic Detection, Service-Based Systems, Multiagent System.

LISTA DE FIGURAS

LISTA DE GRÁFICOS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

ACC	*Agent Communicattion Channel*
ACL	*Agent Communication Language*
AMS	*Agent Management System*
CT	*Container Table*
DF	*Directory Facilitator*
FIPA	*Foundation for Intelligent Physical Agents*
GADT	*Global Agent Descriptor Table*
GUI	*Graphical User Interface*
HTTP	*Hypertext Transfer Protocol*
IdA	Identificador do Agente
IEEE	Instituto de Engenheiros Eletricistas e Eletrônicos
JADE	*Java Agent DEvelopment Framework*
MCC	Mestrado em Ciência da Computação
MIDP	*Mobile Information Device Profile*
MTS	Serviço de Transporte de Mensagem
PA	Plataforma Agente
P-EA	Algoritmo Evolutivo Paralelo cooperativo
POO	Programação Orientada a Objetos
QoS	Qualidade de Serviço
REST	*REpresentational State Transfer*
SBS	Sistemas Baseados em Serviços

SCA	*Service Component Architecture*
SGA	Sistema de Gerenciamento do Agente
SOA	Arquitetura Orientada a Serviço
SOAP	*Simple Object Access Protocol*
SOP	*Service-Oriented Programming*
SMA	Sistema MultiAgente
SOFA	*Service Oriented Framework for Antipatterns*
UDDI	*Universal Description, Discovery and Integration*
W3C	*World Wide Web Consortium*
WSDL	*Web Service Description Language*
XML	*eXtensible Markup Language*

SUMÁRIO

1 INTRODUÇÃO

A Engenharia de Software é definida como a aplicação sistemática de conhecimentos científicos e tecnológicos, métodos e experiência para a concepção, implementação, teste e documentação de *software* [ISO/IEC/IEEE (2010)]. Os conhecimentos e métodos tecnológicos aplicados ao desenvolvimento de *software* evoluem regularmente para desenvolver sistemas de *software* mais flexíveis, de alto desempenho e alta manutenibilidade. Portanto, os paradigmas de desenvolvimento de software existentes atuam na reutilização, facilidade de manutenção e evolução do *software*.

Um dos primeiros paradigmas de desenvolvimento de *software* é o desenvolvimento procedural, simples mas poderoso que segue a execução sequencial de um programa. No entanto, tem várias desvantagens: (1) é difícil relacionar conceitos do mundo real através de seu paradigma de programação; (2) os sistemas são difíceis de manter quando o código cresce; (3) os dados estão publicamente expostos, isto é, não seguros. Para superar essas armadilhas, a Programação Orientada a Objetos (POO) foi introduzida com maior produtividade e reutilização de desenvolvimento de *software*, com menor custo de manutenção de *software* e maior segurança de dados [Champeaux et al. (1993)]. Além disso, o desempenho do *software* e a qualidade do software melhoraram significativamente.

Depois que a internet foi inventada, as indústrias necessitavam conduzir seus negócios e transações remotamente pela *Web*, um novo paradigma de desenvolvimento de *software* era inevitável. Assim, surgiu um novo paradigma de programação, *Service-Oriented Programming* (SOP), para o desenvolvimento de serviços de entidades funcionais autônomas independentes de plataforma e de acesso remoto [Papazoglou et al. (2003); Erl (2004); Turner et al. (2003)]. Comparado com os paradigmas processuais e POO, o SOP introduziu uma camada adicional de abstração de software na camada de serviço, a vantagem dessa camada adicional é que ele ajuda a propagar as mudanças de negócios rapidamente e pode reduzir significativamente os esforços de manutenção [Woods et Mattern (2006)]. Um sistema baseado em serviços baseia-se em blocos de construção e na Arquitetura Orientada a Serviços (SOA) como sua arquitetura subjacente [Erl (2005)].

SOA é uma coleção de princípios e metodologias para projetar e desenvolver Sistemas Baseados em Serviços (SBS) [Papazoglou et al. (2003); Erl (2005); Heffner et al. (2007)]. SOA ajuda as organizações de TI a satisfazerem as suas necessidades de negócio, compondo

serviços funcionais reutilizáveis independentes de plataforma, que encapsulam as aplicações e as lógicas de negócio [Erl (2005)]. O rápido crescimento da SOA é fortemente apoiado pelos principais fornecedores de software, por exemplo, IBM, Sun e SAP, oferecendo plataformas de *middleware* orientadas a serviços, ferramentas de desenvolvimento e implementação de SBS com ambientes de desenvolvimento.

Multi Service é um exemplo típico de antipadrão de serviço em SBS [Dudney et al. (2003)]. *Multi Service* representa um serviço que implementa uma longa lista de operações que variam em abstrações de negócios. Um serviço implementado como um *Multi Service* não é facilmente reutilizável e exibe uma baixa coesão entre suas operações. Sendo sobrecarregado por muitas solicitações de clientes diferentes, o *Multi Service* pode ficar frequentemente inacessível aos seus usuários finais. Essa carga excessiva, no entanto, pode ser reduzida pela implantação de várias instâncias do serviço, que não é uma resolução de baixo custo.

A avaliação do projeto de um SBS envolve uma análise estática, enquanto a avaliação da sua QoS envolve a medição de vários aspectos dinâmicos (isto é, desempenho e disponibilidade). Detectar antipadrões de serviço em SBS é um dos meios eficazes de analisar e avaliar os critérios de projeto e QoS de SBS. A detecção de antipadrões de serviço dentro dos SBS é também parte das seguintes atividades de manutenção: (1) melhorar a qualidade do projeto e QoS; e (2) diminuir o esforço e o custo de sua futura manutenção e evolução.

1.1 JUSTIFICATIVA

Durante a manutenção e a evolução de Sistemas Baseados em Serviços (SBS), as várias alterações funcionais e não-funcionais podem degradar a qualidade do projeto e implementação e a qualidade de serviço (QoS) dos SBS, podendo causar a aparição de soluções ruins para problemas de projeto recorrentes, os antipadrões de serviço. Estas soluções ruins, isto é, antipadrões de serviço, estão documentadas na literatura como as más práticas de concepção comuns durante o desenvolvimento de serviços e SBS. Em outras palavras, os antipadrões de serviço tipicamente capturam alguns aspectos dos problemas de projeto, complexidade de componentes, falta de coesão ou acoplamento alto entre um grupo

de serviços (ou componentes). Avaliar a qualidade de projeto e a QoS de qualquer sistema através da detecção de antipadrões pode ajudar a facilitar sua manutenção e evolução.

A detecção automática utiliza software para a detecção e é recomendada, pois um processo manual gera lentidão no desenvolvimento do projeto. A detecção dinâmica é realizada em tempo de execução, isto é, analisando o comportamento das variáveis necessárias para o surgimento do antipadrão durante a execução do sistema e não somente pelo código, como ocorre na detecção estática, permitindo que um sistema autônomo corrija o desvio de funcionalidade detectado.

O uso de agentes na detecção acelera o processo pois cada agente é responsável por detectar cada antipadrão paralelamente.

1.2 OBJETIVO GERAL

Este trabalho tem como objetivo desenvolver uma abordagem de detecção automática e dinâmica (em tempo de execução) de antipadrões em Sistemas Baseados em Serviços suportada por um Sistema Multiagente. Com essa abordagem os agentes identificam os antipadrões *Bottleneck Service*, *Multi Service* e *Service Chain* agindo sobre o ambiente do *Web Service* (WS) solucionando problemas de performance que venham a degradar a QoS e informam ao administrador do sistema sobre a necessidade de refatoração no código fonte. Os agentes são capazes de atuar apenas sobre antipadrões de desempenho. Caso seja identificado um antipadrão cuja solução seja a refatoração de código, o SMA informará ao administrador para que o mesmo faça a alteração.

1.3 OBJETIVOS ESPECÍFICOS

Para atingir a meta proposta, os seguintes objetivos específicos precisam ser alcançados:

1) Estudar, compreender e aplicar os conceitos, métodos e técnicas relacionados ao domínio de conhecimento de projetos de Sistemas Baseados em Serviços;

2) Estudar, compreender e aplicar o paradigma de antipadrões, os impactos de sua utilização;
3) Estudar o Framework JADE para o desenvolvimento do sistema multiagente;
4) Validar o sistema desenvolvido com testes de detecção de antipadrões;
5) Avaliar o quão significante é o impacto dos antipadrões para a QoS em um Sistema Baseado em Serviços.

1.4 ORGANIZAÇÃO DA DISSERTAÇÃO

Este documento está organizado da seguinte forma: No Capítulo 2 é apresentado o referencial teórico utilizado na elaboração deste trabalho, abordando os conceitos de Arquitetura Orientada a Serviço, Sistemas Baseados em Serviços, Antipadrões e Sistema Multiagente; No Capítulo 3 são apresentados os trabalhos relacionados, esclarecendo o diferencial da proposta deste trabalho; No Capítulo 4 são apresentadas a especificação e a implementação da arquitetura para o desenvolvimento do Sistema Multiagente; No Capítulo 5 são apresentados os resultados obtidos nos testes de detecção dos antipadrões; E, por fim, no Capítulo 6 são apresentadas as considerações finais e trabalhos futuros.

2 REFERENCIAL TEÓRICO

Nesta seção, são apresentados alguns conceitos importantes e temas relevantes nas áreas de Sistemas Baseados em Serviços e Sistemas Multiagentes, e/ou relacionados ao estudo proposto.

2.1 ARQUITETURA ORIENTADA A SERVIÇO

Uma Arquitetura Orientada a Serviços (SOA) é composta por estes quatro elementos: (1) um aplicativo *front-end*, (2) um conjunto de serviços, (3) um repositório contendo especificações de serviços, o que facilita a procura de serviços, e (4) um serviço de barramento que fornece um mecanismo para a suas interações [Rosen et al. (2008)]. Um serviço implementa pelo menos uma interface, que lista e expõe suas capacidades funcionais. Em geral, um serviço implementa uma abstração de negócios. A seguir, identificamos e discutimos os quatro conceitos em sistemas baseados em SOA.

Um serviço em uma arquitetura SOA é uma entidade funcional inteiramente autônoma onde vários serviços autônomos podem ser compostos para atingir metas de negócios de nível mais alto [Erl (2005)]. Os serviços no contexto da SOA possuem duas noções principais: (1) executam tarefas relacionadas ao negócio e não são elementos visíveis e (2) os serviços são entidades de caixa-preta e apenas expõem suas funcionalidades ao ocultar seus detalhes de implementação. Um serviço tem pontos finais através dos quais os potenciais clientes se conectarão a ele. Um nó de extremidade contém um conjunto de operações relacionadas. Cada ponto final define obrigatoriamente um tipo de ligação, isto é, como um cliente irá comunicar com o serviço e um endereço físico, em que o cliente do ponto final está hospedado.

Conforme ilustrado na Figura 1, os sistemas baseados em SOA dependem do modelo de comunicação cliente-servidor e suas comunicações dependem de mensagens SOAP (*Simple Object Access Protocol*) ou de solicitações / respostas HTTP (*Hypertext Transfer Protocol*). Em geral, o cliente pode ser um aplicativo *front-end* ou um navegador da *Web*. O servidor detém o(s) serviço(s), gerencia e atende as solicitações do(s) cliente(s).

Figura 1 - Modelo Cliente-Servidor.

Uma vez que um provedor de serviços elabora um serviço, ele deve descrevê-lo utilizando uma linguagem de especificação, fornecendo a localização física, ou seja, ponto final do serviço para que potenciais clientes se conectem e consumam as funcionalidades fornecidas. O W3C (*World Wide Web Consortium*) define um ponto final de serviço como "uma associação entre uma ligação de interface totalmente especificada e um endereço de rede, especificado por um URI que pode ser utilizado para comunicar com uma instância de um serviço *Web*" [Christensen et al. (2011)].

As especificações WSDL (*Web Service Description Language*) são escritas em formato XML (*eXtensible Markup Language*) e são publicadas na internet. O WSDL contém um conjunto de pontos finais associados a um conjunto de mensagens, isto é, dados e operações trocados, que estão ligados a um protocolo de ligação. Assim, uma especificação WSDL tem três componentes principais: (1) definições de tipos de dados e / ou mensagens, (2) operações de quatro tipos diferentes, ou seja, solicitação unidirecional, solicitação / resposta, resposta unidirecional e notificação e (3) ligações de serviço para conectar a porta de um serviço pelos clientes.

O UDDI (*Universal Description, Discovery and Integration*), um registro independente da plataforma e baseado em XML, desempenha um papel importante em ajudar os clientes a procurar serviços já publicados pela internet. A UDDI é formada por três componentes: (1) páginas brancas contendo endereços e contatos dos provedores, (2) páginas amarelas que fornecem categorizações de serviços padrão do setor e (3) páginas verdes contendo detalhes técnicos sobre serviços, suas capacidades funcionais e protocolos de comunicação. UDDI depende de mensagens SOAP para interagir com clientes.

As etapas da busca do serviço, como é mostrado na Figura 2.

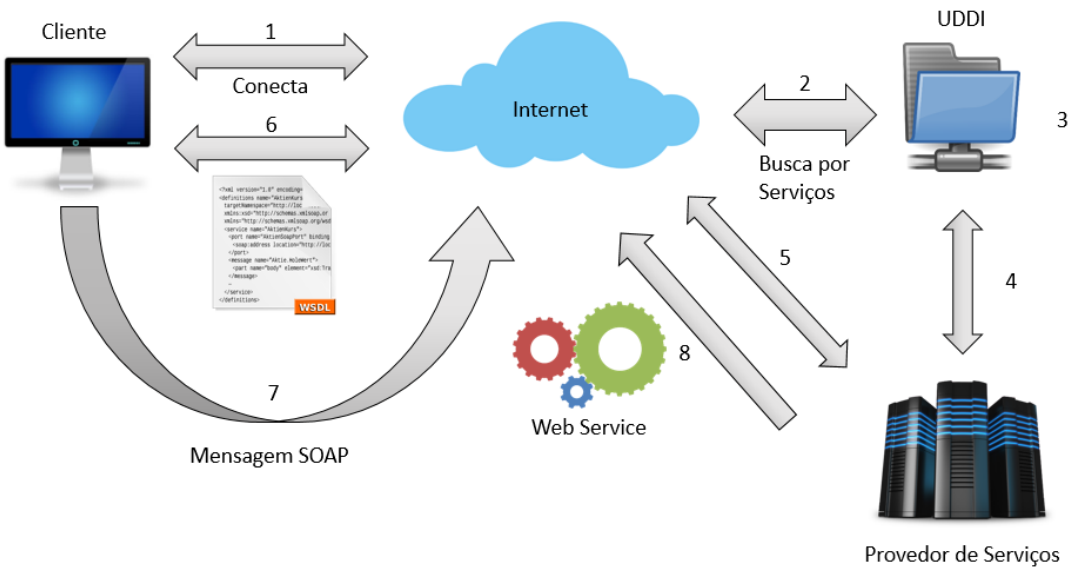

Figura 2 - Um cenário típico de interação do serviço SOA.
Fonte: Christensen et al. (2011) adaptado.

- Passo 1: Um cliente requer um serviço da *Web* e o cliente procura um diretório;
- Passo 2: O cliente se conecta a um diretório UDDI para procurar serviços relevantes;
- Passo 3: O cliente e o diretório UDDI determinam se o serviço desejado está disponível;
- Passo 4: O diretório UDDI contata o provedor de serviços para a disponibilidade do serviço;
- Passo 5: O provedor de serviços responde com um documento WSDL se o serviço estiver disponível;
- Passo 6: O cliente se prepara para consumir o serviço utilizando um *stub* de cliente;
- Passo 7: O cliente interage com o serviço real usando as mensagens SOAP ou HTTP;
- Passo 8: O serviço responde com os dados comerciais do cliente a serem processados pelo cliente;

2.2 SISTEMAS BASEADOS EM SERVIÇOS

Os SBS são projetados e desenvolvidos aplicando padrões e princípios de projeto SOA [Erl (2009)] utilizando um número de diferentes tecnologias e estilos arquitetônicos, tipicamente *REpresentational State Transfer* (REST) [Fielding (2000)], *Service Component Architecture* (SCA) [Chappell (2007)] e SOAP *Web Services* [Alonso et al. (2003)]. Neste trabalho, nos referimos a estas como tecnologias SBS. Google Maps, Amazon, eBay, PayPal e FedEx são exemplos de SBS de escala de indústria. [Spanoudakis et Mahbub (2004)] definiram SBS como sistemas compostos dinamicamente por serviços Web autônomos e cuja composição é controlada por alguns processos de composição. A definição de SBS fornecida por Spanoudakis e Mahbub concentra-se apenas no aspecto composicional dos SBS. A definição a seguir generaliza: Os SBS são construídos em cima dos princípios de *design* da SOA e são compostos de serviços autônomos implementados com tecnologias heterogêneas como seus blocos de construção. [Palma et al. (2013)].

Os sistemas desenvolvidos com estas tecnologias SBS apresentam alta flexibilidade e agilidade, proporcionam rápidas mudanças nos negócios e, mais importante, têm alta reutilização [Erl (2007)]. No entanto, serviços e SBS não estão isentos de alguns desafios comuns de engenharia de *software* tais como manutenção e evolução que ocorrem com a adição ou modificação dos requisitos de usuário. Manutenção e evolução podem ocorrer devido a (1) alterações funcionais, isto é, alterações ao nível da concepção e da implementação; (2) alterações não funcionais, isto é, mudanças nos contextos de execução ou em acordos de nível de serviço.

Do ponto de vista da SOA, o desenvolvimento de serviços de alta manutenibilidade e SBS de boa qualidade são cruciais devido à sua natureza em constante evolução. Devido às mudanças rápidas nas necessidades de negócios (crescentes e globais), os SBS devem adaptar-se às exigências dos clientes, alterando as suas lógicas e regras empresariais subjacentes com o menor atraso [Newcomer et Lomow (2004)]. Um serviço altamente sustentável pode facilitar uma melhor adaptação. Além disso, restrições de tempo firmes, isto é, tempos de libertação de produto de software mais curtos, podem conduzir à produção de sistemas difíceis de serem mantidos. Os SBS operam em um ambiente de execução altamente dinâmico e, assim, os desenvolvedores do SBS precisam tomar um cuidado extra dos comportamentos funcionais desejados (capacidades funcionais do sistema) e não funcionais (rendimento, latência, disponibilidade, etc.) Depois de realizar todos os tipos de atividades de

manutenção. Assim, o desenvolvimento de SBS que exibem alta capacidade de manutenção é um dos fatores-chave para uma adoção bem-sucedida de SOA.

2.3 ANTIPADRÕES

Antipadrões descrevem uma solução de ocorrência comum para um problema que gera um desvio de funcionalidade com consequências negativas. O antipadrão pode ser o resultado de um gerente ou desenvolvedor não conhecer o padrão para uma solução, não ter conhecimento suficiente ou experiência em resolver um determinado tipo de problema, mudanças de trechos de código mal planejadas e pressão de tempo para execução do projeto. Quando devidamente documentado, os antipadrões descrevem: uma noção geral do problema, as causas primárias que levaram à noção geral; sintomas que descrevem como reconhecer a noção geral; as consequências da noção geral; e uma solução refatorada descrevendo como alterar o projeto. Neste trabalho serão analisados 3 antipadrões, todos descritos a seguir.

2.3.1 Catálogo de Antipadrões

O *Bottleneck Service* caracteriza um serviço que é altamente utilizado por outros serviços ou clientes. Seu tempo de resposta pode ser alto porque pode ser usado por muitos clientes externos, para os quais os clientes podem precisar esperar para ter acesso ao serviço. Além disso, sua disponibilidade também pode ser baixa devido ao tráfego.

Multi Service é um exemplo típicos de antipadrão de serviço em SBS que representa um serviço implementando uma longa lista de operações que variam em abstrações de negócios. Um serviço implementado como um *Multi Service* não é facilmente reutilizável e exibe uma baixa coesão entre suas operações. Sendo sobrecarregado por muitas solicitações de clientes diferentes, o serviço pode ficar frequentemente inacessível aos seus usuários finais. Essa carga excessiva, no entanto, pode ser reduzida pela implantação de várias instâncias do serviço, que não é uma solução de baixo custo.

Service Chain corresponde a uma cadeia de serviços que aparece quando os clientes solicitam invocações de serviço consecutivas para cumprir suas metas (um método necessita da resposta de outros métodos para retornar o valor final) [Dudney et al (2003)].

Os antipadrões *Bottleneck Service* e *Multi Service* possuem um caráter de detecção dinâmica pois degradam a QoS, já o antipadrão *Service Chain* possui uma detecção estática a nível de código fonte.

Na Tabela 1 estão organizados os principais pontos da problemática sobre a detecção dos antipadrões citados:

Tabela 1 – Atributos dos antipadrões.

Nome	Ocorrência	Contexto	Classificação	Consequências	Solução
Bottleneck Service	Aumento significativo na quantidade de **Clientes**	Tempo de resposta alto	Dinâmica	Baixa disponibilidade devido ao tráfego; Baixa QoS	Implantação de várias instâncias do serviço
Multi Service	Aumento significativo na quantidade de **Solicitações**	Sobrecarga de Solicitações	Dinâmica	Frequentemente inacessível aos seus usuários finais; Baixa QoS	Implantação de várias instâncias do serviço
Service Chain	Quantidade significativa de **Serviços Encadeados**	Invocações de serviço consecutivas	Estática	Baixa QoS	Refatoração do Código

2.4 SISTEMA MULTIAGENTE

Segundo [REIS (2003)] a área da 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.

Embora a Inteligência Artificial 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, para além de poder ser considerado apenas a uma subárea da IA.

Um agente é algo que age (a palavra agente vem do latim *agere*, que significa fazer). No entanto, espera-se que um agente computacional tenha outros atributos que possam distingui-lo de meros "programas", tais como operar sob controle autônomo, perceber seu ambiente, persistir por um período de tempo prolongado, adaptar-se a mudanças e ser capaz de assumir metas de outros [RUSSELL; NORVING (2010)].

Então, pode-se dizer que um agente é tudo o que pode ser considerado capaz de perceber seu ambiente por meio de sensores e de agir sobre esse ambiente por intermédio de atuadores [RUSSELL; NORVING (2010)]. Na Figura 4 é ilustrada essa ideia. Em outras palavras, um agente tem a capacidade de analisar um ambiente e atuar de forma autônoma, modificando ou não o ambiente, para alcançar suas metas. As características de um agente são: autonomia, interação, reação e iniciativa.

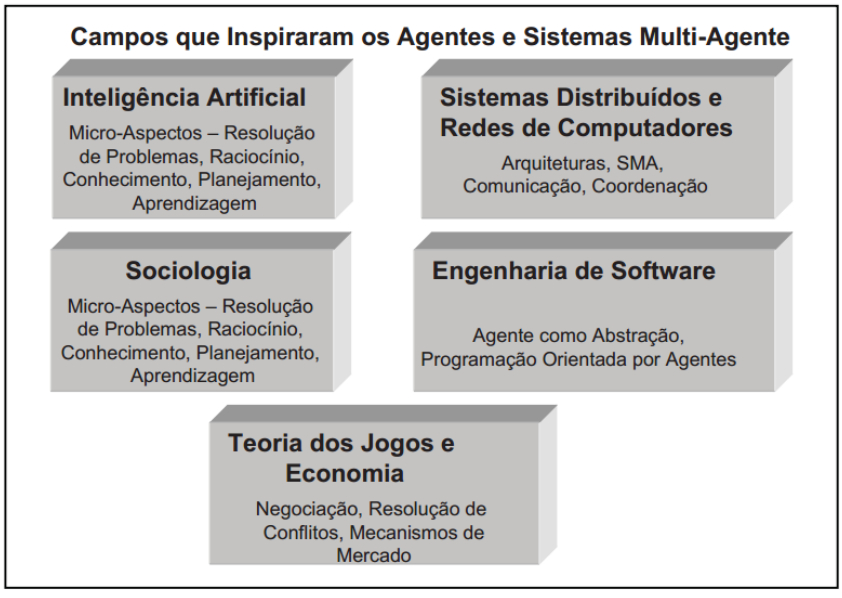

Figura 3 - Campos que inspiraram os agentes e Sistemas Multiagentes.

Fonte: RUSSELL; NORVING (2010).

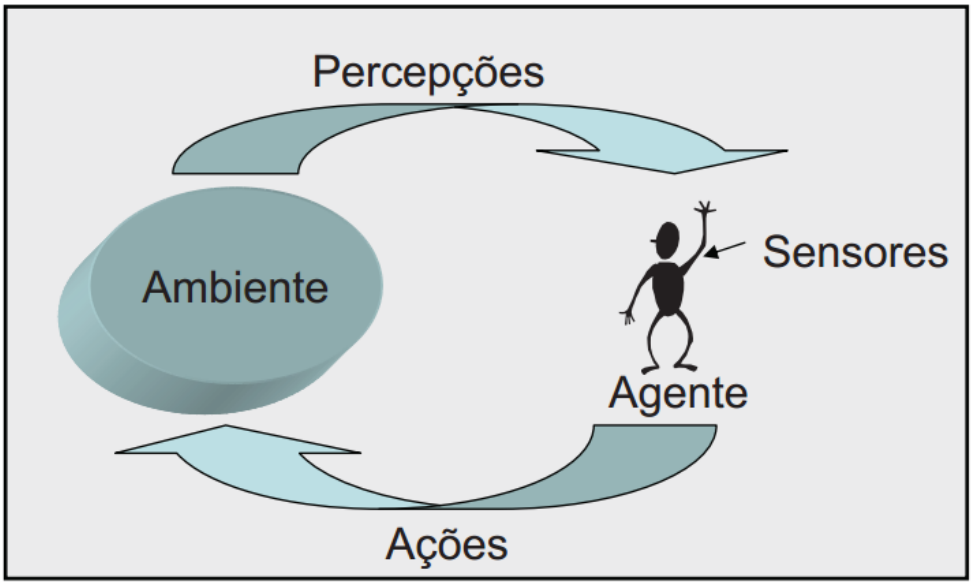

Figura 4 Agentes interagem com ambientes por meio de sensores e atuadores.

Fonte: RUSSELL; NORVING (2010).

O termo percepção faz referência às entradas perceptivas do agente em qualquer momento dado. Em geral, a escolha da ação de um agente em qualquer instante dado pode depender da sequência inteira de percepções observadas até o momento. Um agente é completamente especificado pela "função do agente", que mapeia a sequência de percepções em ações [RUSSELL; NORVING (2010)].

Russell e Norving (2010) descrevem quatro tipos básicos de agentes:

- Agentes reativos simples: esses agentes selecionam ações com base na percepção atual, ignorando o restante do histórico de percepções. Os agentes reativos simples têm a admirável propriedade de serem simples, mas se caracterizam por terem inteligência muito limitada;
- Agente reativo com estado: é um agente reativo mais complexo, pois mantém um estado interno que depende de seu histórico de percepções e ações. A cada nova percepção, o agente atualiza seu estado interno, e suas ações agora dependem não só da percepção recebida, mas também do seu estado interno. Esse agente também é chamado de agente baseado em modelo;
- Agentes baseados em objetivos: esses agentes precisam saber quais são os seus objetivos para que assim possam alcançá-los. Esse agente mantém um registro interno do estado do mundo baseado em suas percepções e também um conjunto de metas que ele tenta alcançar. Os objetivos são alcançados através de busca e planejamento;

- Agente baseado em utilidade: além dos objetivos, este agente é capaz de mensurar o grau de “felicidade” a partir do estado em que se encontra. Isto é feito através da função de utilidade que o agente mantém internamente. Essa função de utilidade mapeia um estado (ou uma sequência de estados) em um número real, que descreve o grau de “felicidade” associado.

Os agentes deste trabalho são reativos com estado. O estado interno é atualizado a cada nova solicitação de uma operação pelo cliente, modificando a média de tempo calculada e influenciando na tomada de ação do agente.

Segundo Chauhan (1997) as funções desempenhadas pelos agentes podem ser classificadas nos seguintes tipos:

- Agentes de Entretenimento: são agentes que tem como propósito o entretenimento. Dentro desse grupo, estão aqueles que são desenvolvidos especialmente para os mais diversos tipos de jogos;
- Agentes de Informação: são agentes que têm acesso e são capazes de coletar e manipular informações de muitas fontes. As informações obtidas normalmente são resultados de consultas propostas por usuários e/ou agentes;
- Agentes de Interface: são agentes que apoiam e fornecem assistência ao usuário. Eles observam e monitoram as ações do usuário, aprendem com ele e sugerem melhores formas para realizar uma tarefa. Seu aprendizado pode ser através de observação e imitação do usuário, ou por *feedback* positivo ou negativo, por instruções explícitas do usuário ou através de outros agentes;
- Agentes Colaborativos: são agentes que trabalham com autonomia e cooperação entre si e com o objetivo de realizar suas tarefas. Seus atributos chaves incluem autonomia, habilidade social, correspondência e pró-atividade. Visando obter uma organização coordenada, eles podem negociar para obter, mutuamente, acordos aceitáveis em alguns assuntos;
- Agentes Móveis: são agentes que possuem a habilidade de se mover de um *host* a outro de uma rede durante suas execuções, carregando consigo seus estados de execução. Questões como segurança, transparência, mecanismo de transporte e autenticação são bastante relevantes neste tipo de agente;

- Agentes Reativos: são agentes que não possuem modelo interno do seu ambiente. Em vez disso, eles respondem de acordo com o estado do ambiente em que estão inseridos;
- Agentes Híbridos: são agentes que combinam dois ou mais tipos de agentes com o objetivo de maximizar as capacidades e minimizar as deficiências de cada um deles.

Os agentes tratados neste trabalho são híbridos sendo tanto de informação (coletando os dados de tempo entre cada solicitação para detectar e corrigir os antipadrões) como de interface (sugerindo ao usuário/administrador do sistema sobre a diminuição no encadeamento de métodos).

Agentes de software, inteligentes ou não, são programas que executam tarefas específicas em favor de um usuário, de forma independente ou com alguma orientação. Um agente inteligente efetua, de forma reativa e/ou proativa, tarefas interativas adaptadas à necessidade de usuário humano ou de outros agentes. Para realizar estas tarefas, é necessário possuir um conjunto de características: independência, aprendizado, cooperação, raciocínio e inteligência [BUI; LEE, (1999) apud KANEKO, (2005)].

O projeto de um agente inteligente deve especificar sua percepção, suas ações e objetivos, ou seja, o *Performance Environment Actuators Sensors* (PEAS). Para Wooldridge (2009) as arquiteturas de agentes são, dessa maneira, arquiteturas de softwares para sistemas de tomada de decisão que estão inseridos em um ambiente.

Para Russell e Norving (2010) o ambiente onde o agente atua é onde o problema se encontra, para o qual o agente é a solução. Um dos ambientes mais importantes para agentes inteligentes é a Internet. As tecnologias de inteligência artificial servem de base para muitas ferramentas na internet, como mecanismos de pesquisa, sistemas de recomendação e sistemas de construção de *Web* sites. A seguir são apresentadas algumas propriedades de ambiente:

- Completamente Observável vs. Parcialmente Observável: Um ambiente é completamente observável se os sensores dos agentes detectam todos os aspectos que são relevantes para a escolha da ação. E pode ser parcialmente observável por causa dos ruídos ou sensores não apurados, ou simplesmente porque nem todos os estados do ambiente podem ser percebidos pelos sensores do agente;

- Determinístico vs. Estocástico: Se o próximo estado do ambiente é completamente determinado pelo atual estado e a ação executada pelo agente, então dizemos que o ambiente é determinístico, caso contrário, estocástico;
- Episódico vs. Sequencial: Em um ambiente episódico, a experiência do agente é dividida em episódios atômicos. Cada episódio consiste do agente percebendo e executando uma única ação. O próximo episódio não depende das ações tomadas nos episódios anteriores. No ambiente episódico a escolha da ação em cada episódio depende unicamente do próprio episódio. Dessa forma, o agente não precisa pensar à frente. Já no ambiente sequencial, a decisão atual pode afetar as decisões futuras, como num jogo de xadrez, por exemplo;
- Estático vs. Dinâmico: Se o ambiente pode mudar enquanto um agente está deliberando, então dizemos que o ambiente é dinâmico; caso contrário, ele é estático e o agente não precisa continuar observando o ambiente enquanto está tomando uma decisão e não precisa se preocupar com o passar do tempo;
- Discreto vs. Contínuo: A distinção entre ambiente discreto e contínuo pode ser aplicada ao estado do ambiente, considerando o tempo, as percepções e ações do agente;
- Agente único vs. Multiagente: Um agente resolvendo palavras cruzadas sozinho está nitidamente num ambiente com agente único, enquanto que um agente jogando xadrez está em um ambiente com mais de um agente. Essa distinção de ambientes faz diferença na visão do agente, já que em um ambiente multiagente ele precisa considerar a presença dos outros agentes para maximizar sua medida de desempenho. Em um ambiente multiagente competitivo, para que um agente maximize seu desempenho, ele precisa minimizar a medida de performance dos agentes concorrentes. Isso acontece também em um ambiente multiagente cooperativo, onde o agente precisará se preocupar com o desempenho dos demais agentes para que todos possam atingir o objetivo em conjunto. Os problemas nos ambientes de agente único e multiagente são muitas vezes completamente diferentes. A comunicação, por exemplo, frequentemente emerge como um comportamento racional em sistemas multiagente. Em ambientes competitivos, o comportamento estocástico é racional, porque evita que sejam criadas armadilhas explorando a

previsibilidade. As características do agente e do ambiente devem ser consideradas no projeto de agentes em um Sistema Multiagente (SMA).

O ambiente SBS no qual o SMA opera possui as seguintes propriedades dentre as citadas acima: Completamente Observável, Determinístico, Sequencial, Dinâmico, Contínuo, Multiagente.

Sistemas multiagente é um subcampo relativamente pertinente na Ciência da computação. A área só começou a ser estudada nos anos 80 e só ganhou reconhecimento em meados dos anos 90. Mas, desde então, o interesse científico e industrial no campo cresceu. Este rápido crescimento tem sido impulsionado pela constatação de que agentes são um paradigma de software apropriado para explorar necessidades apresentadas pelas modernas plataformas de computação e informação: distribuídas, abertas, de grande volume e heterogêneas [WOOLDRIDGE, 2009].

Para Wooldridge (2009), um sistema multiagente consiste de um número de agentes que interagem uns com os outros, tipicamente através de troca de mensagens ou através de alguma infraestrutura de rede de computador. No caso mais geral, os agentes em um sistema multiagente estariam representando ou agindo em nome dos usuários ou proprietários com os mais diversos objetivos e motivações. A fim de interagir com sucesso, esses agentes possuem assim a capacidade de cooperar, coordenar e negociar uns com os outros, da mesma maneira que nós cooperamos, coordenarmos e negociamos com outras pessoas em nossas vidas diárias.

Segundo Sycara (1998), as características de SMA são tais que:

- Cada agente possui uma visão limitada;
- Não há controle global do sistema;
- Trabalham com dados descentralizados; e
- A computação é assíncrona, de maneira a permitir a comunicação entre agentes heterogêneos.

Para Wooldridge (2009) um SMA contém um número de agentes que interagem por comunicação, são capazes de agir em um ambiente, possuem diferentes esferas de influência, e são ligados uns aos outros por relações organizacionais, como é ilustrado na Figura 5. Neste caso, as esferas de influência representam cada antipadrão a ser detectado por cada agente em específico em uma relação organizacional comandada pelo agente *Root*. Cada agente possui

diferentes controles sobre diferentes partes do ambiente, podendo ou não existir sobreposição de nicho entre eles. O fato de essas esferas poderem coincidir faz surgir dependências entre os agentes. Os agentes também estão ligados por outros relacionamentos hierárquicos.

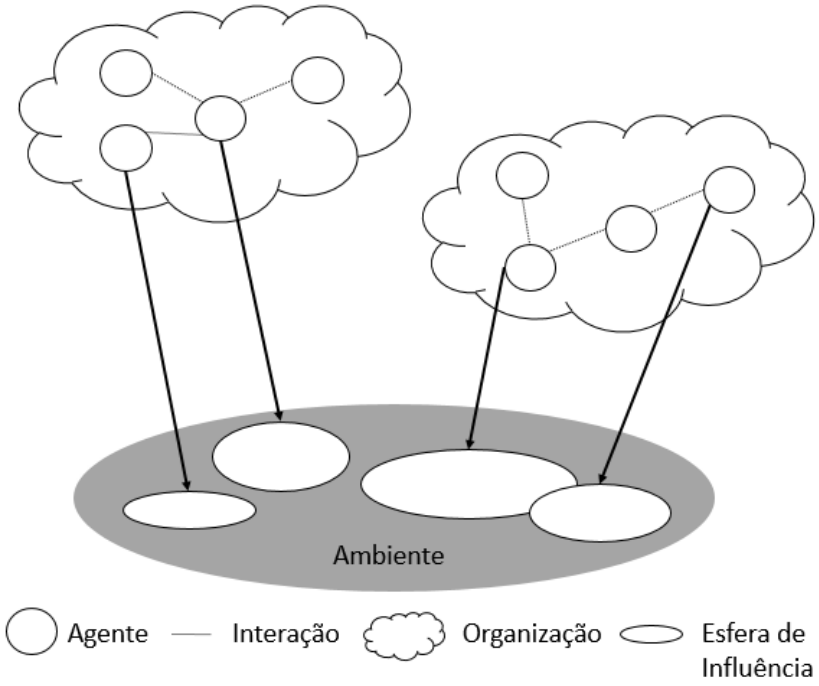

Figura 5 - Estrutura típica de um sistema multiagente.
Fonte: [WOOLDRIDGE (2009)] adaptado.

Segundo Bellifemine, Caire e Greenwood (2007), coordenação é um processo no qual agentes se comprometem em ajudar a garantir que uma comunidade de agentes individuais aja de maneira coerente. São várias as razões pelas quais o SMA necessita ser coordenado, tais como:

- Os objetivos dos agentes podem causar conflitos entre as suas ações;
- Os objetivos dos agentes podem ser interdependentes;
- Agentes podem ter diferentes capacidades e diferentes conhecimentos; e
- Metas globais do sistema podem ser alcançadas mais rapidamente se diferentes agentes agirem em cada subproblema.

2.4.1 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*), uma sociedade da IEEE (Instituto de Engenheiros Eletricistas e Eletrônicos).

Esta fundação surgiu como uma associação, sem fins lucrativos, de empresas e organizações com o intuito de 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.

A principal missão da FIPA é o trabalho de padronização destinado a facilitar a interoperabilidade entre sistemas de agentes, uma vez que, além da linguagem de comunicação, a FIPA especifica também quais os principais agentes envolvidos na gestão de um sistema e ainda o nível de transporte dos protocolos.

A FIPA definiu um conjunto de especificações para guiar a implementação de plataformas multiagentes (ambiente 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.

Na Figura 6 é ilustrada uma visão funcional das especificações da FIPA, que mostra os serviços obrigatórios e opcionais em uma plataforma.

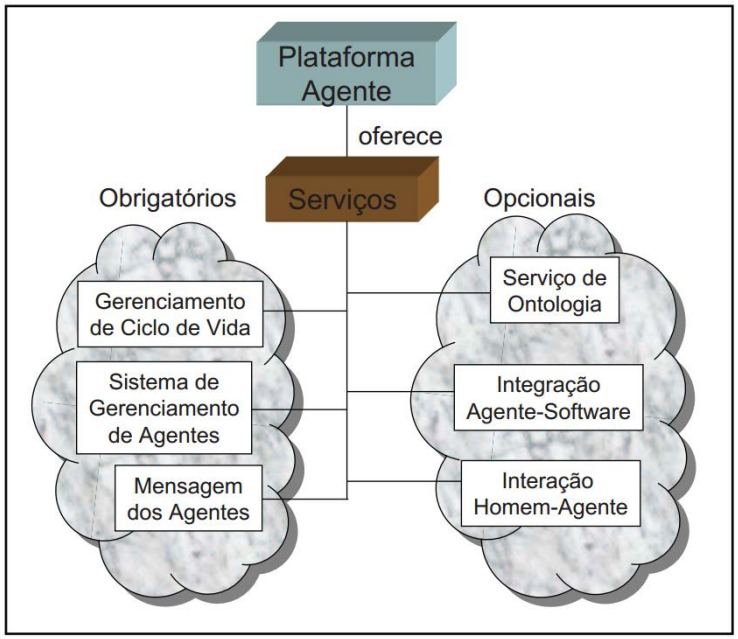

Figura 6 - Arquitetura abstrata para arquitetura concreta.

No desenvolvimento de uma arquitetura de agentes é necessária a descrição dos agentes, de seus serviços, do seu ciclo de vida e o transporte de mensagens entre eles. Dessa forma, a identificação dos agentes e suas relações fica mais fácil. Esses relacionamentos ajudam a identificar a capacidade de interoperabilidade entre os agentes da arquitetura aqui proposta.

O padrão FIFA inclui o serviço de gerenciamento do ciclo de vida, sendo este 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, Identificador do Agente (IdA). Este identificador deve ser imutável e único para permitir que um agente ou serviço possa contatar 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 da FIPA prevê a possibilidade de implementação de mecanismos de segurança no transporte de mensagens. Os agentes implementados se comunicam usando a Linguagem de Comunicação de Agentes *fipa-sl* para o envio de uma *string* de resposta ao *Root* caso tenha encontrado um antipadrão.

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 se comunicar entre si;
- **Estado de Espera:** é nesse estado que os agentes estão esperando a chegada de mensagens podendo realizar uma ação.

A mudança entre os estados possíveis de um agente é feita através da chamada de métodos, como podemos observar na Figura 7 (espera, criação e acordar).

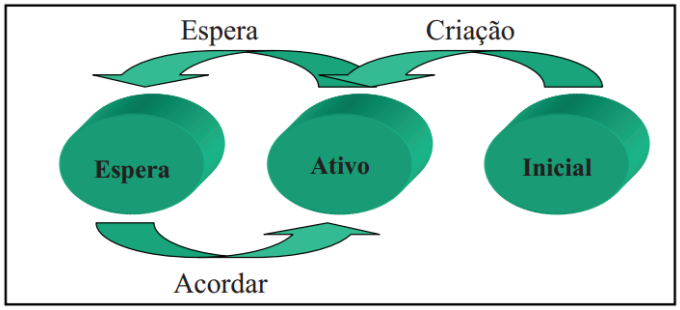

Figura 7 - Ciclo de vida de um agente.

Além dos serviços que são especificados pela FIPA, ela ainda sugere uma arquitetura de referência para plataformas de agentes. Esta arquitetura 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:

Dois aspectos fundamentais da comunicação entre agentes são:

- Estrutura da mensagem: A estrutura de uma mensagem (Figura 8) é formada por emissor, receptor e conteúdo. As mensagens contêm os nomes do emissor e do receptor, expressos como nomes de agentes. O conteúdo da mensagem é a própria mensagem a ser enviada.
- Transporte da mensagem: Quando uma mensagem é enviada é codificada em uma informação útil *payload* e incluída num formato chamado mensagem de transporte. A informação útil é codificada para o seu transporte pela rede. A mensagem de transporte é formada pela informação útil mais o envelope. O envelope. E por sua vez 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).

Figura 8 – Composição de uma mensagem.

2.4.2 Framework JADE

O JADE é um framework de *software* que provê funcionalidades na camada do *middleware*. Totalmente escrito em Java. É interoperável, uma vez que permite a comunicação entre agentes JADE com outros tipos de agentes fora da plataforma. Uniforme e portátil já que faz uso homogêneo de várias APIs independentes da versão Java e APIs para J2SE e J2ME. Aplicações JADE podem ser executadas em vários tipos de dispositivos desde servidores até celulares; dispositivos portáteis devem ter suporte a Java MIDP (Mobile Information Device Profile) 1.0 ou versões mais altas.

Sistemas baseados em JADE podem ser distribuídos através de máquinas (que nem sequer precisam compartilhar o mesmo sistema operacional) e a configuração pode ser controlada através de uma GUI (Graphical User Interface) remota. A configuração pode ser alterada, mesmo em tempo de execução por agentes que se deslocam de uma máquina para outra, quando necessário for [BELLIFEMINE (2007)].

JADE é um sistema distribuído, onde os agentes habitam e possuem como forma básica de comunicação as mensagens assíncronas; baseado no paradigma peer-to-peer. A estrutura dessas mensagens é baseada na linguagem ACL (Agent Communication Language) do padrão FIPA que contém campos como contexto da mensagem e o tempo limite de aguardo da resposta da mensagem. Os agentes são identificados por um nome global único, possuindo recursos para depuração e monitoramento, permitindo assim controlar plataformas JADE e seus componentes distribuídos. Integração com diversas tecnologias web como applets, web services, dentre outros. [TEIXEIRA (2010)].

JADE é composto de vários elementos. É composto de *containers* de agentes e que podem estar distribuídos na rede. O *container* é um ambiente que permite a habitação de agentes; é responsável por disponibilizar todos os recursos para que o agente execute as suas tarefas. O *container* pode abrigar de um a vários agentes e o conjunto de *containers* forma a plataforma JADE. Plataformas JADE obrigatoriamente devem conter o *Main Container*, responsável pelo gerenciamento de agentes e recursos da plataforma; este é composto pelos componentes AMS, DF e ACC além do CT (*Container Table*), registra os *containers* da plataforma e do GADT (*Global Agent Descriptor Table*), que registra todos os agentes presentes na plataforma, inclusive agentes estrangeiros e suas localizações; este *container* deve ser o primeiro a ser inicializado pois este é responsável pelo gerenciamento de toda a

plataforma. Só existe um *container* por host (BELLIFEMINE; CAIRE; GREENWOOD, 2007).

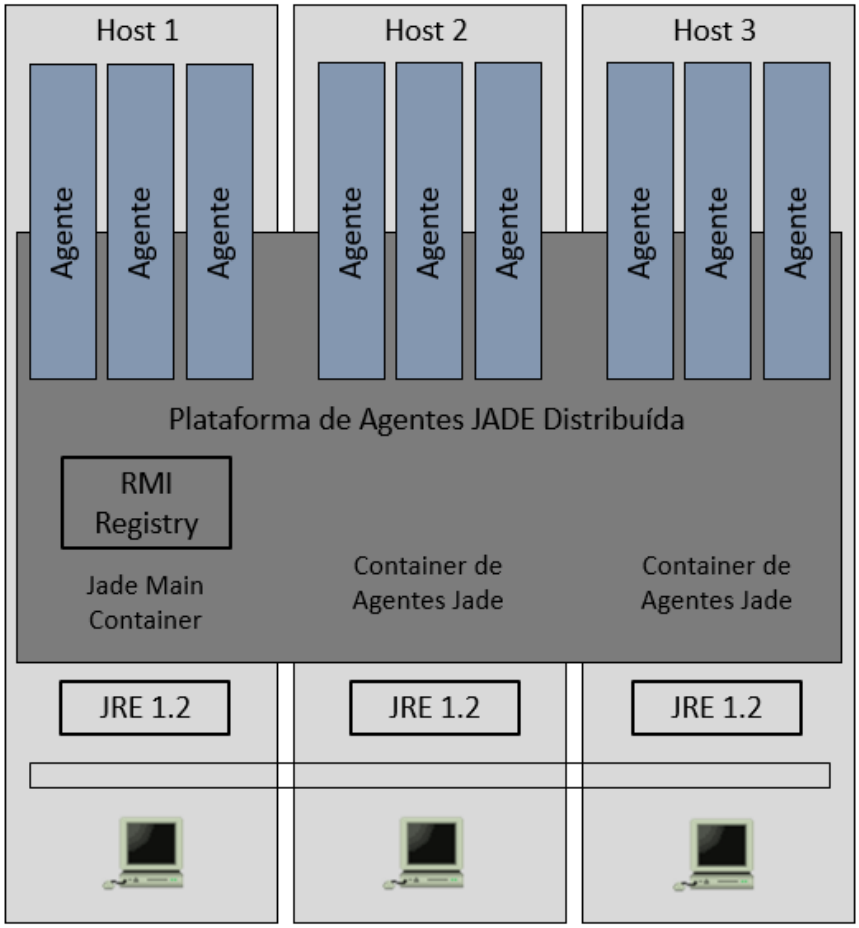

Figura 9 - Arquitetura do JADE.
Fonte: TEIXEIRA (2010) adaptado.

Na Figura 9 é ilustrada a arquitetura do JADE. Como podemos observar, a plataforma é o conjunto de *container*s, onde cada um é executado em um único *host*.

O *host* 1 possui o *Main Container* que gerencia a plataforma. Cada *container* está executando 3 agentes de software.

O agente JADE é uma instância da classe Agent da API do JADE. Possui uma *thread* interna para sua execução. Dispõem de comportamentos que são as tarefas executas pelos agentes e ciclo de vida. Além de poder migrar entre *containers* e plataformas. Na Figura 10 é ilustrada a arquitetura de um agente JADE; como pode ser observado, um agente JADE possui crenças e capacidades que são recursos implementados pelo programador; existe um gerenciador de ciclo de vida que é mostrado adiante; um escalonador de comportamento para escalonar quais comportamentos o agente deve executar e quando executar; e uma fila de mensagens que o agente recebe de outros agentes.

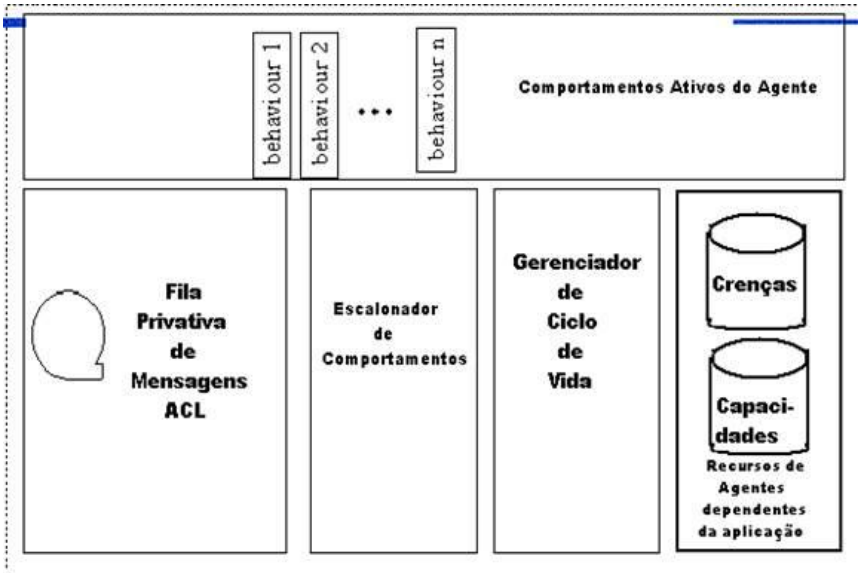

Figura 10 - Arquitetura de um agente JADE.

Fonte: SILVA (2003)

Agentes JADE possuem um ciclo de vida. Este ciclo de vida é ilustrado na Figura 11. Inicialmente o agente é criado e logo após inicializado. Ao ser invocado ele se torna ativo para execução de tarefas ou comportamentos. Do estado ativo ele pode ser suspenso e depois voltar ao estado de ativo. Outra ocasião é quando o agente está em modo de espera quando em busca de algum recurso que outro está executando e acordar quando este recurso for liberado. Outro estado é quando o agente está em trânsito se movimentando pela rede. Em qualquer estado que o agente esteja ele pode ser destruído.

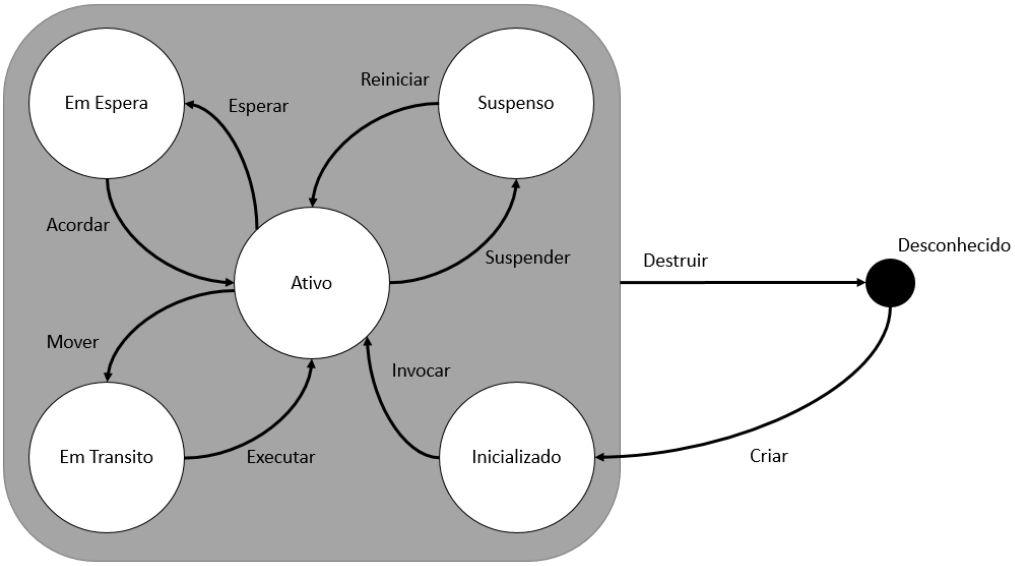

Figura 11 - Ciclo do agente JADE.

Fonte: SILVA (2003) adaptado.

3 TRABALHOS RELACIONADOS

Dentre os trabalhos relacionados: [PALMA (2015)] apresenta uma abstração unificando 3 tecnologias SBS: Web Services, SCA e REST. Usando essa abstração, descreve uma abordagem unificada, SODA (*Service Oriented Detection for Antipatterns*), apoiado por um framework, SOFA (*Service Oriented Framework for Antipatterns*), para avaliação da concepção e QoS de SBS. Usando a abordagem SODA, foram definidas regras de detecção para 31 antipadrões de serviços e 10 padrões de serviços independentes de suas tecnologias. Com base nas regras definidas para cada antipadrão foram automaticamente gerados (para serviços SCA e *Web Services*) ou implementamos (para REST) seus algoritmos de detecção. A aplicação e validação desses algoritmos em termos de precisão e recall ocorreu em (1) dois sistemas SCA, (2) em mais de 120 web services SOAP, e (3) um conjunto de 15 serviços *RESTful* amplamente utilizados, incluindo Facebook, Twitter e YouTube. Os resultados da detecção fornecem evidências da presença de antipadrões de serviços em SBS. A detecção relatada apresenta uma alta precisão e um recall de desempenho de detecção aceitável em termos de tempo de detecção. A abordagem SODA e a ferramenta subjacente podem ajudar os profissionais a avaliar o seu SBS, o que pode resultar em um SBS (1) com melhor qualidade de projeto e manutenção fácil e (2) com QoS melhoradas para os usuários finais.

[MOAH (2012)] aplica uma abordagem, apoiada por um framework, para especificar e detectar antipadrões em SBS. Usando essa abordagem, foram especificados 10 antipadrões bem conhecidos e comuns, incluindo *Bottleneck Service*, *Multi Service e Service Chain*, e gerados automaticamente seus algoritmos de detecção. Os algoritmos de detecção foram aplicados e validados em termos de precisão e *recall* em *Home-Automation*, um SBS desenvolvido de forma independente. Esta validação demonstra que essa abordagem permite a especificação e detecção de antipadrões SOA com precisão de mais de 90% e a *recall* de 100%.

[OUNI (2015)] propõe uma abordagem automatizada para a detecção de antipadrões de serviços da Web usando um Algoritmo Evolutivo Paralelo cooperativo (P-EA). A ideia é que vários métodos de detecção combinados e executados em paralelo durante um processo de otimização encontrem um consenso quanto à identificação de antipadrões em *Web Services*. São relatados os resultados de um estudo empírico usando oito tipos de antipadrões comuns de Web Services. Foram comparadas com a implementação da abordagem cooperativa P-EA com pesquisa aleatória, duas abordagens baseadas em população única e

uma técnica de detecção de última geração, não baseada em pesquisa heurística. A análise estatística dos resultados obtidos demonstra que a abordagem é eficiente na detecção de antipadrões, com uma pontuação de precisão de 89% e uma pontuação de *recall* de 93%.

[SHARMA (2014)] apresenta uma abordagem para detectar automaticamente antipadrões de desempenho importantes em um aplicativo, aproveitando a análise de código estático e informações sobre a implantação prospectiva dos componentes do aplicativo na Nuvem. Também mostra experimentalmente que esses antipadrões podem se tornar proeminentes e afetar o desempenho da aplicação se o aplicativo for migrado para a Nuvem. Os resultados mostram que o desempenho de partes da aplicação com esses antipadrões sofre significativamente e, portanto, a detecção desses antipadrões tem um significado abrangente no domínio do desenvolvimento de *software* para a Nuvem. A abordagem apresentada também foi implementada em uma ferramenta protótipo de avaliação de migração em Nuvem.

Os trabalhos citados apresentam soluções relacionadas a detecção de antipadrões e estão diretamente relacionados com a proposta desse trabalho que se diferencia dos demais por apresentar um SMA, permitindo que cada agente realize buscas por cada antipadrão em tempo de execução procurando soluções automáticas através de ações para problemas que estejam relacionados a QoS. Quando o SMA detecta um erro na implementação, o sistema informa uma sugestão de solução que possa auxiliar na tomada de decisão do administrador em corrigir o código do projeto.

Na Tabela 2 é mostrada a comparação entre trabalhos.

Tabela 2 – Comparação entre trabalhos.

Trabalho	Detecção Automática	Detecção em tempo de execução	Framework	SBS	Nuvem	Correção Automática e Sugestões
MOAH (2012)	*		*			
SHARMA (2014)	*				*	

PALMA (2015)	*		*			
OUNI (2015)	*					
Este Trabalho	*	*				*

4 ESPECIFICAÇÃO E IMPLEMENTAÇÃO DA ARQUITETURA

Nesta seção, são apresentadas as tecnologias e softwares utilizados para a implementação do SMA, a organização dos agentes e arquitetura do sistema e, por fim, as regras de especificação para a detecção dos antipadrões.

Como suporte ao método de detecção fez-se uso de um Sistema Multiagente cujo o funcionamento será descrito a seguir.

4.1 TECNOLOGIAS E SOFTWARES UTILIZADOS

Para implementação do Sistema Multiagente desse trabalho foi utilizada a linguagem de programação JAVA versão 1.8 com o auxílio do JADE (*Java Agent DEvelopment Framework*) versão 4.3.3, um framework para a implementação de sistemas multiagentes através de um *middleware* que segue as especificações de compilação FIPA (uma organização de padrões IEEE Computer Society que promove a tecnologia baseada em agentes e a interoperabilidade das suas normas com outras tecnologias).

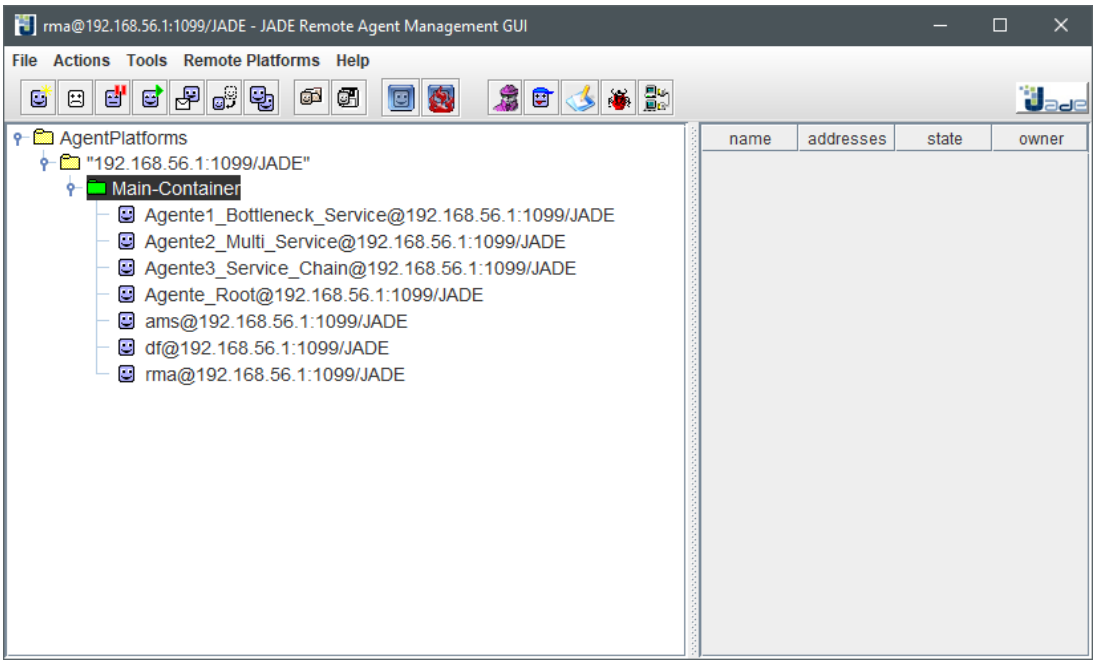

Figura 12 - Interface gráfica do Framework JADE.

Na Figura 12 é ilustrada a interface gráfica do Framework JADE, contendo o endereço e a porta de comunicação e o *container* principal, que comporta os agentes padrões da FIPA para a gerência da plataforma externa, tais como AMS, DF e o RMA e os agentes *Root*, *Agente1_Bottleneck_Service*, *Agente2_Multi_Service* e *Agente3_Service_Chain*.

A FIPA é uma associação internacional de várias companhias que trabalham para especificar tecnologias de agentes genéricas. Promove um conjunto de tecnologias para diferentes áreas de aplicação para criar sistemas complexos com grande interoperabilidade. A FIPA define um modelo de plataforma que permita a existência, operação e gerenciamento de agentes. Na Figura 13 é ilustrado o modelo de gerência e comunicação do SMA utilizando-se o padrão FIPA. O modelo é composto pelo AMS (*Agent Management System*), responsável por supervisionar a plataforma e gerenciar o ciclo de vida dos agentes da plataforma externa; DF (*Directory Facilitator*), responsável por registrar as solicitações de serviços; ACC (*Agent Communicattion Channel*), que gerencia a comunicação entre os agentes dentro e fora da plataforma; e a Plataforma Externa, que contém os agentes responsáveis pelo sensoriamento e atuação do SMA proposto neste trabalho em ambientes SBS. A ligação entre os agentes da plataforma externa exibe o papel de coordenação e submissão dos demais agentes para com o Agente *Root*.

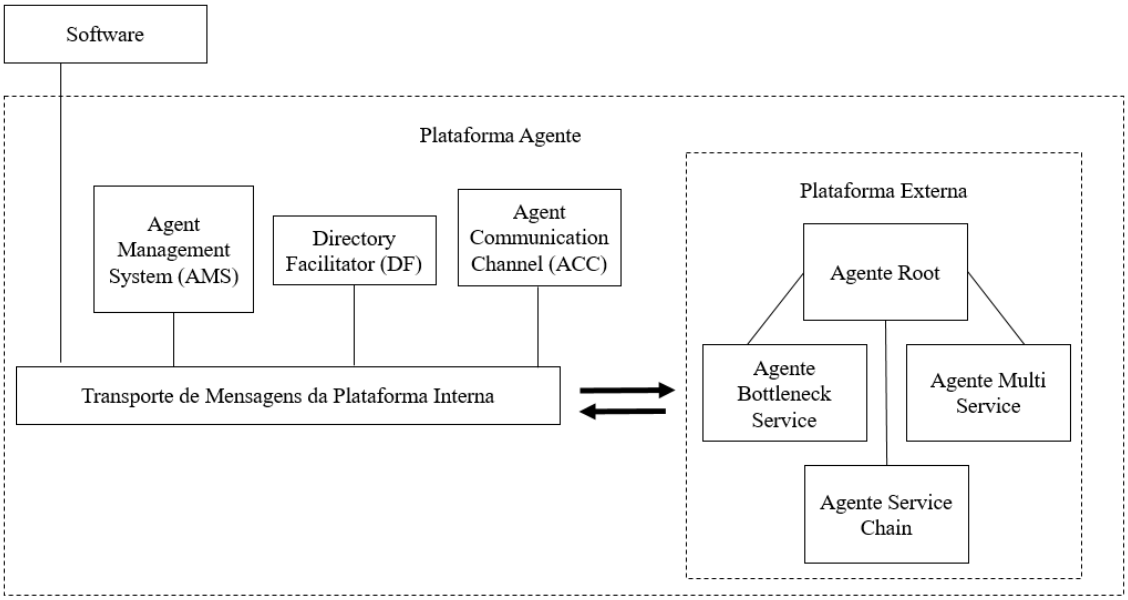

Figura 13 - Modelo de gerência e comunicação do SMA utilizando-se o padrão FIPA.

Agentes JADE possuem um ciclo de vida. Este ciclo de vida é ilustrado na Figura 11. Além do ciclo de vida de agente JADE existe também o ciclo de escalonamento de comportamentos, que corresponde como o agente JADE procede para executar as suas tarefas.

Os comportamentos do agente JADE são classificados em dois tipos: comportamentos ativos, tarefas que não foram concluídas e comportamentos bloqueados, que são tarefas que foram concluídas. No primeiro momento é analisado se o agente está inicializado, caso contrário ele é criado e em seguida inicializado. O Agente *Root* está sempre ativo e é responsável por ativar os demais quando há mudanças em relação ao tempo médio de resposta, e de suspendê-los quando suas tarefas forem encerradas.

Na Figura 14 é ilustrado, a partir de um fluxograma, o ciclo de comportamentos durante a execução do SMA. A cada nova solicitação o agente *Root* monitora os atrasos nas solicitações, verifica se existe variação considerável no tempo médio de resposta e analisa o contexto para encaminhar ao agente responsável pelo antipadrão que corrige ou informa ao administrador.

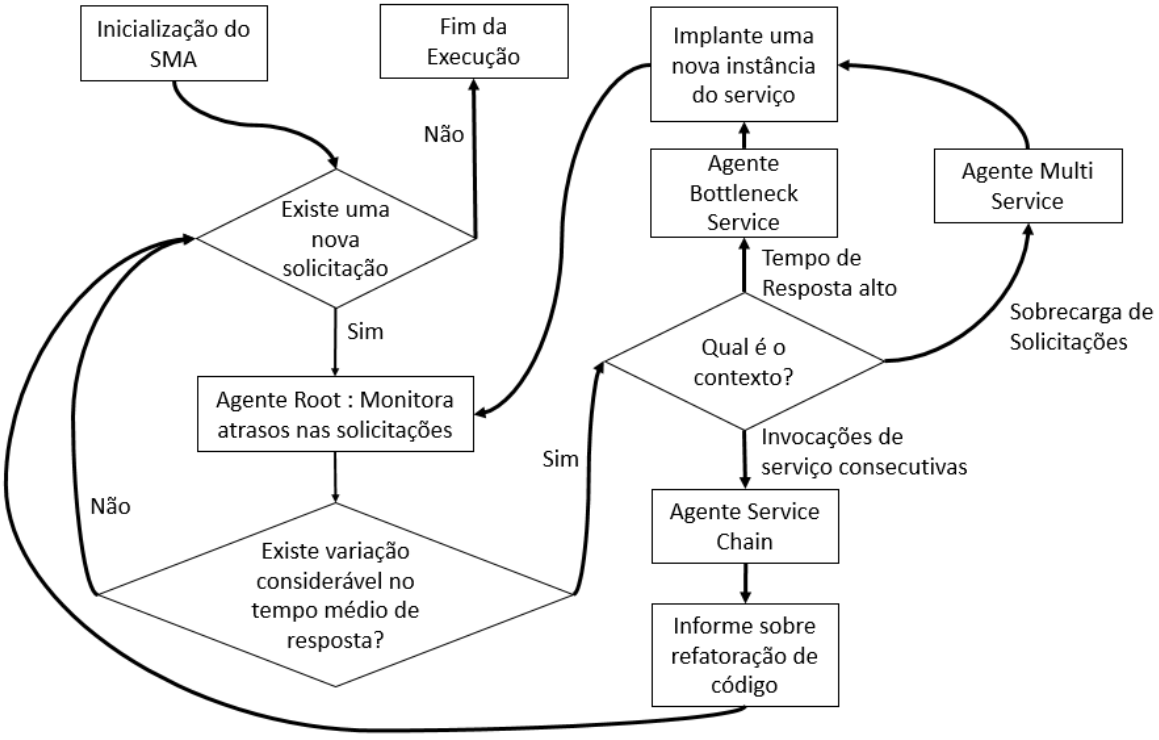

Figura 14 - Ciclo de comportamentos durante a execução do SMA.

Como servidor de aplicações foi utilizado o Oracle GlassFish Server (v 4.1.1), um servidor para desenvolvimento e implantação de aplicativos Java Platform, Enterprise Edition (plataforma Java EE) e tecnologias web baseadas na tecnologia Java. GlassFish Server fornece o seguinte: um núcleo leve e extensível baseado nos padrões da OSGi Alliance; um Console de Administração fácil de usar para configuração e gerenciamento; suporte para *clustering* de alta disponibilidade e balanceamento de carga [GlassFish Server].

4.2 ORGANIZAÇÃO DOS AGENTES E ARQUITETURA DO SISTEMA

Na Figura 15 mostra o diagrama de sequência, o qual indica a ordem das ações e a relação entre os objetos do projeto. O *Web Service* é gerenciado por um administrador que é responsável por operar o *hardware* bem como realizar a manutenção do *software* que fornecerá serviços aos clientes. O sistema multiagente é o *software* que busca por antipadrões durante a execução dos serviços do SBS.

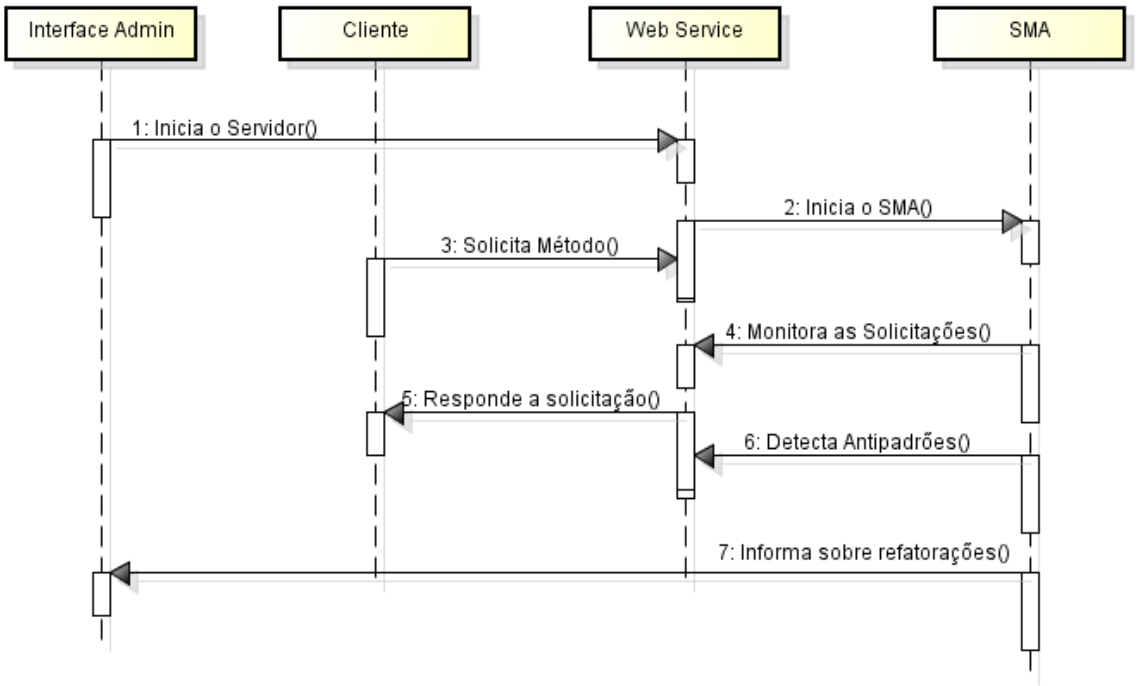

Figura 15 - Diagrama de Sequência: Comportamento dos Objetos do Sistema.

O SMA contém quatro agentes (Um agente mestre e três subordinados a ele, Figura 16) que interagem por comunicação e são capazes de agir em um ambiente de SBS atuando sobre cada antipadrão, e interagem com o agente *Root* pela relação organizacional mestre-escravo. O agente *Root* é o agente mestre que monitora as operações de serviço do SBS, calculando a média de tempo gasto no retorno da solicitação do cliente e, caso o tempo da operação do serviço atual seja maior que a média (do tempo das respostas das solicitações anteriores) calculada pelo agente *Root*, o mesmo envia uma mensagem para que os seus agentes subordinados procurem por antipadrões. Cada um dos agentes subordinados é responsável por detectar um antipadrão (cada agente é um *thread* que age de maneira independente na busca dos antipadrões agilizando a detecção em tempo de execução).

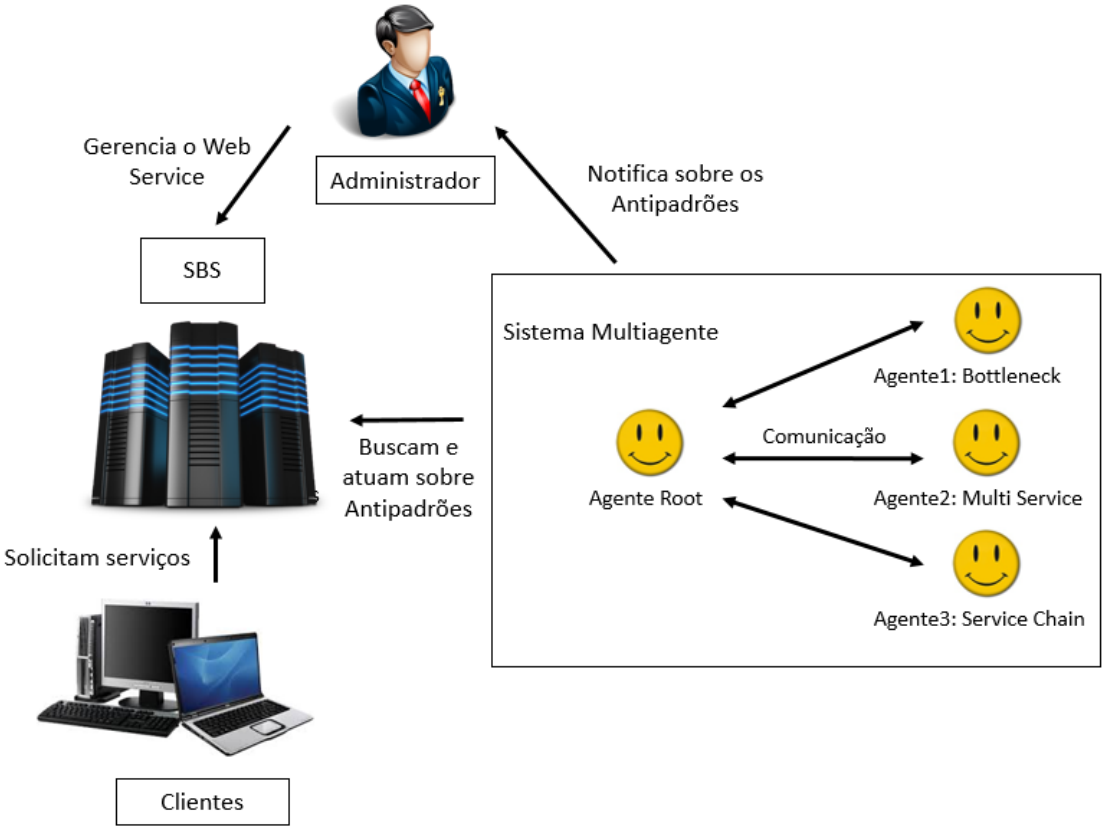

Figura 16 - Arquitetura do Sistema e seu escopo.

O Antipadrão *Bottleneck Service* é analisado pelo agente 1, de tal forma que sua detecção se dá pelo excesso no número de métodos solicitados em um determinado período de tempo, com base no histórico de solicitações armazenados pelo agente; Como solução, o agente redireciona as solicitações para uma nova instância do servidor, diminuindo assim a sobrecarga de solicitações. O Antipadrão *Multi Service* é analisado pelo agente 2 e sua detecção é devido um grande número de clientes solicitando os serviços, em um número maior do que o respondido em solicitações anteriores; Como solução, o agente sugere a mesma solução do *Bottleneck Service*. O antipadrão *Service Chain* é analisado pelo agente 3 e é detectado no caso de algum serviço recursivo não obter sua resposta em um tempo hábil, em certos casos até um *loop* infinito; Como solução, o agente sugere que o administrador revise a maneira como a recursividade dos métodos dos serviços será implementada a fim de diminuir o encadeamento de solicitações de serviço.

4.3 ESPECIFIÇÃO DOS ANTIPADRÕES

A seguir a sequência de passos adotada para a realização da metodologia abordada. O 1º passo é a seleção e descrição textual dos Antipadrões: nesta etapa foi realizado um estudo sobre quais antipadrões estão catalogados na literatura pela leitura e análise dos trabalhos relacionados citados no Capítulo 3, descrevendo para cada antipadrão: uma noção geral do problema; as causas primárias que levaram à noção geral; sintomas que descrevem como reconhecer a noção geral; as consequências da noção geral; e uma solução refatorada descrevendo como alterar o projeto (apresentada no início desta sessão). O 2º passo é o desenvolvimento das regras de detecção, que são os passos a serem tomados pelos agentes reativos simples na busca pelos antipadrões. Descritos a seguir:

Agente Bottleneck Service:

*Se: número_de_clientes_solicitando > média_de_clientes_ solicitanto_anteriormente E tempo_de_resposta > 2 * tempo_médio_solicitações_anteriores*

Então: Redirecionar_para_outra_instância_do_servidor;

Agente Multi Service:

*Se: número_de_solicitações_sendo_atendidas > número_médio_de_solicitações_ anteriores E tempo_de_resposta > 2 * tempo_médio_solicitações_anteriores*

Então: Redirecionar_para_outra_instância_do_servidor;

Agente Service Chain:

*Se: tempo_aguardo_> 2 * tempo_médio_solicitações_anteriores*

Então: Sugerir_refatoração_de_código_ao_admin;

A variável *tempo_aguardo* grava o tempo em milissegundos que o servidor está ocioso pela espera da resposta de um método encadeado a outro.

O 3º passo é a implementação das regras elaboradas, para que de fato o SMA possa ser capaz de detectar os antipadrões.

5 RESULTADOS E DISCUSSÃO

Com o intuito de verificar a eficácia do SMA em detectar os antipadrões, foi desenvolvida uma aplicação *Web Service* utilizando-se do protocolo SOAP. A aplicação é um sistema financeiro composto por operações tais como débito (representado pelo símbolo '-') e crédito (representado pelo símbolo '+'), correção (representado pelo símbolo '*') e conversão em cotas (representado pelo símbolo '!') sendo o mesmo dividido em 2 projetos, o primeiro executado do lado do servidor e o segundo do lado do cliente.

A aplicação do lado do cliente contém arquivos em XML com descrições para ter acesso aos métodos do servidor, e jsp e Java para entregar uma página *Web* ao cliente com funções de solicitar a operação desejada.

A aplicação do lado do servidor contém os métodos para requisitar os serviços em Java como é mostrado na Figura 17. A descrição de como o cliente pode acessar o método é feita em linguagem XML e está contida no arquivo WSDL (Web Services Description Language), como mostrado na Figura 18.

```
/**
 * Operação de Web service
 */
@WebMethod(operationName = "credito")
public int add(@WebParam(name = "i") int i, @WebParam(name = "j") int j) {
    //TODO write your implementation code here:


    return i+j;
}
```

Figura 17 - Método crédito implementado como um serviço.

```
▼<binding name="CalculatorWSPortBinding" type="tns:CalculatorWS">
   <soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="document"/>
  ▼<operation name="add">
     <soap:operation soapAction=""/>
    ▼<input>
       <soap:body use="literal"/>
     </input>
    ▼<output>
       <soap:body use="literal"/>
     </output>
   </operation>
```

Figura 18 - Descrição de uma operação no arquivo XML.

Na Figura 19 é mostrada a *Graphical User Interface* (GUI) que é exibida ao administrador durante a execução do SBS, neste caso, o CalculatorApp desenvolvido. A GUI informa se algum antipadrão foi detectado pelo SMA, quantas quedas de QoS ocorreram desde a identificação do antipadrão e um campo de texto contendo a análise do SMA durante o tempo de execução do SBS. Neste mesmo campo de texto são exibidos o tempo em milissegundos que uma operação demorou para enviar a sua resposta ao cliente, o número de quantas operações desse mesmo método já foram chamados, a média de tempo que o SBS está gastando para responder ao cliente uma determinada operação e em quanto o valor da chamada atual variou em relação à média das chamadas anteriores em porcentagem.

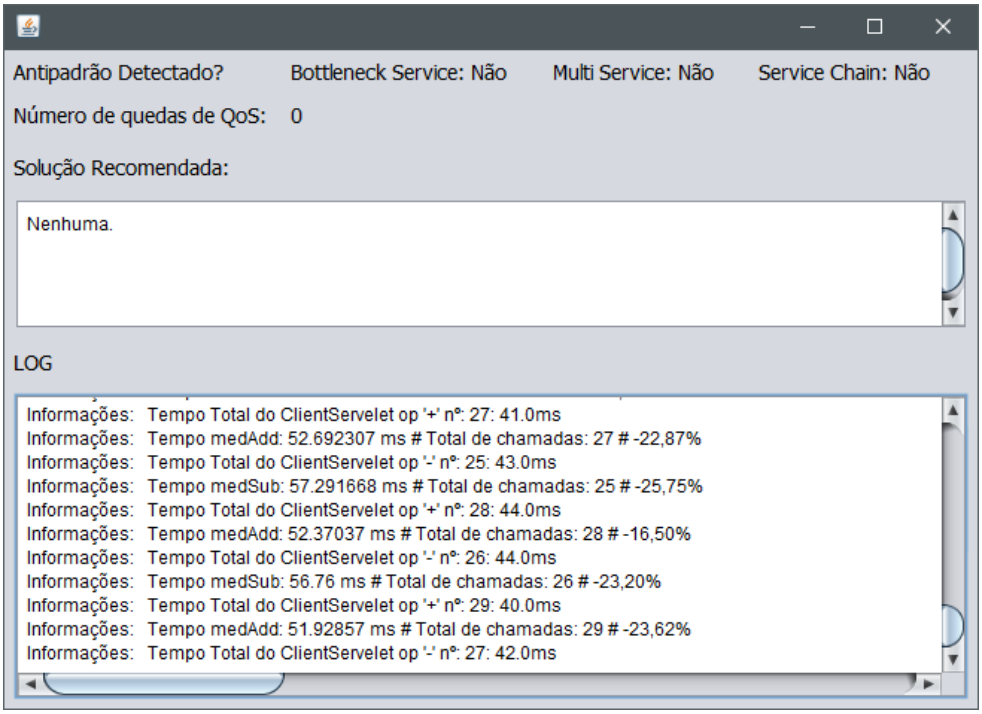

Figura 19 - Execução do SBS com 2 clientes.

O teste de detecção dos antipadrões foi executado no laboratório de informática MCC (Mestrado em Ciência da Computação) da Universidade do Estado do Rio Grande do Norte utilizando-se uma rede cabeada de 10 Mbps, um notebook rodando a aplicação do lado do servidor com a seguinte configuração: Processador Intel Core i7 quad-core de 2.0GHz, memória RAM de 8GB e Sistema Operacional Windows 10. Como clientes foram utilizados 5 computadores de mesma configuração: Processador AMD Athlon dual-core de 3.4GHz, memória RAM de 2GB e Sistema Operacional Windows 7.

Cada cliente solicita duas operações do sistema financeiro e, em seguida, obtém uma resposta do servidor já solicita a seguinte, uma forma de automatizar o processo e provocar o surgimento do antipadrão.

De início apenas dois computadores se conectaram ao servidor e o SBS suportou as solicitações sem apresentar nenhuma anomalia, como podemos observar na Figura 19. Posterior à chamada 1028 (Figura 20) foram conectadas 5 máquinas ao servidor, o que logo apresentou uma instabilidade no tempo de resposta chegando a demorar mais do que o dobro do tempo (representado pelo 100% em amarelo na figura) em média necessário para responder a solicitação. A porcentagem expressa que o SBS está respondendo somente uma operação enquanto poderia estar respondendo ao menos duas operações. Devido ao alto tempo de resposta e o número de clientes conectados, o SMA logo detectou o antipadrão *Bottleneck Service* e o Agente1 agiu dividindo a carga com uma outra instância do servidor, o que veio a normalizar o tráfego.

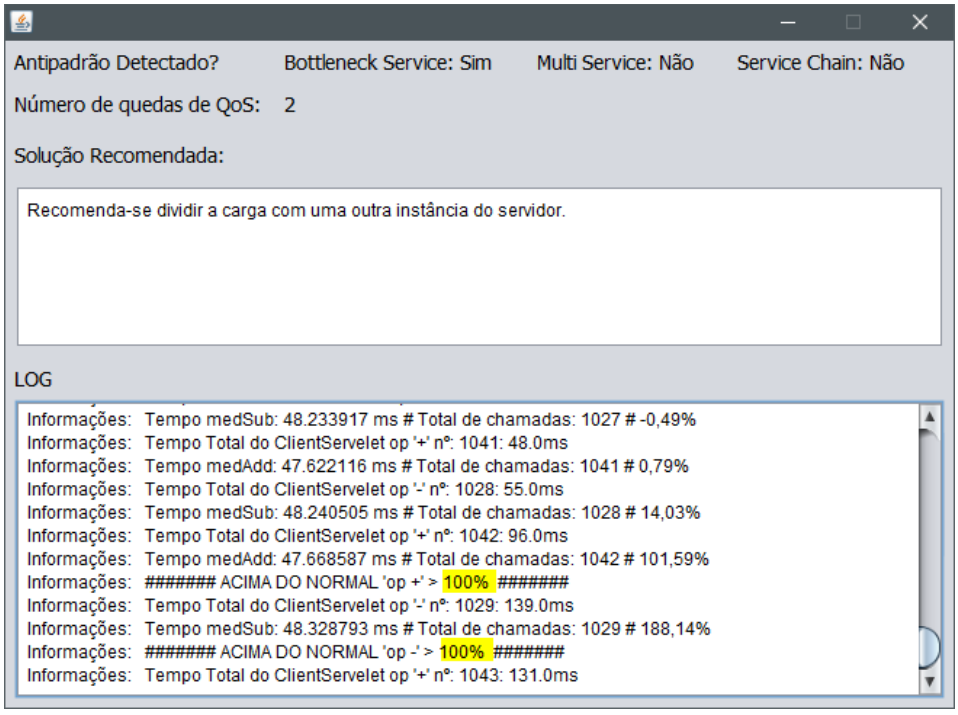

Figura 20 - Detecção do antipadrão Bottleneck Service com 5 clientes.

No Gráfico 1 e no Gráfico 2 são ilustrados dois trechos dos resultados obtidos pelo monitoramento do SMA. O Gráfico 1 contém os dados das chamadas iniciais dos clientes envolvendo as operações de débito e crédito com dois clientes solicitando por esses métodos. Nessa configuração o WS foi capaz de responder às solicitações em menos de 50 milissegundos e o tempo das últimas operações não superou o tempo médio das operações

anteriores, o que indica um baixo impacto na espera pela resposta dos métodos solicitados. No Gráfico 2 é ilustrado um outro cenário no qual foram conectados 5 clientes, onde é possível observar a alteração no valor do tempo atual superando o valor do tempo médio, indicando um acréscimo de tempo significante à média e um valor de tempo de resposta bem acima de 50 milissegundos, superior em mais de duas vezes à média tolerada. No gráfico 2 o tempo da última operação superou o tempo médio das operações anteriores indicando o surgimento do antipadrão *Bottleneck Service* e degradando a QoS.

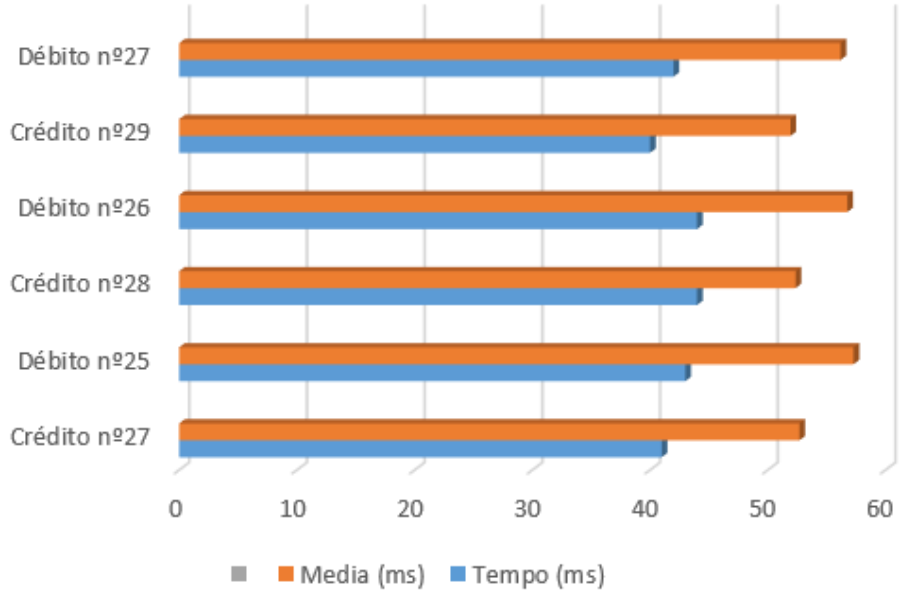

Gráfico 1 - Tempo médio das chamadas anteriores e o tempo da chamada atual (2 clientes).

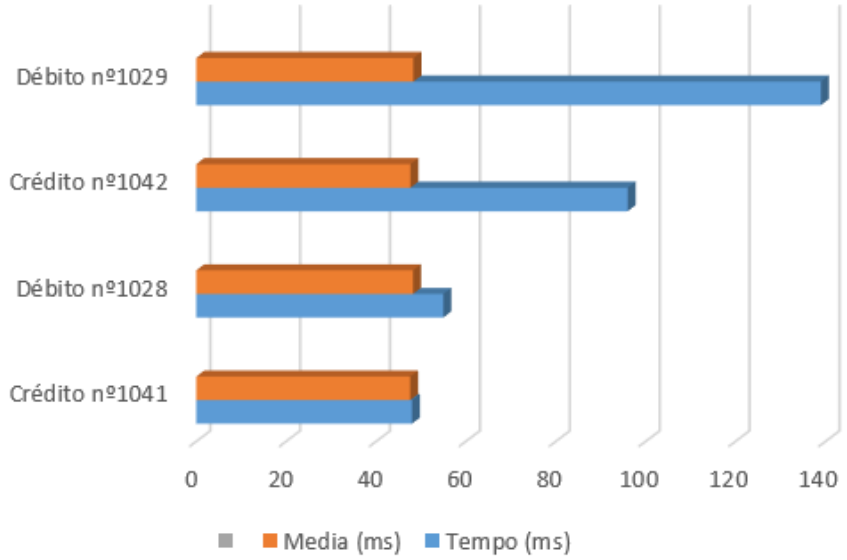

Gráfico 2 - Tempo médio das chamadas anteriores e o tempo da chamada atual (5 clientes).

O teste de detecção do antipadrão *Multi Service* foi realizado verificando se a adição de mais uma operação entre as solicitações poderia prejudicar a QoS. De início apenas os métodos Crédito e Débito foram solicitados e o SBS conseguiu responder sem quedas de desempenho. Após 100 operações efetuadas, foi dado início ao método correção, favorecendo o surgimento do *Multi Service*, 18 chamadas após o seu início, como é mostrado na Figura 21 e no Gráfico 3 também.

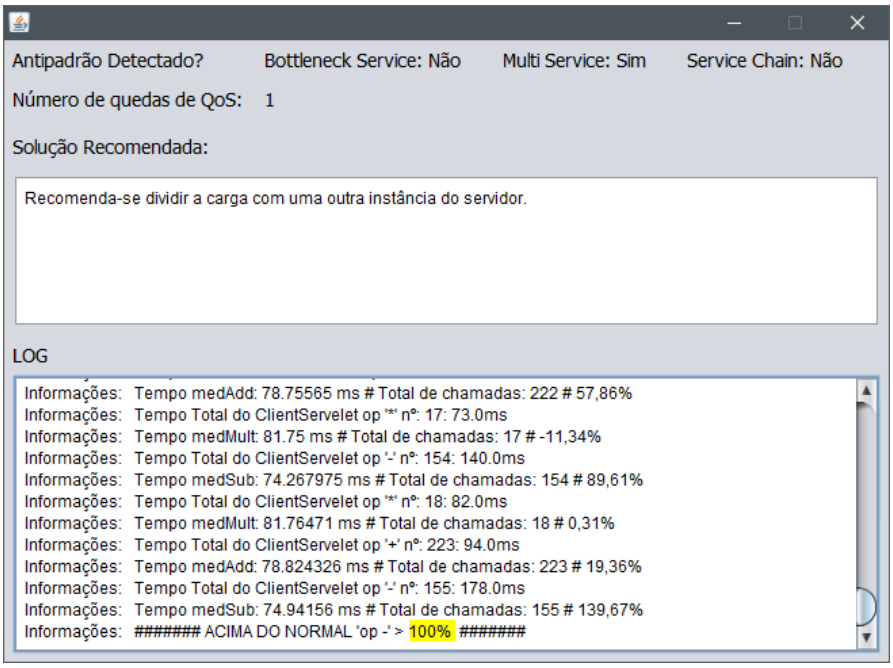

Figura 21 - Detecção do antipadrão Multi Service com 3 métodos solicitados.

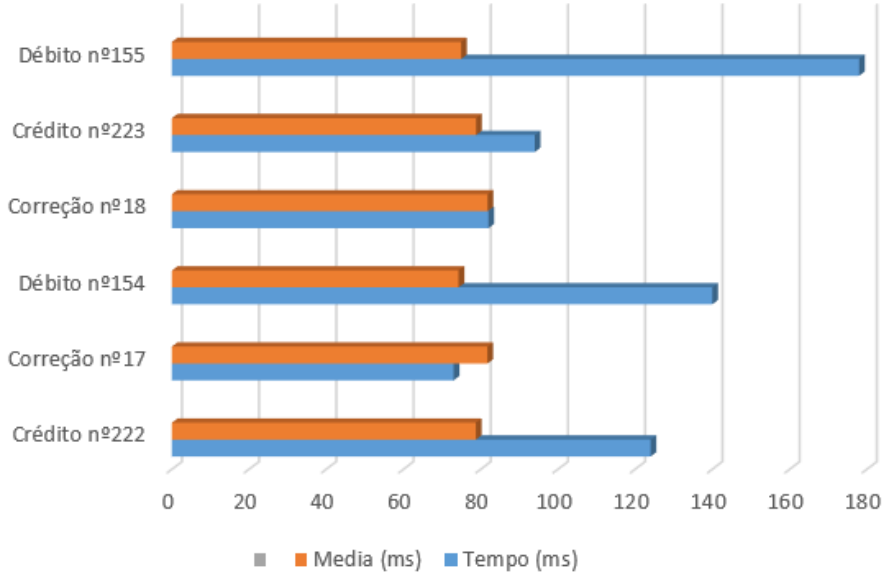

Gráfico 3 - Tempo médio das chamadas anteriores e o tempo da chamada atual (3 métodos solicitados).

Para influenciar no surgimento do *Service Chain*, Figura 23, o código do lado do servidor foi alterado para que a cada vez que uma operação de Crédito ou Débito se realizasse, seus parâmetros passassem primeiro pelo método de correção, como é mostrado na Figura 22.

```
/**
 * Operação de Web service
 */
@WebMethod(operationName = "credito")
public int add(@WebParam(name = "i") int i, @WebParam(name = "j") int j) {
    //TODO write your implementation code here:


    return conv (i)+ conv(j);
}

@WebMethod(operationName = "correcao")
public int conv(@WebParam(name = "k") int k) {
    int fator_correcao = 3;
    //TODO write your implementation code here:


    return k * fator_correcao;
}
```

Figura 22 – Alteração no código para o encadeamento de serviços.

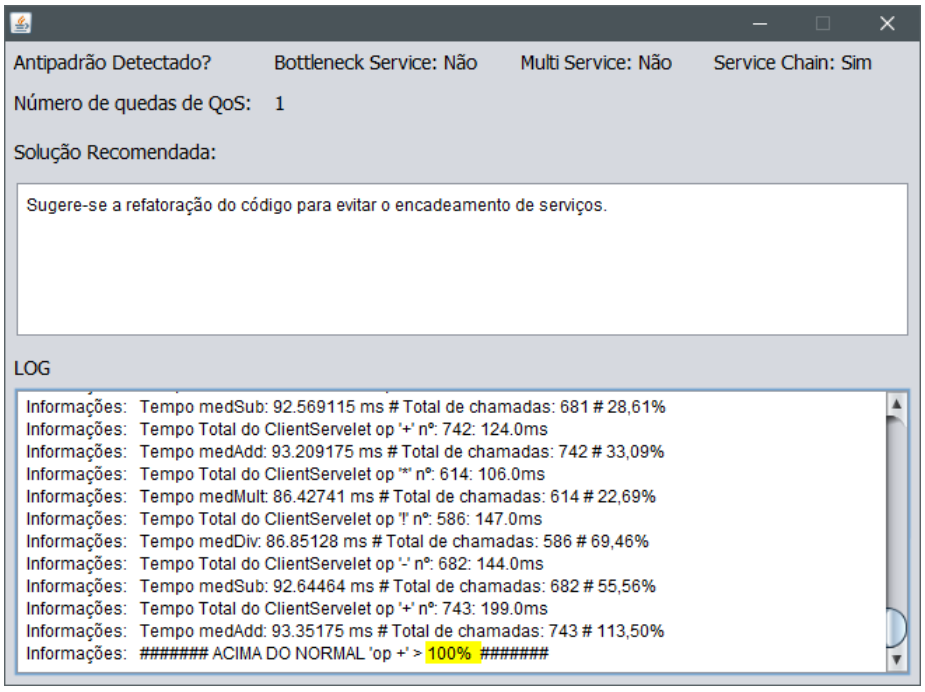

Figura 23 - Detecção do antipadrão Service Chain com os métodos encadeados.

No gráfico 4 é mostrado a interferência que as chamadas encadeadas do método correção causam no atraso de resposta sobre o método crédito.

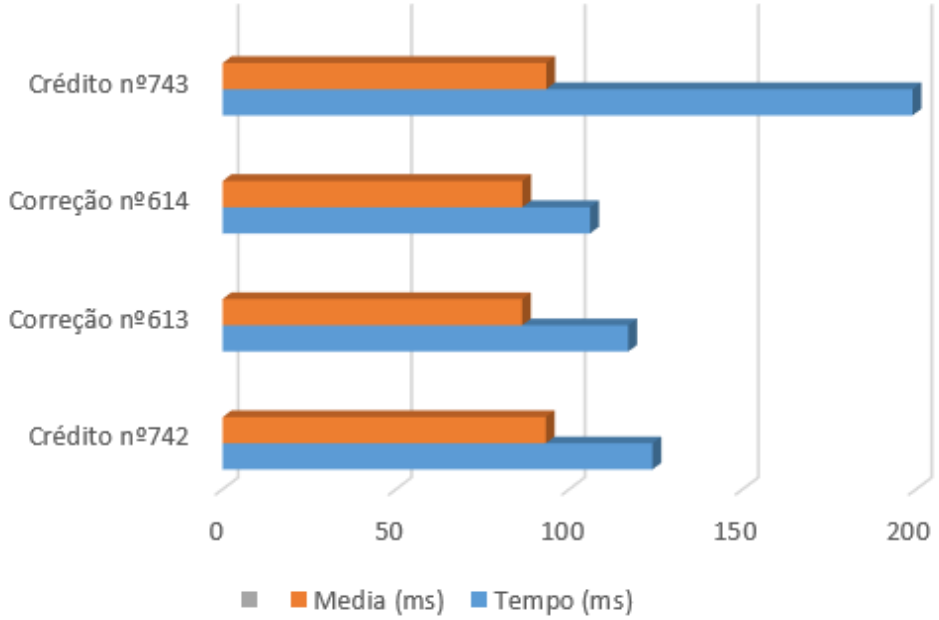

Gráfico 4 - Tempo médio das chamadas anteriores e o tempo da chamada atual (serviços encadeados).

Os resultados mostram que, nos piores casos, o surgimento de antipadrões chegou a extrapolar em até o dobro de tempo de resposta dos serviços em um *Web Service*. O SMA foi capaz de detectar, em tempo de execução, os 3 antipadrões estudados no trabalho. Quando adotadas, as soluções propostas pelo SMA também normalizaram o tempo de resposta.

6 CONCLUSÃO E TRABALHOS FUTUROS

Este trabalho apresentou o desenvolvimento de um Sistema Multiagente para detecção dinâmica de antipadrões em Sistemas Baseados em Serviços, apresentando conceitos que conduziram a pesquisa, como Arquitetura Orientada a Serviço (SOA), Sistemas Baseados em Serviços (SBS), Antipadrões e Sistemas Multiagente (SMA).

O objetivo do SMA desenvolvido é realizar a detecção automática e dinâmica (em tempo de execução) de antipadrões em Sistemas Baseados em Serviços. Usando essa abordagem é possível identificar, solucionar problemas de performance causados por antipadrões que venham a degradar a QoS e informar ao administrador do sistema sobre refatorações de códigos necessárias. Para atingir a meta proposta foram analisados o projeto de SBS; compreendidos e aplicados os paradigmas de antipadrões e seus impactos na sua utilização; e o estudo do *framework* JADE para o desenvolvimento da plataforma SMA que suporta os métodos de detecção.

Como contribuição, este trabalho apresenta a validação na capacidade do SMA em detectar dinamicamente 3 tipos de antipadrões e corrigir os problemas de performance que venham a degradar a QoS, bem como, uma solução ao administrador de como combater as anomalias detectadas a nível de código. Através dessa validação foi possível observar o quão significante é o impacto dos antipadrões para a QoS em um *Web Service*, o qual chegou a extrapolar em até o dobro de tempo de resposta dos serviços.

Como perspectivas futuras, tem-se:

- A realização de testes com outros SBS além de Web Service;
- Adição de mais antipadrões ao catálogo de detecção do SMA;
- Desenvolvimento de um framework que forneça suporte aos desenvolvedores que visem desenvolver suas aplicações com a ajuda do SMA desenvolvido neste trabalho.

REFERÊNCIAS

ALONSO, G. and CASATI, F. and KUNO, H. and MACHIRAJU, V. (2003). Web Services: Concepts, Architectures and Applications. Springer, Data-Centric Systems and Applications. 2003

BATISTA, A. F. DE M. Desenvolvendo Sistemas Multiagentes na plataforma JADE, 2008.

BELLIFEMINE, F.; CAIRE, G.; GREENWOOD, D. Developing multi-agent systems with JADE. United Kingdom: Jhon Wiley & Sons, 2007.

BENNETT, K., H. and RAJLICH, T., V. Software Maintenance and Evolution: A Roadmap. *Proceedings of the Conference on The Future of Software Engineering*. ACM, New York, NY, USA, ICSE '00, 73–87, 2000.

BUI, T.; LEE, J. An agent-based framework for building decision support systems. *Decis. Support Syst.*, Elsevier Science Publishers B. V., Amsterdam, The Netherlands, The Netherlands, v. 25, n. 3, p. 225–237, abr. 1999.

CHAUHAN, D. *JAFMAS: A Java-based Agent Framwork for Multiagent Systems Development and Implementation*. Tese (Doutorado) — University of Cincinnat, Cincinnat, 1997.

CHAPPELL D. *Introducing SCA*. Chappell & Associates, USA. 2007.

CHAMPEAUX, D., D. and LEA, D. and FAURE, P. Object-oriented System Development. Lectures in Mathematics Eth Zurich. Addison-Wesley. 1993.

CHISTENSEN, E. and CUERBA, F. and MEREDITH, G. and WEERAWARANA, S. Web Services Description Language (WSDL) 1.1. Rapport technique, W3C. 2011.

DUDNEY, B. and ASBURY, S. and KROZAK, J. K. and WITTKOPF, K. *J2EE AntiPatterns*. John Wiley and Sons. 2003

ERL, Thomas. Service-Oriented Architecture: A Field Guide to Integrating XML and Web Services. Prentice Hall PTR, Upper Saddle River, NJ, USA. 2004

ERL Thomas. *Service-Oriented Architecture: Concepts, Technology, and Design*. Prentice Hall PTR. 2005.

ERL, Thomas. *SOA Principles of Service Design*. Prentice Hall PTR, Upper Saddle River, NJ, USA. 2007

ERL Thomas. *SOA Design Patterns*. Prentice Hall PTR. 2009.

FIPA. FIPA Agent Communication Language Overview, Foundation for Intelligent Physical Agents. [S.l.], 1999. Disponível em: <Disponível em http://www.fipa.org [Consultado em 29/08/2016]>.

HEFFNER, R. and SCHWABER, C. and BROWNE, J. and SHEEDY, T. and STONE, J. and LEGANZA, G. Planned SOA Usage Grows Faster Than Actual SOA Usage. *Business Data Services North America, Europe, And Asia Pacific*. Forrester. 2007.

ISO/IEC/IEEE (2010). Systems and Software Engineering – Vocabulary. ISO/IEC/IEEE 24765:2010(E), 1–418.

MOAH, N.; PALMA, F.; NAYROLLES, M.; CONSEIL, B., J.; YANNGAEL, G. et al.. Specification and Detection of SOA Antipatterns. Chengfei Liu, Heiko Ludwig, Farouk Toumani. International Conference on Service Oriented Computing, Nov 2012, Shanghai, China. 2012.

NEWCOMER, ERIC and LOMOW, G. Understanding SOA with Web Services (Independent Technology Guides). Addison-Wesley Professional. 2004.

OUNI A, KESSENTINI M, INOUE K, Ó CINÉIDE M,. Search-based Web Service Antipatterns Detection. IEEE TRANSACTIONS ON SERVICES COMPUTING 2015.

PALMA, F. and NAYROLLES, M. and MOHA, N. and GUÉHÉNEUC, Y. and BAUDRY, B. and JÉZÉQUEL, J. SOA Antipatterns: An Approach for their Specification and Detection. *International Journal of Cooperative Information Systems,* 2013. *22* (04).

PALMA, F. Unifying Service Oriented Technologies for the Specification and Detection of their Antipatterns. 157 f. Tese. 2015.

PAPAZOGLOU, M., P. and TRAVERSO, P., T. and DUSTDAR, S. and LEYMANN, F. Service-oriented Computing. Communications of the ACM, 46, 25–28. 2003

ROSEN, M. and LUBLINSKY, B. and SMITH, K., T. and BALCER, M., J. (2008). *Applied SOA: Service-Oriented Architecture and Design Strategies*. Wiley.

REIS, L. P. Coordenação em Sistemas Multi-Agente: Aplicações na Gestão Universitária e Futebol Robótico. Dissertação (Mestrado), 2003.

RUSSELL, S.; NORVING, P. *Artificial Intelligence: A Modern Approach*. 3. ed. [S.l.]: Prentice Hall, 2010. ISBN 0136042597.

SHARMA, V., S.; ANWER, S. Performance Antipatterns: Detection and Evaluation of their Effects in the Cloud. IEEE International Conference on Services Computing. 2014.

SILVA, L. A. DE M. E. Estudo e desenvolvimento de sistemas multiagentes usando JADE: Java Agent DEvelopment framework. Monografia-Fortaleza: Universidade de Fortaleza, 2003.

SPANOUDAKIS, G. and MAHBUB, K.(2004). Requirements Monitoring for Servicebased Systems: Towards a Framework based on Event Calculus. *Automated Software Engineering, 2004. Proceedings. 19th International Conference on*. 379–384.

SYCARA, K. P. Multiagent systems. *AI Magazine*, v. 19, n. 2, p. 72–92, 1998.

TEIXEIRA, F. V. JADE - Java Agent DEvelopment framework, 2010.

TURNER, M. and BUDGEN, D. and BRERETON, P. Turning software into a service. *Computer*, *36* (10), 2003, 38–44.

WOOLDRIDGE, M. *An Introduction to MultiAgent Systems*. 2. ed. Chichester, England: John Wiley and Sons, LTD, 2009.

WOODS, D. and MATTERN, T. *Enterprise SOA: Designing IT for Business Innovation*. O'Reilly Media. 2006.