UNIVERSIDADE FEDERAL DE JUIZ DE FORA
INSTITUTO DE CIÊNCIAS EXATAS
DEPARTAMENTO DE CIÊNCIA DA COMPUTAÇÃO

USO DE AGENTES DE SOFTWARE PARA IDENTIFICAÇÃO E RESOLUÇÃO DE FALHAS: UM ESTUDO DE CASO NA PLATAFORMA INTEGRA

Thiago Baesso Procaci

JUIZ DE FORA
JULHO, 2008

USO DE AGENTES DE SOFTWARE PARA IDENTIFICAÇÃO E RESOLUÇÃO DE FALHAS: UM ESTUDO DE CASO NA PLATAFORMA INTEGRA

Thiago Baesso Procaci

Universidade Federal de Juiz de Fora

Instituto de Ciências Exatas
Departamento de Ciência da Computação
Bacharel em Ciência da Computação

Orientadora: Profa. Fernanda Cláudia Alves Campos

JUIZ DE FORA

JULHO, 2008

USO DE AGENTES DE SOFTWARE PARA IDENTIFICAÇÃO E RESOLUÇÃO DE FALHAS: UM ESTUDO DE CASO NA PLATAFORMA INTEGRA

Thiago Baesso Procaci

MONOGRAFIA SUBMETIDA AO CORPO DOCENTE DO DEPARTAMENTO DE CIÊNCIA DA COMPUTAÇÃO DO INSTITUTO DE CIÊNCIAS EXATAS DA UNIVERSIDADE DE JUIZ DE FORA, COMO PARTE INTEGRANTE DOS REQUISITOS NECESSÁRIOS PARA OBTENÇÃO DO GRAU DE BACHAREL EM CIÊNCIA DA COMPUTAÇÃO.

Aprovada pela banca constituída pelos seguintes professores:

Profa. Fernanda Cláudia Alves Campos – orientadora
D. Sc. em Engenharia da Sistemas, COPPE/UFRJ, 1999

Profa. Regina Maria Maciel Braga Villela
D. Sc. em Engenharia da Sistemas, COPPE/UFRJ, 2000

Prof. Rubens de Oliveira
D. Sc. em Engenharia Civil, PUC-RJ, 1994

JUIZ DE FORA, MG - BRASIL
JULHO, 2008

Agradecimentos

À minha família pelo apoio incondicional.

Aos amigos que estiveram presentes em minha vida.

À professora Fernanda pelos bons ensinamentos.

À todos os demais professores que de alguma forma contribuíram para minha formação.

Sumário

Lista de Figuras

Resumo

Esse trabalho descreve a concepção e o desenvolvimento do módulo *Agent* da Plataforma Integra. O módulo proposto consiste na construção de agentes de software que atuaram na Plataforma Integra como mecanismo identificação e resolução de falhas sistêmicas. O trabalho discorre sobre toda a base teórica que guiou o desenvolvimento do módulo, além de, uma descrição detalhada sobre a Plataforma Integra que é ambiente onde o módulo atua.

O módulo *Agent*, assim como a Plataforma Integra, foi desenvolvido em Java e envolve conceitos tais como programação orientada a agentes, padrões de projetos, arquitetura de software, *Web Services* e sistemas distribuídos.

Palavras-chave

Agentes de Software, Sistemas Multi-Agentes, Padrões de Projetos, Arquitetura de Software, Tolerância a Falhas.

1. Introdução

1.1. Motivação

No mundo contemporâneo a sociedade científica e as empresas necessitam cada vez mais de softwares que descrevam soluções computacionais robustas e sofisticadas para seus problemas. O desenvolvimento e a manutenção dessas soluções implica em adicionar complexidades a todas as etapas do desenvolvimento de software.

Porém, a construção desses softwares envolve alguns desafios dentro da Ciência da Computação e, muitas propostas de soluções passam pelo conceito de agentes de software. Entre esses desafios pode-se citar: a análise de grandes massas de dados, buscas de conteúdos que trazem resultados relevantes, simulações matemáticas de crescimentos populacionais, simulações de comportamento, entre outros.

De maneira geral, agentes podem ser vistos como softwares que tratam um determinado problema de maneira autônoma, fornecendo uma solução para o mesmo sem que seja necessária intervenção humana. O conceito de aplicações de resolvam problemas de forma independente e sem influências externas tem despertado grande interesse nas mais diversas áreas da comunidade científica.

1.2. Objetivos

Este trabalho tem como objetivo apresentar conceitos relacionados a agentes de software e descrever uma aplicação dos mesmos em um estudo de caso. Mais especificamente, serão apresentados os agentes desenvolvidos para a Plataforma Integra, os detalhes técnicos da implementação tanto da Plataforma quanto dos agentes. No estudo de caso será proposto um módulo adicional compostos de agentes que servirão como mecanismo de tolerância a falhas.

1.3. Organização do Trabalho

Esta monografia está dividida em outros cinco capítulos, além deste primeiro que introduz o tema e apresenta como o trabalho está organizado. O capítulo 2 aborda conceitos de agentes de software, bem como suas definições e classificações. No capítulo 3 tem-se uma descrição geral da Plataforma Integra, abordando seu contexto e detalhes relevantes de sua implementação. No capítulo 4 é realizado o estudo de caso que trata dos agentes desenvolvidos para a Plataforma

Integra como mecanismo de tolerância a falhas. E, por fim, no capítulo 5 são feitas algumas considerações finais do trabalho expondo alguns pontos interessantes sobre o estudo de caso realizado e também são propostos alguns possíveis trabalhos futuros.

2. Agentes de Software

2.1. Introdução

O desenvolvimento de sistemas baseados em agentes de software é uma das mais promissoras e crescentes áreas na pesquisa acadêmica e na indústria de software [1, 2, 3 4, 5]. Avanços na tecnologia de redes, o aumento de dispositivos móveis no mercado e uma demanda crescente por aplicações para problemas complexos têm revitalizado a investigação da tecnologia de agentes como um novo paradigma promissor para a engenharia de software de sistemas complexos e distribuídos [2, 6, 7, 8]. Atualmente, a tecnologia de agentes tem sido adotada em uma grande variedade de domínios de aplicação, incluindo comércio eletrônico [9, 10], educação à distância [11, 12], computação móvel [13, 14, 15], otimização [16, 12], interface homem–máquina [17, 18], dentre outros [12, 19].

O progresso de diferentes áreas dentro da Ciência da Computação é evidente, sendo que existe uma demanda constante pela integração de métodos e processos de geração de soluções originárias de diferentes áreas (por exemplo engenharia de software, inteligência artificial, redes, *Web* semântica etc). A tecnologia de agentes de software [5, 20, 21] está se desenvolvendo com o objetivo de minimizar problemas que possam surgir na integração destas áreas, facilitando o uso de inovações tecnológicas.

Um agente de software é composto por crenças, metas, planos, capacidades e um conjunto de propriedades comportamentais como autonomia, adaptação, interação, colaboração, aprendizado e mobilidade [22] . Cada uma destas propriedades introduz complexidade adicional ao processo de modelagem, projeto e implementação de sistemas, e, conseqüentemente, aumenta o nível de detalhes da abordagem e a probabilidade da manifestação de situações excepcionais, falhas de segurança e outros fatores que aumentam o risco do desenvolvimento de aplicações [23, 24, 21].

A programação orientada a agentes (*Agent-Oriented Programming* - AOP) é um paradigma de desenvolvimento relativamente novo que traz consigo conceitos de teorias de inteligência artificial distribuída. Segundo [25], AOP basicamente modela uma aplicação como uma coleção de componentes chamados de agentes que, entre outras coisas, são caracterizados pela autonomia, pró-atividade e a capacidade de se comunicar. O conceito de autonomia e de sistemas capazes de executar tarefas complexas de forma inteligente e independente tem despertado grande interesse na comunidade científica. A idéia de sistemas com agentes trabalhando juntos para alcançar um objetivo comum vem reforçando e amadurecendo o conceito de sistemas multi-agentes. Como

conseqüência disso, várias metodologias, arquiteturas e ferramentas estão sendo desenvolvidas para facilitar o desenvolvimento desse tipo de sistema.

2.2. Definição

O uso de agentes já é antigo e seu conceito veio sendo usado em aplicações ligadas à inteligência artificial na tentativa de modelar o raciocínio e o conhecimento dos seres humanos em agentes. Segundo [26], um agente é algo que pode ser visto como observador ou perceptor dentro de um ambiente. As percepções ocorrem através de sensores, e baseado nelas, um agente pode agir no ambiente através de seus efetores. Um agente humano, por exemplo, possui olhos, ouvidos e outros órgãos relacionados à percepção como sensores e mãos, pernas e demais partes do corpo como efetores que lhe permitirá agir no ambiente e interagir com o mesmo. A figura 1 ilustra um agente genérico.

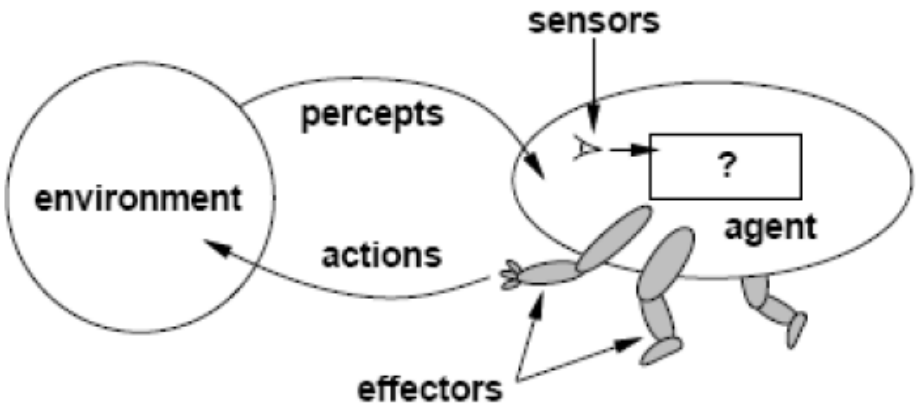

Figura 1: Agente genérico (visão parcial da interação de um agente com o ambiente através de sensores e efetores) [26] .

Para [26] um agente inteligente ideal é aquele que para cada seqüência possível de percepção, ele seja capaz de executar ações de forma a maximizar e otimizar seu desempenho, baseado em evidências providas de sua percepção juntamente com seu conhecimento nativo ou prévio.

Em síntese, um agente consiste em um pedaço de software que executa determinadas tarefas utilizando informações recolhidas em seu ambiente para agir de uma forma adequada, de modo a completar suas tarefas com sucesso. O software deve ser capaz de adaptar-se com base nas

alterações que ocorrem em seu ambiente, de maneira que qualquer mudança de circunstâncias ele continuará a produzir o resultado pretendido.

O termo agente, ou agente de software atualmente se encontra em uma série de aplicações, tecnologias e tem sido amplamente usado em várias áreas, ou seja, não somente em inteligência artificial (área de onde o termo é oriundo) mas também em banco de dados, sistemas operacionais, redes etc. De fato, ainda não existe uma única definição para o termo, porém a maioria delas concorda que um agente se trata de um componente de software especial que tem autonomia, provê interfaces interoperáveis para um sistema arbitrário e que possui comportamentos similares aos dos seres humanos.

Sistemas multi-agentes podem modelar sistemas complexos e introduzem a possibilidade de se ter agentes com objetivos comuns ou não [25]. Esses agentes podem interagir de forma indireta (através do ambiente) ou de forma direta (através da comunicação e negociação entre eles). Agentes podem cooperar para benefício mútuo ou competir para alcançarem seus próprios objetivos [25].

2.3. Características

Na literatura encontram-se várias características importantes que um agente pode possuir. Essas características são divididas em duas categorias: características que representam uma noção fraca de agentes e características que representam uma noção forte [27].

A noção fraca do conceito de agentes é talvez o modo mais geral no qual o termo agente é usado. Essa noção é usada geralmente para denotar o software baseado em um sistema de computador (agente de software) que desfruta das seguintes características [27]:

- Autonomia: agentes que operam sem intervenção humana e que têm algum tipo de controle sobre suas ações;
- Habilidade social: refere-se a capacidade que um agente tem de se comunicar com outros agentes (inclusive agentes humanos);
- Reatividade: agentes que percebem o seu ambiente, e respondem em um modo oportuno a mudanças que acontecem. Isto pode requerer que um agente gaste a maioria de seu tempo em um estado de espera do qual se despertará de acordo com as mudanças que podem ocorrer em seu ambiente;

- Pró-atividade: agentes que simplesmente não agem em resposta ao seu ambiente. Eles podem exibir comportamento tomando sua própria iniciativa;
- Orientadores de meta: um agente que é capaz de manipular tarefas complexas de alto-nível. A decisão tal como uma tarefa é melhor dividida em sub-tarefas menores, e em qual ordem e em qual modo estas sub-tarefas deveriam ser executadas melhor, é feita pelo próprio agente.

A noção forte do conceito de agentes está relacionada às propriedades similares ao comportamento humano tais como conhecimento, crença, intenção etc. Esse conceito carrega consigo as seguintes características [27]:

- Capacidade de aprendizado: característica que refere-se à capacidade que um agente tem de aprender sobre o ambiente que ele se encontra e sobre usuários ou outros agentes que interagem com ele;
- Benevolência: é a suposição que um agente não têm objetivos contraditórios;
- Mobilidade: habilidade de um agente migrar para novos ambientes (percepção de outros ambientes);
- Racionalidade: é a suposição que um agente agirá para alcançar suas metas e não agirá de modo a impedir que suas metas sejam alcançadas. O raciocínio de um agente pode ser de dois tipos: baseado em regras (consiste em um conjunto de regras prévias para poderem avaliar qualquer condição) ou baseado em uma base de conhecimento (dados sobre cenários, ações anteriores dos quais eles inferem seu movimento futuro).
- Colaboração: um agente não deve impulsivamente aceitar e executar instruções. Ele deve levar em conta que um usuário humano comete enganos (por exemplo uma ordem que contém metas contraditórias), omite informação importante e/ou provê informação ambígua. Cabe ao agente identificar esses tipos de erros e mostrar diretrizes de como resolvê-los para o usuário. O outro tipo de colaboração está relacionado à troca de informações entre agentes. Neste caso, essa troca, pode ajudar agentes a cumprirem mais facilmente suas tarefas;
- Robustez: é a capacidade de um agente tomar decisões baseadas em informações incompletas ou escassas, além de saber como lidar em situações não comuns (falhas, erros).

De acordo com as características acima, pode-se observar que implementar agentes com todas as características citadas não é um tarefa trivial. Porém, nem sempre é necessário um agente que tenha todas as características. Existem agentes que possuem apenas uma ou algumas dessas características e nem por isso deixam de ser agentes. O fato é que, atualmente, não existe uma

concordância sobre quais propriedades são efetivamente necessárias para a caracterização de agentes. O consenso é que um agente que possui essas características ou algumas delas diferem muito de simples objetos [27].

2.4. Classificação

Atualmente existem diversas classificações de agentes. Essas classificações são baseadas em seus propósitos, sua arquitetura, tecnologia etc. Segundo [26], a classificação de agentes tem a função de ilustrar as diferentes composições e complexidades que um agente pode assumir. Basicamente o autor classifica agentes em quatro tipos:

- Agentes de reflexo simples: categoria mais simples de agentes. Neste caso, um agente utiliza um conjunto de regras simples do tipo condição-ação como suporte às tomadas de decisões. A figura 2 ilustra esse tipo de agente;
- Agentes que rastreiam o mundo: agentes que necessitam da representação do ambiente em um conjunto de estados possíveis para poderem tomar alguma decisão. O fato é que nem sempre os sensores dos agentes conseguem ver quais são todos os possíveis estados. Neste caso, é necessário que o agente guarde informações internas para auxiliar-lo em suas tomadas de decisões;
- Agentes baseados em metas: agentes que necessitam cumprir metas e, para isto, eles procuram planejar uma seqüência de ações para serem tomadas;
- Agentes baseados na utilidade: são agentes que realizam ações para cumprirem suas metas visando sempre maximar suas expectativas. Suas ações podem ser baseadas em componentes heurística que visam escolher o melhor caminho para atingir suas metas. O melhor caminho nem sempre é a solução ótima (caminho mais curto) para cumprir uma determinada meta. Em alguns casos, achar uma solução aceitável é mais fácil e rápido que achar a ótima.

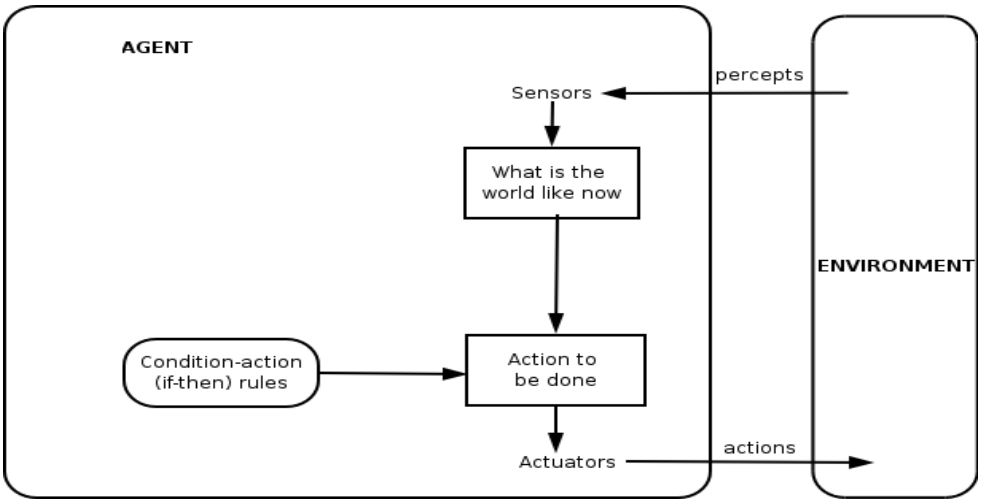

Figura 2: Agentes de reflexo simples [26]

Além das categorias definidas por [26], há outras categorias de agentes (algumas parecidas com as já citadas) definidas na literatura. Dentre elas pode-se citar [27]:

- Agentes móveis: são agentes que tem a mobilidade como principal característica. Esse tipo de agente é capaz de mover-se entre ambientes, transportando-se através de plataformas levando dados e códigos;
- Agentes situados ou estacionários: são agentes opostos aos móveis. Isto é, tratam-se de agentes fixos em um mesmo ambiente;
- Agentes competitivos: são agentes que competem entre si para a realização de seus objetivos ou tarefas. Em outras palavras, o conceito de colaboração não existe nesse tipo de agente;
- Agentes coordenados ou colaborativos: agentes com a finalidade de alcançar um objetivo maior e comum. Para isso eles executam tarefas especificas porém coordenadas. Essa coordenação permite que as tarefas se completem de forma a prover benefícios a cada um dos agentes;
- Agentes reativos: agentes que reagem a estímulos sem ter memória do que já foi realizado no passado e nem previsão da ação a ser tomada no futuro. Eles não têm representação do seu ambiente ou de outros agentes e são incapazes de prever ou antecipar ações. Geralmente são baseados em modelos de organização biológica ou etológica como, por exemplo, uma colônia de formigas, cupins, abelhas. O modelo de funcionamento de um agente reativo é formado pelo par estímulo-resposta ou ação-reação. As principais propriedades dos agentes e dos sistemas multi-agentes reativos são: ausência de representação explícita do

conhecimento, não há representação do ambiente (o agente somente responde quando percebe estímulos), não mantém nenhum tipo de histórico de suas ações, ou seja, o resultado de uma determinada ação passada não influência diretamente na decisão de uma ação futura, organização etológica (é similar à observada por animais que vivem em grandes comunidades. No caso de uma colônia de formigas uma única delas não apresenta muita inteligência, porém, quanto age em grupo, o todo comporta-se como uma entidade com uma certa inteligência.);

- Agentes cognitivos: esses, ao contrário dos agentes reativos, podem raciocinar sobre as ações tomadas no passado e planejar ações a serem tomadas no futuro. Ou seja, um agente cognitivo é capaz de resolver problemas por ele mesmo. Ele tem objetivos e planos explícitos os quais permitem atingir seu objetivo final. [28] afirma que para que isso se concretize, cada agente deve ter uma base de conhecimento disponível, que compreende todos os dados e todo o *know-how* que o auxilia na realização de tarefas, na interação com outros agentes e com o próprio ambiente. Sua representação interna e seus mecanismos de inferência o permitem atuar independentemente dos outros agentes e lhe dá uma grande flexibilidade na forma de expressão de seu comportamento. Além disso, devido à sua capacidade de raciocínio baseado nas representações do mundo, são capazes de ao mesmo tempo memorizar situações, analisá-las e prever possíveis ações. Esse tipo de agente é ilustrado na figura 3 de forma esquemática;
- Agentes de interface: também chamados assistentes pessoais ou agentes de usuário, possuem o objetivo de simplificar as tarefas rotineiras realizadas por um usuário. Esse tipo de agente observa e monitora as ações tomadas pelo usuário na interface, aprende novos atalhos e sugere melhores formas de executar a tarefa. O agente de interface age como um assistente pessoal autônomo o qual coopera com o usuário na realização de determinadas tarefas;
- Agentes híbridos: são agentes que possuem mais de uma característica. Se mostram como uma alternativa para dotar o agente com capacidades reativas apropriadas, visando solucionar a incapacidade de ação adequada por parte de um agente puramente cognitivo no momento em que o mesmo deve tomar uma decisão rápida e espontânea ao enfrentar uma situação imprevista. Em contra-partida, servem também para proporcionar a um agente puramente reativo, capacidade de raciocínio e planejamento quando o mesmo se deparar com uma situação na qual o ambiente diverge bastante dos seus objetivos iniciais. A figura 4 mostra um exemplo de tipologia de agentes providos de mais de uma característica.

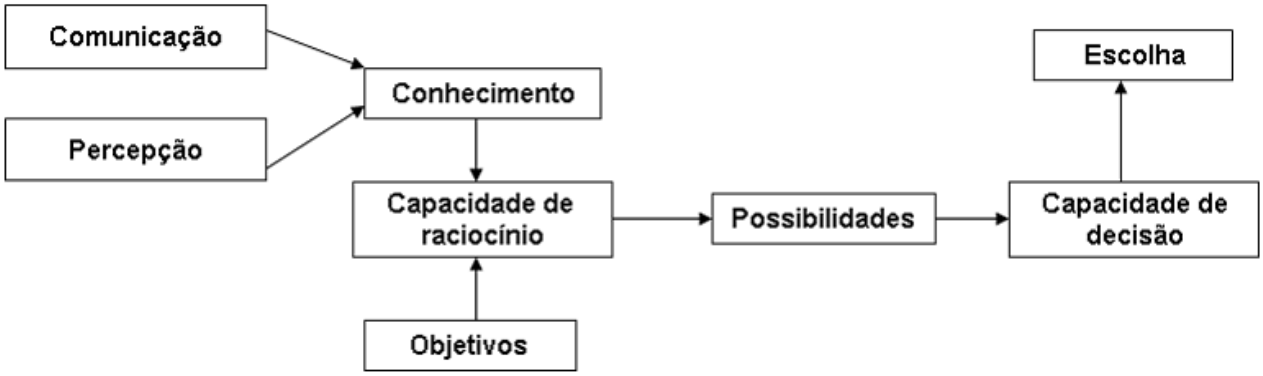

Figura 3: Estrutura de um agente cognitivo genérico [27]

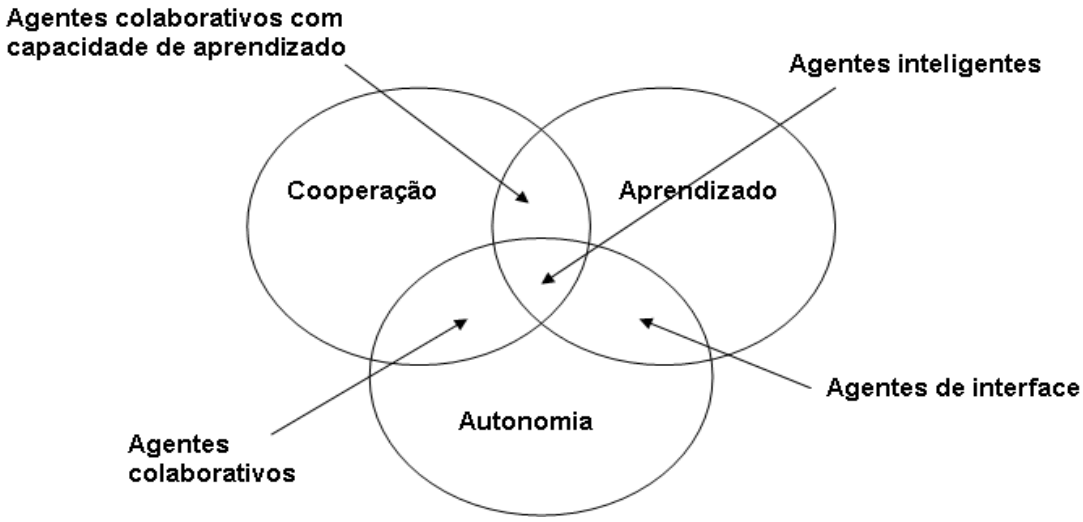

Figura 4: Tipologia de agentes [19]

2.5. Comunicação entre Agentes

A comunicação entre agentes em sistemas multi-agentes é um requisito fundamental para que haja cooperação e negociação entre eles. Em sistemas muti-agentes, é necessário que a comunicação seja disciplinada para que os objetivos dos agentes sejam alcançados de forma efetiva e eficiente [27]. Para isso, são necessárias regras ou protocolos de comunicação para que qualquer troca de mensagem entre agentes possa ser entendida por todos. A comunicação tem como objetivo principal prover a partilha de conhecimento e a coordenação de atividades entre agentes. Em outras palavras, a comunicação deve permitir que agentes troquem informações e coordenem suas atividades de maneira a resultar sempre em um sistema coerente.

Existem diversas maneiras para agentes trocarem informações uns com os outros em um sistema multi-agentes [29]. Agentes podem trocar mensagens diretamente (conhecida como

comunicação direta), podem se comunicar através de um agente facilitador (comunicação assistida), podem também se comunicar através de difusão de mensagens (*broadcast*) ou utilizar um modelo de comunicação do tipo quadro-negro (*blackboard*).

Na comunicação direta cada agente comunica diretamente com qualquer outro agente sem qualquer intermediário. Há um estabelecimento de uma conexão direta (ponto-a-ponto) entres os agentes através de um conjunto de protocolos que garantem a chegada de mensagens com segurança. Nesse tipo de comunicação faz-se necessário que cada agente envolvido tenha conhecimento da existência dos outros agentes e da forma de como endereçar mensagens para eles.

A principal vantagem deste tipo de comunicação entre agentes é o fato de não existir um agente coordenador da comunicação. Agentes coordenadores podem ser o gargalo do sistema. Em uma situação onde há um grande número de troca de mensagens entre agentes, os agentes coordenadores podem ficar sobrecarregados de tarefas e com isso o desempenho do sistema pode ficar comprometido. Uma desvantagens da comunicação direta é a implementação que se torna muito complexa quando comparada às outras formas de comunicação. A figura 5 ilustra esse tipo de comunicação.

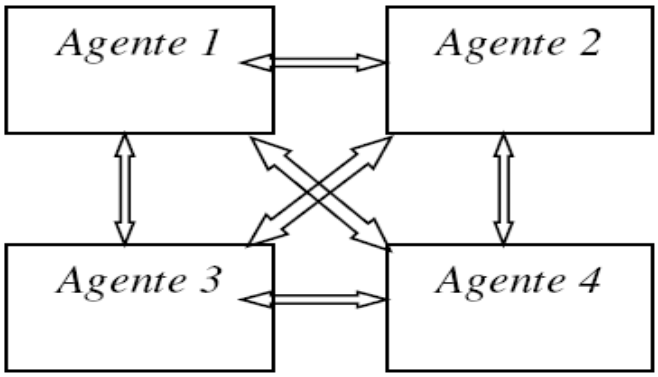

Figura 5: Comunicação direta entre agentes [27]

Na comunicação assistida [29], os agentes utilizam algum sistema ou agente especial para coordenar suas atividades. Ou seja, uma estrutura hierárquica de agentes é definida e a troca de mensagens dá-se através de agentes especiais designados facilitadores ou mediadores. Essa é uma alternativa bem popular à comunicação direta pois diminui muito o custo e a complexidade necessária aos agentes individuais na realização da comunicação. A figura 6 ilustra a comunicação assistida.

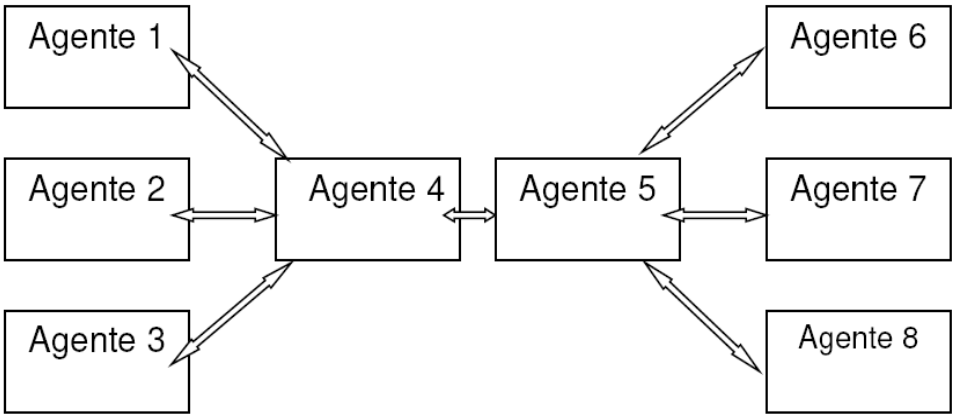

Figura 6: Comunicação assistida [27]

A comunicação por difusão de mensagens ou *broadcast* geralmente é utilizada em situações onde a mensagem deve ser enviada para todos os agentes do ambiente ou quando o agente remetente não conhece o agente destinatário ou seu endereço. Em síntese, todos os agentes recebem a mensagem enviada.

A comunicação por quadro-negro ou *blackboard*, segundo [29], é bastante usada na inteligência artificial como modelo de memória compartilhada. Ou seja, nada mais é que um repositório onde os agentes escrevem mensagens a outros agentes e obtêm informações sobre o ambiente.

2.6. Considerações Finais

Neste capítulo foi feita uma revisão bibliográfica de agentes de software para servir de base conceitual para os demais capítulos desse trabalho. O principal objetivo dessa revisão foi apresentar as diversas concepções relacionadas a agentes, características, classificação e a motivação de sua utilização em sistemas.

3. Projeto Integra – Proposta de Desenvolvimento de um Ambiente de Colaboração Integrado ao Siga e aos Serviços Oferecidos pelo Google.

3.1. Introdução

Ferramentas de colaboração cada vez mais estão sendo necessárias nas instituições de ensino [38]. Ambientes virtuais colaborativos, através da interconexão de máquinas fornecida pelas redes de computadores e pela Internet, visam facilitar as atividades em grupo [35]. Segundo [30], a contribuição de pessoas com diferentes entendimentos, pontos de vista alternativos e habilidades complementares pode gerar resultados que dificilmente seriam encontrados individualmente. Pessoas que são membros de um grupo podem ajudar a identificar inconsistências no raciocínio dos indivíduos e buscar em conjunto idéias, informações e referências para auxiliar na resolução dos problemas. Geralmente, um grupo tem mais capacidade de gerar criativamente alternativas, levantar as vantagens e desvantagens de cada uma, selecionar as viáveis e tomar melhores decisões do que os indivíduos separadamente [31]. Trabalhar em grupo também traz motivação para o membro, pois seu trabalho vai estar sendo observado, comentado e avaliado por pessoas de uma comunidade da qual ele faz parte (seu grupo de trabalho) [32]. Quando um indivíduo expressa idéias para poder se comunicar com outros, ele trabalha ativamente seus conceitos, refletindo sobre os mesmos e refinando-os, trazendo uma melhoria à qualidade do trabalho e do aprendizado [33].

Hoje o Instituto de Ciências Exatas (ICE) da Universidade Federal de Juiz de Fora (UFJF) enfrenta o desafio de desenvolver uma plataforma de colaboração *Web* integrada aos serviços que o Google [37] oferece e ao Sistema Integrado de Gestão Acadêmica (SIGA) [50] já existente na instituição [38]. Esta plataforma trata-se de um projeto *open source*, desenvolvido em Java, e desfruta de conceitos que envolvem padrões de projeto [34], *Web Services* [36], sistemas distribuídos entre outros. Este trabalho vem sendo desenvolvido em parceria entre ICE e alunos com experiência em desenvolvimento de sistemas.

3.2. Contexto do Projeto

Atualmente a UFJF conta com algumas formas de comunicação que buscam prover a interação entre as pessoas que fazem parte dela [38]. Dentre essas formas pode-se citar: listas de e-mail, sistemas de calendários, *home pages* de professores, diretórios de arquivos, e-mail de

professores, funcionários e alunos, sistemas de agendamento de salas entre outros. A existência de vários sistemas com o mesmo propósito na instituição carrega consigo alguns problemas [38]:

- Falta de padronização: existem vários sistemas de calendário e agendamento de recursos espalhados pela UFJF, além de, diversos provedores de listas de e-mail e e-mail de professores e funcionários. A maioria desses sistemas não comunicam entre si e, portanto, não estabelecem nenhuma relação;
- Dificuldade e rejeição de utilização: estas formas de comunicação na UFJF, quando existem, não seguem um padrão, dificultando a utilização em massa e tornando sua adoção dependente de um grande interesse dos usuários;
- Replicação de dados: vários desses sistemas não estabelecem também nenhuma comunicação com o sistema acadêmico da instituição - SIGA. Em alguns deles, é necessário o recadastro de dados já existentes no SIGA. Como conseqüência, tem-se dados replicados e ainda sujeitos a erros e dessincronia.

O fato é que a existência desses vários sistemas está longe do ideal [38]. O ideal seria a existência de um único sistema de comunicação e colaboração que contemplasse todas as funcionalidades de todas as formas de comunicação existentes na instituição e ainda integrado ao SIGA. Porém, o desenvolvimento desse tipo de sistema também envolve alguns problemas. Dentre eles pode-se citar [38]:

- Falta de mão de obra: para que se desenvolva um sistema de comunicação integrado que contemple um sistema de calendário, agendamento, e-mail para alunos, professores e funcionários, listas de e-mail entre outros, são necessários desenvolvedores, analistas de sistemas altamente capacitados;
- Custo: sistemas de comunicação são difíceis de se desenvolver e caros de se manter, tanto a infra-estrutura de hardware quanto a equipe técnica. Um exemplo seria a dificuldade de se manter um servidor de e-mail para alunos e professores.

De fato, todos os serviços de comunicação citados são muito custosos de se criar (principalmente o serviço de e-mail) e já existem abundantemente no mercado. Logo, recriá-los além de complicado, requer a heróica atividade de mantê-los competitivos com o que há fora da instituição, sob pena de não utilização dos recursos criados pela UFJF [38].

Dentre os serviços já existente no mercado pode-se citar o Google Apps Education Edition [37]. Trata-se de um serviço oferecido pelo Google que tem como objetivo oferecer, às instituições de ensino, ferramentas gratuitas de comunicação (e-mail com domínio da instituição, listas de e-mail, criação e hospedagem de *home pages* sem necessidade de conhecimento técnico), agendamento, colaboração e publicação (compartilhamento de conteúdo) entre outras. Com o

Google Apps Education não há necessidade de uma instituição de ensino se preocupar com toda a infra-estrutura necessária para um sistema de comunicação e colaboração [37].

Nesse contexto, nasceu o Projeto Integra que tem como objetivo principal criar um ambiente de colaboração na universidade integrado ao SIGA e ainda, utilizando recursos que o Google oferece. A figura 7 ilustra a idéia do Projeto Integra.

Figura 7: Idéia do Projeto Integra [38]

O projeto consiste na criação de uma Plataforma, chamada Integra, que pode ser vista como uma forma de se abstrair a complexidade da união de dados providos de sistemas diferentes através de interfaces de gerenciamento de recursos. Basicamente, através dessas interfaces será possível fornecer uma conta de e-mail institucional para cada aluno, professor e funcionário, criar listas de e-mail e calendários gerenciáveis para cada disciplina, departamento e grupo de estudo, criar diretórios *on-line* para compartilhamento de documentos, compartilhamento de vídeos e imagens entre usuários. Além disso, essa integração traz consigo as seguintes vantagens [38]:

- Padronização dos recursos: os usuários acessarão a Plataforma de integração para compartilhar documentos, criar listas de e-mail, agendar eventos etc;
- Baixo custo relativo: grande parte dos serviços serão mantidos na infra-estrutura do Google, evitando assim custos com infra-estrutura na instituição;
- Consistência de dados: os dados podem ser "importados" do SIGA, eliminando assim problemas de replicação de dados;
- Acessível a todos: devido à padronização, ao uso de interfaces amigáveis, à facilidade de utilização e à alta disponibilidade, diversos tipos de usuários (não somente os mais afeitos a tecnologia) terão acesso a recursos de e-mail sofisticados, calendário on-line ou possibilidade de criar sua própria *home page*, por exemplo;
- Maior colaboração: os professores poderão disponibilizar listas de exercícios, trabalhos, textos e planilhas por meio digital. Os alunos poderão entregar seus trabalhos dessa mesma forma;
- Maior usabilidade: eventos, datas de entregas de trabalhos, reuniões, inscrições, ocupação de salas de aula, horários de aulas, entre outros estarão todos registrados numa interface padronizada e de fácil manipulação.

3.3. Detalhes Técnicos do Projeto Integra

Nesta seção serão comentados alguns detalhes técnicos da Plataforma. Mais especificamente os módulos responsáveis pela comunicação inter-sistemas, os serviços envolvidos, padrões de projetos utilizados e as arquiteturas do software.

3.3.1. Módulos de Interoperação – Comunicação entre Sistemas

A Plataforma Integra contêm três módulos responsáveis pela integração: SigaDao, GoogleDao e IntegraDao [38].

O módulo SigaDao é o responsável por comunicar-se com o SIGA. Através dele, torna-se possível a obtenção de dados de alunos, professores, funcionários, turmas, disciplinas da UFJF. Este módulo faz uso do padrão de projeto *value object*. Esse padrão é utilizado sempre que uma aplicação necessita de dados de um serviço remoto [36]. Em síntese, consiste em trazer todos os dados remotos necessários, em uma única requisição encapsulados em um objeto chamado de *value object*. A utilização desse padrão melhora o desempenho do sistema, uma vez que, buscar uma informação por vez via rede pode ter um alto custo em termos de desempenho.

O módulo GoogleDao é o responsável por se comunicar com o Google. Basicamente o módulo consome *Web Services* disponibilizados pelo Google de forma a tornar possível a criação de contas de e-mail, criação de listas de e-mail, agendamento de eventos etc.

O módulo IntegraDao foi construído para manipular os dados intermediários necessários para integração (dados que fazem a ligação entre SIGA e Google). Esse módulo também é o responsável por persistir *logs* de qualquer tarefa que envolva a comunicação entre os sistemas.

É importante ressaltar que os três módulos de integração trabalham de forma independente, ou seja, um não sabe da existência do outro [38]. Para isso, em cada um desses módulos, foi utilizado o padrão de projeto *data access object* – DAO para abstrair e encapsular o acesso à sua respectiva base de dados e disponibilizar um acesso transparente aos demais módulos do sistema. Porém, como o projeto trata da integração dos sistemas, em várias situações as saídas providas de um desses módulos são as entradas de outro. Um exemplo disso é a criação de e-mail de um aluno, onde primeiramente o SigaDao fica encarregado de buscar os dados necessários para a criação do e-mail no SIGA, em seguida o IntegraDao salva os dados intermediários e, por fim, o GoogleDao cria o e-mail do aluno no Google. Para a realização desse tipo de tarefa foi criada uma camada de serviço (*service layer*) no sistema que faz a junção dos módulos de integração de forma a fazê-los trabalhar em conjunto. A figura 8 ilustra um esquema dessa camada de serviço.

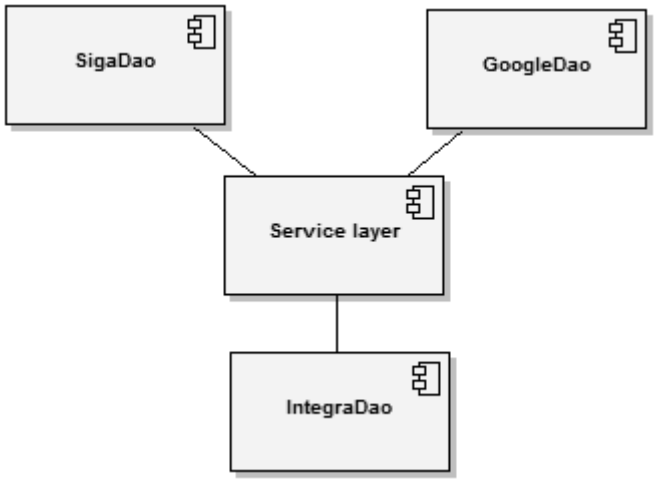

Figura 8: Visão esquemática da camada de serviço - junção dos módulos de integração [38]

Um ponto que deve ser levado em consideração, tratando-se de integração, é com relação à consistência de bases de dados [38]. No caso da Plataforma, a base de dados intermediária deve estar consistente tanto com o SIGA quanto com o Google. Caso essa base intermediária esteja inconsistente tem-se um sistema não coerente. Por exemplo, no caso da criação de e-mail para um aluno (citado acima), caso ocorra algum erro durante a criação do e-mail no Google quando os dados intermediários já foram salvos. Nesta situação temos um sistema incoerente, pois há dados na base intermediária que dizem que um determinado aluno tem um e-mail no Google sendo que, na verdade, ele não tem. Para evitar esse problema, foram implementados mecanismos de transação na camada de serviço para assegurar que todas as tarefas requeridas para a integração sejam realizadas. Em outras palavras, caso todas as etapas necessárias para prover uma integração coerente não forem realizadas, a camada de serviço se encarrega de realizar o *rollback* de todas as ações efetuadas durante a tentativa de integração. Desta forma tem-se sempre um sistema consistente.

3.3.2. Arquiteturas do Sistema

O uso de camadas (*layers*) é um padrão arquitetural que auxilia na tarefa de separar responsabilidades, promovendo baixo acoplamento e alta coesão entre as partes de um sistema [36]. No caso da Plataforma Integra foi utilizado esse padrão arquitetural com o objetivo de separar as tarefas do sistema em níveis diferentes de abstração. O sistema é composto por quatro camadas [38]:

- Dao: camada constituída pelos módulos de integração;
- Serviço: camada responsável pela junção (padrão *facade*) [34] dos Daos;

- *Controller*: permite acesso da camada visão aos serviços de forma seleta. Além disso torna os dados independentes das telas.
- Visão: interfaces gráficas com o usuário, para a manipulação do sistema.

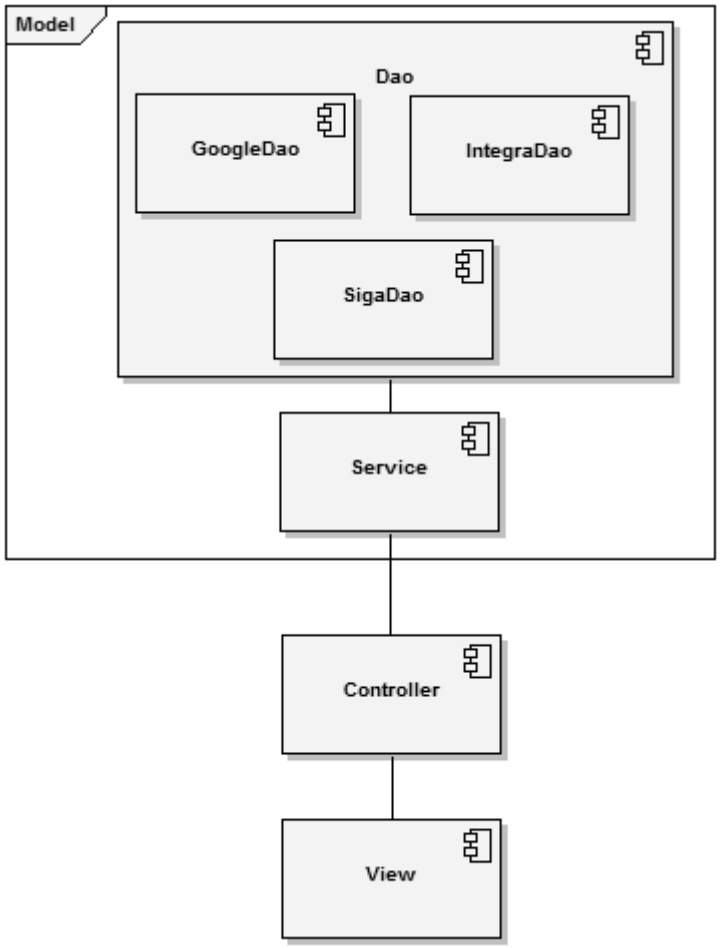

Figura 9: Arquiteturas do sistema (camadas – MVC) [38]

Foi utilizado também o padrão arquitetural *model-view-controller* (MVC) com o intuito de prover a separação entre os dados (*model*) e o *layout* (*view*) [36]. Desta forma, alterações feitas na visão não afetam a manipulação de dados, e estes poderão ser reorganizados sem alterar a visão. É importante ressaltar que a visão só tem acesso a dados através de *controllers*. A figura 9 ilustra as arquiteturas do sistema.

3.3.3. SAML - *Single Sign-On*

Outro aspecto técnico importante do projeto é o serviço de *Single Sign-On* (SSO). De forma resumida, esse serviço permite que um usuário acesse sua conta Google sem que o Google o

autentique, mas sim o SIGA. A figura 10 ilustra todas as etapas (oito passos) desse mecanismo de autenticação.

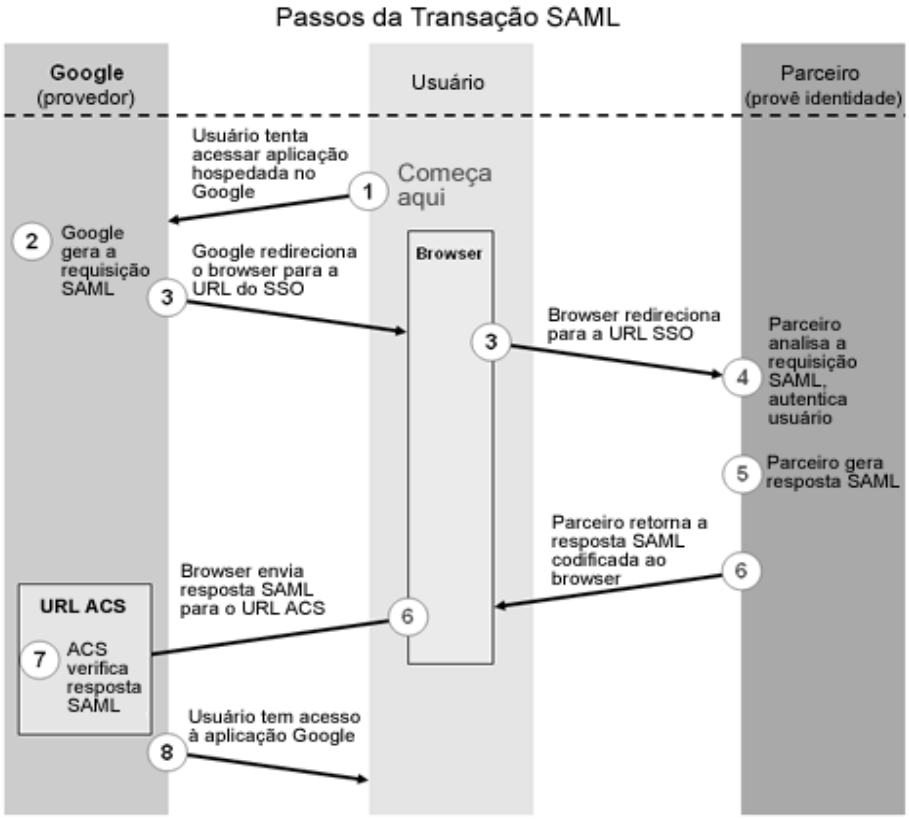

Figura 10: SAML - Single Sign-On [37]

O SSO ocorre da seguinte forma: sempre que um usuário for acessar sua conta Google (passo 1), ele será redirecionado para a Plataforma Integra juntamente com uma requisição de autenticação (passos 2 e 3). A Plataforma então analisa a requisição e pede ao usuário que entre com sua matrícula e senha do SIGA (passo 4). Assim que o usuário for autenticado pelo SIGA, através da Plataforma, é gerada uma resposta à requisição recebida (passo 5) que em seguida é enviada para o Google (passo 6). Após o envio, o Google irá examinar a resposta para averiguar se o usuário foi corretamente autenticado (passo 7). Caso a autenticação tenha sido bem sucedida, o usuário pode assumir total controle de sua conta no Google (passo 8). A requisição e resposta ocorrem através de um arquivo XML no padrão *security assertion markup language* (SAML) e são enviados através da URL do *browser*.

Por questões de praticidade, a Plataforma Integra foi implementada de forma a simular uma requisição de autenticação do Google. Desta maneira, basta que um usuário simplesmente entre na Plataforma para que tenha acesso os serviços do Google, pois estará automaticamente autenticado.

3.4. Considerações Finais

O Projeto Integra é relativamente novo estando portanto ainda em fase de desenvolvimento. Apesar disso, o projeto já possui várias funcionalidades implementadas (todas integrando os sistemas de forma transparente para o usuário) e prontas para serem usadas. Uma dessas funcionalidades é a de criação de contas Google com autenticação e dados do SIGA. Outra é a de criação de grupos de discussão. Cada grupo é composto por vários participantes e possui uma ou mais listas de e-mail para que todos os membros possam se comunicar de maneira ágil.

De forma geral, o desenvolvimento do Projeto Integra traz consigo um leque de possibilidades no contexto da colaboração e comunicação. Essas possibilidades são oriundas da forma de como o projeto está sendo pensado e desenvolvido pelo grupo de trabalho.

A equipe de desenvolvimento está sempre estudando as melhores práticas de programação, as tecnologias mais adequadas para que, no fim do projeto, tenha-se uma ferramenta de colaboração sofisticada, robusta, que atenda às necessidades da universidade e sirva como instrumento de apoio e difusão do conhecimento entre alunos, professores e funcionários da instituição. Além disso, espera-se que os recursos oferecidos pela Plataforma traga facilidades para todos que fazem parte da UFJF principalmente em termos de agilidade de acesso às informações.

4. Agentes na Recuperação de Falhas: Um Estudo de Caso na Plataforma Integra

4.1. Introdução

Em software, falhas são inevitáveis. Atualmente, tem-se cada vez mais necessidade de softwares que descrevam soluções sofisticadas para problemas do mundo real. Como conseqüência disso, são introduzidas complexidades a todo processo de desenvolvimento para tornar possível a confecção dessas soluções. Uma vez um sistema desenvolvido, ele estará sempre sujeito à mudanças e adaptações para que suas soluções estejam sempre em sincronia com o domínio do problema para o qual ele foi projetado. Cada complexidade adicional nas etapas de desenvolvimento, mudanças e adaptações aumentam as chances de manifestação de comportamentos não esperados do sistema [39]. Esses comportamentos são chamados de falhas.

De maneira geral, falhas podem ser classificadas como físicas ou humanas [40]. As físicas referem-se a falhas de componentes de hardware e geralmente são causadas por desgaste material. Já as humanas, compreendem erros de projeto decorrentes das fases de desenvolvimento do software. Falhas podem levar um sistema a um estado errôneo e, caso não seja tratado em tempo hábil, ocorrerá um defeito que se manifestará pela execução indesejada de um determinado serviço ou função do sistema. A ocorrência de um defeito pode dar origem a outras falhas que, por sua vez, podem gerar novos defeitos. Esse fenômeno é conhecido como propagação de falhas e é ilustrado pela figura 11. Segundo [40], falhas também podem ser classificadas de acordo com sua duração:

- Transientes: são aquelas de duração limitada, causadas por mau funcionamento temporário ou por alguma interferência externa;
- Intermitentes: são aquelas que se repetem mas não necessariamente em períodos definidos;
- Permanentes: são caracterizadas pela sua presença constante.

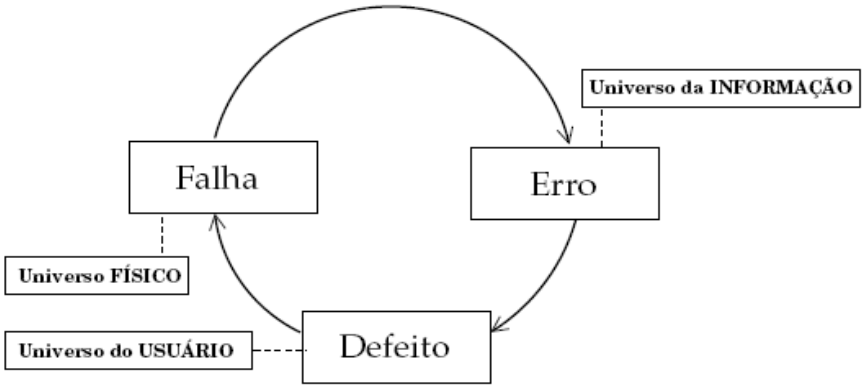

Figura 11: Ciclo de ocorrência de falhas, erros e defeitos [41]

Apesar da probabilidade de um sistema não apresentar falhas ser baixa, sabe-se previamente o que no sistema deve ser mantido consistente independente das falhas que possam vir a ocorrer. Nesse contexto, nasceu a proposta deste trabalho que é a utilização de agentes de software como mecanismo de tolerância a falhas. A idéia da proposta é desenvolver agentes como um módulo autônomo do sistema, providos de características adequadas, que conheçam antecipadamente o que no sistema deve estar coerente a todo momento. Basicamente a função desse módulo será fazer checagens otimizadas, de forma a amenizar o *overhead* no sistema, para averiguar se existe ou não falhas e, caso exista, realizar ações (baseadas em regras ou em sua cognição) para recuperar-se delas e manter o funcionamento do sistema conforme sua especificação.

Para ilustração da proposta, será feito um estudo de caso sobre o desenvolvimento de agentes de software para a Plataforma Integra para recuperação de falhas.

4.2. Agentes na Plataforma Integra

4.2.1. Preliminares

Como já fora citado anteriormente, a Plataforma Integra realiza a integração entre dois sistemas distintos. Como todas as integrações de sistemas, dados precisam ser combinados de forma que a integração tenha sentido e significado coerente no contexto da aplicação. No caso da Plataforma Integra, a base de dados intermediária é que faz a combinação entre os dados dos sistemas para prover a integração. Porém, caso a base intermediária não esteja consistente (devido a falhas sistêmicas), o sistema pode apresentar defeitos e como conseqüência comprometer seu uso. Existem alguns fatores relacionados ao próprio ciclo de vida de software que podem contribuir para o aparecimento de falhas na Plataforma. Dentre eles pode-se citar:

- Sistema em fase inicial de uso: a Plataforma encontra-se em suas primeiras versões para utilização. Segundo [39], cerca de 50 a 70% de todo esforço gasto em um software serão despendidos depois da primeira vez que ele foi implantado devido à manifestação de falhas. Em algumas dessas possíveis falhas pode ser que a base de dados intermediária fique inconsistente;
- Novas versões em desenvolvimento: para a construção de novas versões, sempre há modificações no sistema e adição de novas funcionalidades. Isso implica na possibilidade de aparecimento de novos erros que podem comprometer a consistência da base de dados intermediária.

Apesar da Plataforma dispor de mecanismos de transação para qualquer atividade que envolva a integração entre os sistemas, é importante salientar que esses mecanismos também podem ser falhos em situações imprevistas durante o desenvolvimento do sistema (mesmo que sejam situações extremamente não comuns).

Levando em consideração essas possíveis situações e fatores, e ainda sabendo que a base intermediária deve estar sempre consistente, foi proposto para a Plataforma Integra um módulo composto por agentes autônomos, chamado de *Agent*, que é responsável pela manutenção da base de dados intermediária consistente, sem que seja necessário intervenção humana.

4.2.2. Módulo *Agent* da Plataforma Integra

O módulo *Agent* é composto por três agentes que colaboram entre si para fazer a sincronização entre as bases de dados utilizadas pela Plataforma Integra quando necessário. Esses agentes são denominados de: IntegraAgent, GoogleAgent e o CoordinatorAgent.

O IntegraAgent é o agente responsável por atender às requisições de consultas à base de dados intermediária da Plataforma Integra. Sempre que existir a necessidade de consultas a essa base de dados, o IntegraAgent é contactado para a realização dessa tarefa. Para isso, ele dispõe de serviços para a comunicação com a base de dados.

O GoogleAgent tem função parecida com o IntegraAgent. Ele é o responsável por atender às requisições de consultas de dados ao Google. Essas consultas são feitas através serviços que consomem *Web Services* disponibilizados pelo Google.

O CoordinatorAgent é o responsável por coordenar a sincronização entre os sistemas. De tempos em tempos (definido por um *tick*) ele solicita ao IntegraAgent e ao GoogleAgent dados sobre as respectivas bases que cada um tem acesso. Com referência nos dados (respostas) obtidos do IntegraAgent e do GoogleAgent, o CoodinatorAgent compara os dados (verifica se o que existe

na base intermediária existe no Google e vice-versa) e baseado em suas regras (refere-se a que decisão tomar quando identificar inconsistência entre as bases de dados – regras comportamentais no formato QUANDO-SE-ENTÃO – figura 13) e conhecimento (dados de configuração, ou seja, que devem existir em apenas uma das bases de dados) inicia o processo de sincronização caso seja necessário. Neste módulo, não houve a necessidade de um agente que inspecionasse a base de dados do SIGA, pois a Plataforma só acessa os dados acadêmicos para leitura (o que elimina problemas de inconsistências). A figura 12 mostra através de um diagrama de seqüência a interação e comunicação dos agentes propostos para este módulo.

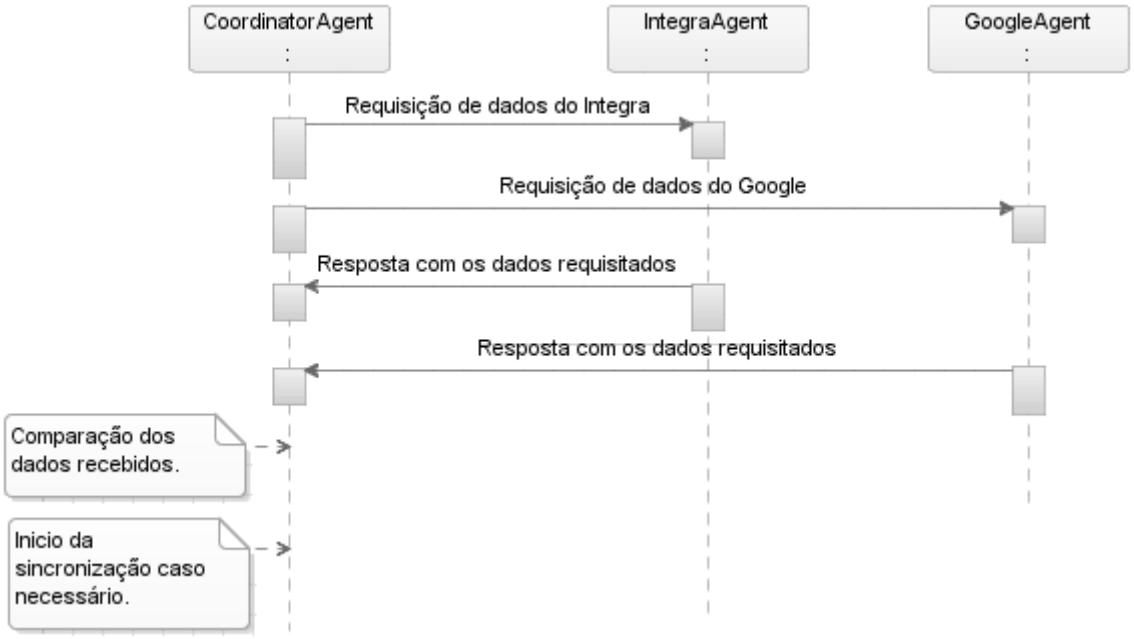

Figura 12: Interação dos agentes

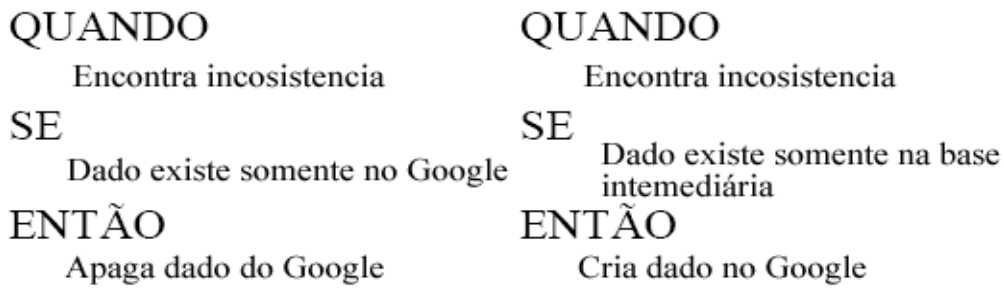

Figura 13: Regras comportamentais

Apesar de um agente se comunicar diretamente com o outro, a princípio eles não precisam se conhecer. Isto ocorre devido à maneira como o módulo foi projetado e implementado. Em síntese, o módulo *Agent* utiliza um repositório de serviços onde os agentes publicam serviços (somente a descrição do serviço). Desta forma agentes podem procurar por serviços, sempre que precisarem, sem necessidade de saber antecipadamente quem disponibiliza o mesmo. Um vez que

um agente encontra no repositório um serviço procurado, ele começa a se comunicar com o agente que o disponibiliza para sua realização. Por exemplo, no caso do módulo, o IntegraAgent e o GoogleAgent publicam serviços de acesso a dados de cada uma de suas respectivas bases e o CoordinatorAgent apenas procura pelos serviços oferecidos. Em síntese, o repositório é utilizado como um mecanismo para o compartilhamento de informações ou dados por vários componentes de softwares (estilo arquitetural *blackboard*). O repositório é ilustrado pela figura 14.

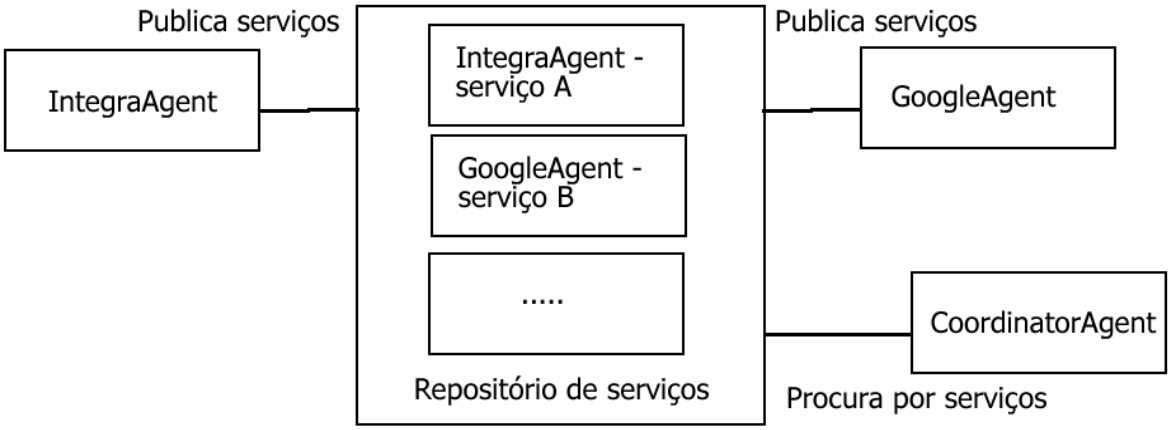

Figura 14: Repositório de serviços

O IntegraAgent e o GoogleAgent possuem as mesmas características pelo fato de desempenharem tarefas similares, que é o acesso a dados, diferenciando somente na forma de como o acesso é feito. Dentre as características que eles possuem, pode-se citar:

- Habilidade social: ambos respondem às solicitações de serviços do CoodinatorAgent;
- Autonomia: os agentes agem sem intervenção humana;
- Reatividade: os agentes ficam em estado de espera do qual se despertam assim que percebem alguma solicitação de um agente (CoodinatorAgent) para execução de algum serviço.

O CoodinatorAgent, além de possuir todas as características dos demais agentes, possui algumas outras que são:

- Pró-atividade: executa ações independente de respostas do ambiente;
- Racionalidade: o agente possui um conjunto de regras e uma base de conhecimento para executar suas ações.

Concluindo, este componente adicional na Plataforma Integra trata-se de um módulo autônomo e é executado em *background* sempre que a mesma é iniciada. Ou seja, ele executa suas tarefas através de suas percepções do ambiente independente se há ou não interação humana com o software. A figura 15 ilustra a interação do módulo com o ambiente.

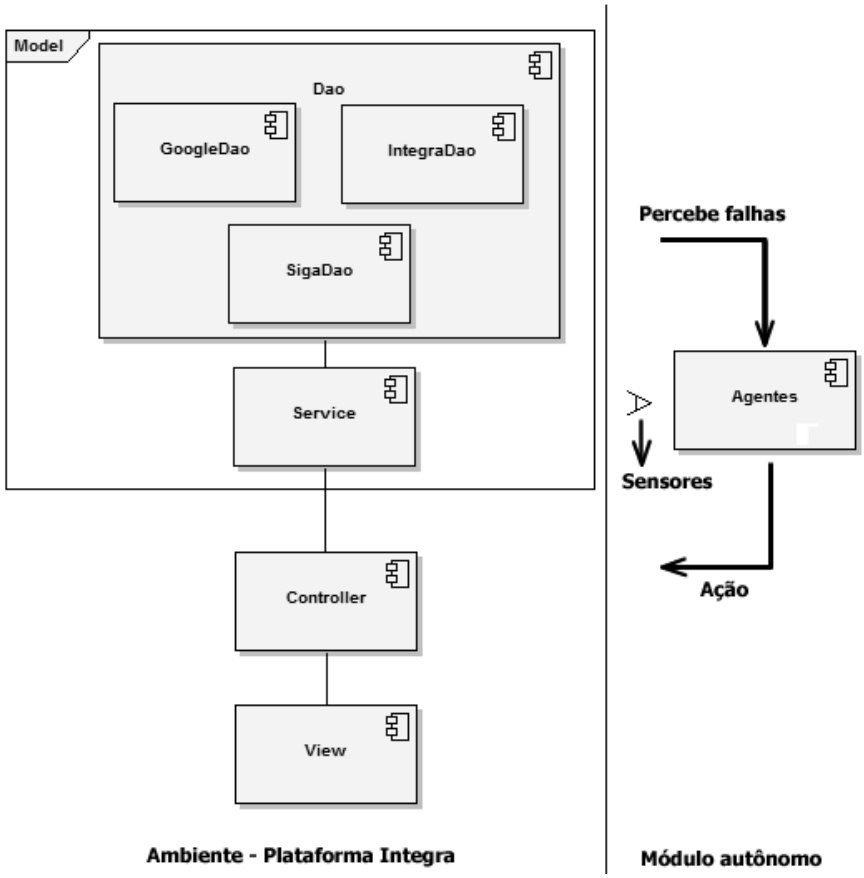

Figura 15: Interação módulo Agent – Ambiente

4.2.2. Implementação Computacional do Módulo *Agent*

Para a implementação deste módulo, foram utilizadas as seguintes tecnologias:

- Linguagem Java versão 6 [42]: linguagem de programação de propósito geral. Foi a linguagem utilizada para a implementação do módulo *Agent;*
- Eclipse versão 3.2 [48]: ambiente integrado de desenvolvimento, em inglês IDE (*Integrated Development Enviroment*), usado para criação do módulo. É amplamente extensível, com o uso de *plugins*, permitindo a edição de diversos tipos de arquivos;
- JADE versão 3.6 (Java *Agent DEvelopment Framework*) [43]: *framework* que oferece um ambiente de desenvolvimento de aplicações baseadas em agentes de acordo com os padrões de interoperabilidade entre sistemas multi-agentes. O ambiente JADE fornece através um vasto conjunto de funcionalidades e facilidades para implementação de agentes de software;
- Spring versão 2.5 [44]: *framework* não intrusivo baseado nos padrões de projeto inversão de controle (loc) e injeção de dependência. O Spring é responsável por gerenciar os *beans*

de uma aplicação através de sua criação automática e definição de suas dependências através de arquivos de configurações em formato XML. Oferece diversas funcionalidades que podem ser utilizadas de acordo com as necessidades do projeto tais como persistência de dados, acesso remoto, mecanismos de segurança e programação orientada a aspectos;

- Hibernate versão 3.3 [45]: *framework* para o mapeamento objeto-relacional desenvolvido em Java. Facilita o mapeamento dos atributos entre uma base de dados relacional e um modelo orientado a objetos de uma aplicação mediante o uso de arquivos XML ou anotações (*annotations*);
- Banco de dados MySQL versão 5.0 [46]: sistema de gerenciamento de banco de dados relacionais que utiliza a linguagem SQL (*Structured Query Language* - Linguagem de Consulta Estruturada) como interface. No caso da implementação do módulo *Agent*, ele foi utilizado para armazenar o conhecimento do CoordinatorAgent;
- JUnit versão 3.8 [47]: *framework* que com suporte à criação de testes unitários automatizados para aplicações Java;
- Tomcat versão 6.0 [49]: servidor de aplicações Java para *Web*. Trata-se de um software robusto e eficiente o suficiente para ser utilizado em ambientes de produção.

4.2.3. Arquitetura dos Agentes

Nesta seção será apresentada a arquitetura interna dos agentes do módulo proposto. A idéia básica por trás da arquitetura de um agente é mostrar as diversas partes que o compõem, e ainda, explicitar a interação entre elas. A descrição da arquitetura dos agentes é ilustrada pela figura 16.

De acordo com a arquitetura proposta (figura 16), na parte superior, encontram-se os comportamentos ativos de um agente. Esses representam as ações ou intenções atuais que cada agente possui. Um exemplo de comportamento, em analogia com o módulo *Agent*, seria a solicitação de dados do CoordinatorAgent para o IntegraAgent e GoogleAgent os quais apresentam comportamento de espera por uma solicitação de dados do CoordinatorAgent.

Na parte inferior esquerda, se encontra a fila de mensagens de um agente onde ficam armazenadas todas as mensagens recebidas de outros agentes. Esta fila tem um papel fundamental na comunicação entre agentes, pois ela organiza a ordem com que as mensagens chegam facilitando assim a comunicação.

Ao centro, tem-se o escalonador de comportamentos e o gerenciador de ciclo de vida do agente. O primeiro, é o responsável por escalonar a ordem de execução dos comportamentos determinando se eles seguem alguma ordem de precedência ou não. O gerenciador de ciclo de vida

é o controlador do estado atual do agente (inicializado ou destruído no caso do módulo). Assim que um agente muda de estado o gerenciador se encarrega de realizar todas as tarefas necessárias para que a mudança de estado do agente seja bem sucedida (inicializa ou destrói variáveis, serviços etc).

Na parte direita da figura 16 tem-se os recursos que o agente utiliza que são dependentes da aplicação (acesso a serviços, dados da aplicação). Nesse componente, são armazenados também o conhecimento e capacidades que o agente adquiriu durante execução da aplicação ou já possuía previamente.

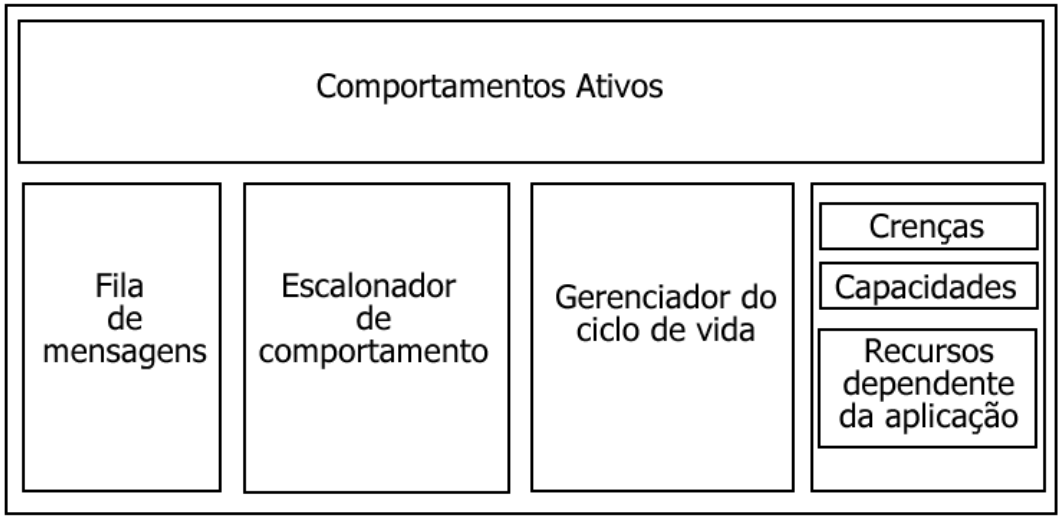

Figura 16: Arquitetura dos agentes

4.2.4. Modelagem dos Agentes

Nesta seção será apresentada parte do diagrama de classes do módulo *Agent*. Mais especificamente, serão mostradas parcialmente as classes referentes aos agentes implementados. O diagrama é ilustrado pela figura 17.

No módulo *Agent*, para que os agentes pudessem realizar algumas de suas respectivas funções tal como acesso a dados de bases externas, foi necessário que eles tivessem acesso a alguns serviços já oferecidos pela Plataforma Integra. Para viabilizar isso, foi criada uma classe *Singleton* no módulo, chamada AgentService, com objetivo de reunir todos os serviços da Plataforma que os agentes necessitavam (padrão *facade*). Desta forma, a classe AgentService serviu de interface entre os recursos da Plataforma Integra e o módulo *Agent*.

Como já foi dito, o módulo proposto contém três agentes que são o IntegraAgent, GoogleAgent e CoodinatorAgent. Esses três agentes possuem uma estrutura em comum diferenciando somente na maneira com que essas estruturas são implementadas. Essa estrutura

comum é herdada da classe AbstractAgent que, por sua vez, utiliza a AgentService visando disponibilizar para suas sub-classes acesso a serviços de forma transparente.

Os comportamentos (ações ou intenções) dos agentes são representados através de *inner classes* (classes inteiramente declaradas dentro do corpo de outra classe). Dentre elas, cita-se:

- RequestIntegraData (*inner class* do CoordinatorAgent): Representa o comportamento responsável por requisitar dados da base de dados intermediária ao IntegraAgent;
- RequestGoogleData (*inner class* do CoordinatorAgent): Representa o comportamento responsável por requisitar dados da base do Google ao GoogleAgent;
- InformationRequested (*inner class* do IntegraAgent): Representa o comportamento que fica aguardando por uma requisição de dados da base de dados intermediária. Após a requisição, o IntegraAgent busca os dados e os retorna para o agente que fez a requisição (CoordinatorAgent);
- InformationRequested (*inner class* do GoogleAgent): Representa o comportamento que fica aguardando por uma requisição de dados da base ao Google. Após a requisição, o GoogleAgent busca os dados e os retorna para o agente que fez a requisição (CoordinatorAgent).

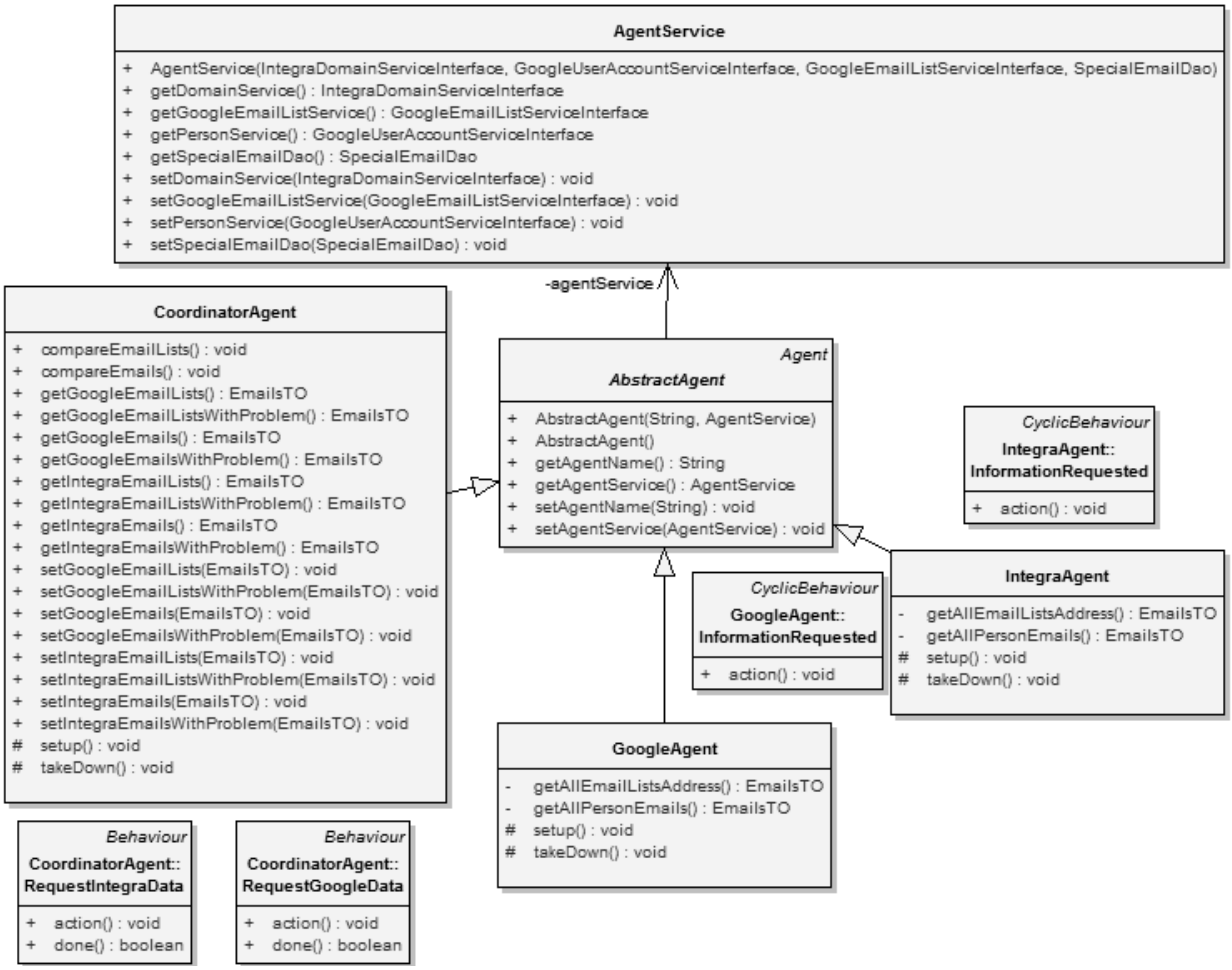

Figura 17: Diagrama de classes – agentes (visão parcial)

4.3. Visualização do Funcionamento

Neste tópico será comentado sobre o funcionamento básico do módulo *Agent* na Plataforma Integra. O módulo sempre inicia seu funcionamento assim que a Plataforma Integra é inicializada no servidor de aplicação onde está hospedada. A figura 18 ilustra o momento da inicialização da Plataforma juntamente com o módulo *Agent*. Para melhor ilustração da inicialização foi utilizada a interface de gerenciamento remoto de agentes fornecida pelo *framework* JADE. Através dessa interface é possível visualizar todos os agentes que estão inseridos na aplicação, além de permitir controle sobre os estados do ciclo de vida dos agentes.

Para visualizar a interação entre os agentes foi utilizado a ferramenta SnifferAgent (JADE). Em síntese, trata-se de uma ferramenta que mostra a troca de mensagens entre agentes de forma similar a um diagrama de seqüências UML (*Unified Modeling Language*). Toda mensagem enviada de um agente para outro é rastreada e disponibilizada em uma interface gráfica para o usuário. O

SnifferAgent é ilustrado pela figura 19 onde, do lado esquerdo encontram-se todos os agentes da aplicação e do lado direito encontra-se a visualização gráfica da troca de mensagens entre os agentes

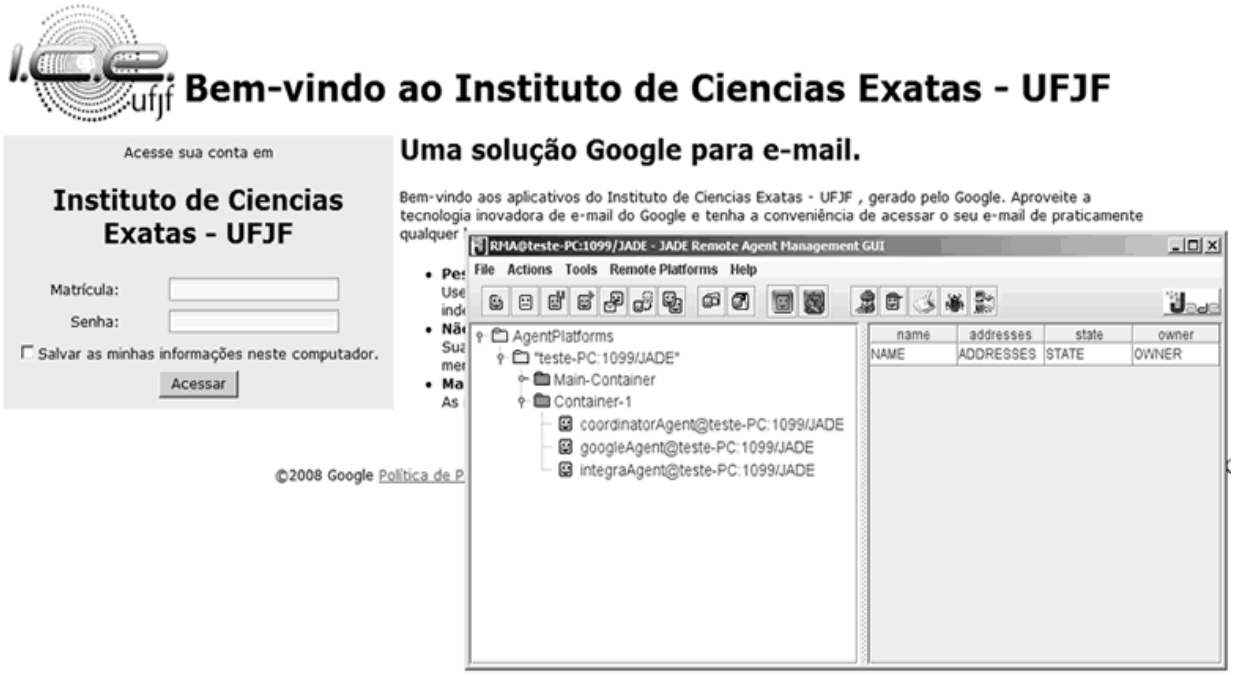

Figura 18: Inicialização da Plataforma Integra

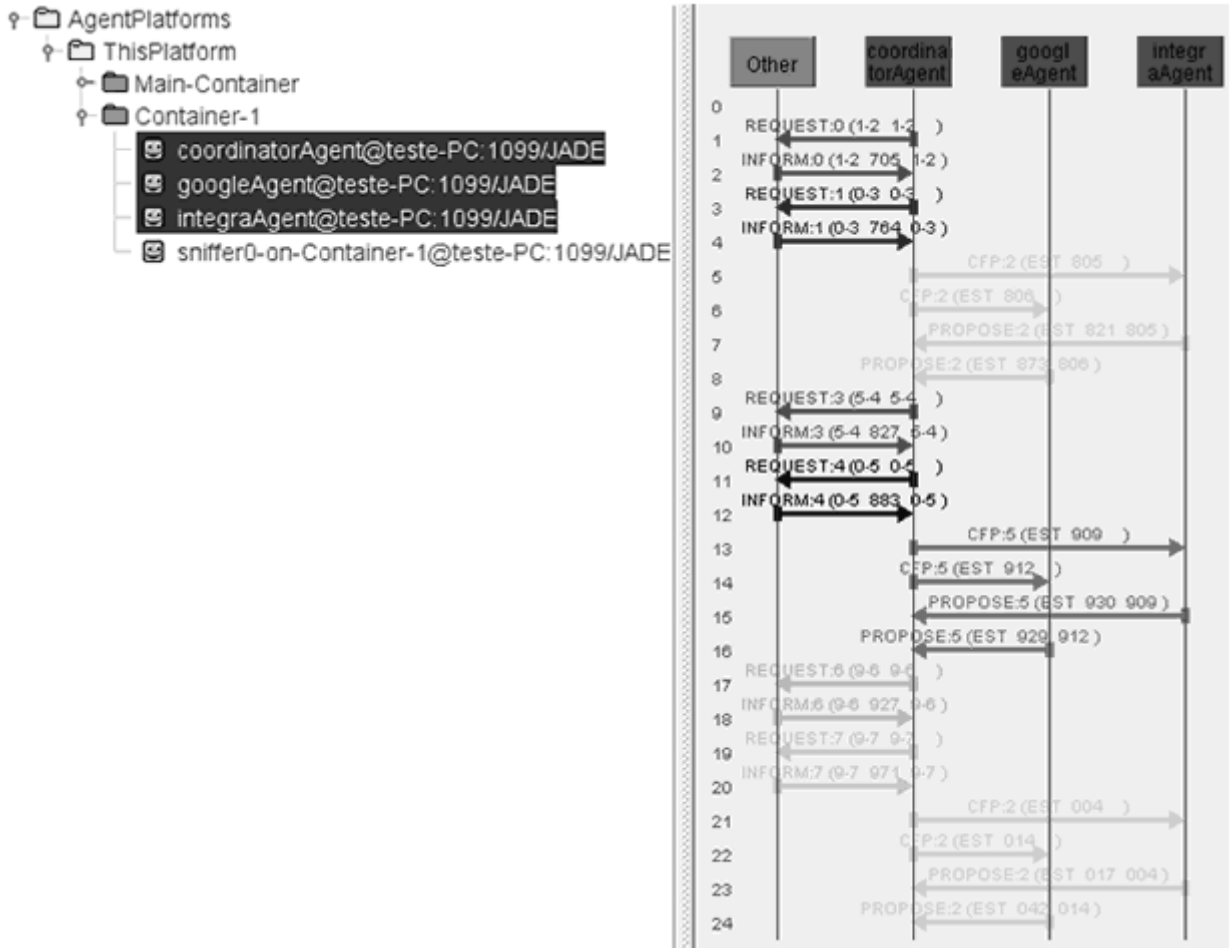

Figura 19: SnifferAgent

4.4. Comentários Finais

Neste capítulo foi apresentado o módulo que contêm os agentes implementados para a Plataforma Integra. A idéia principal do capítulo foi mostrar uma aplicação de agentes de software trabalhando em conjunto, em um sistema em funcionamento, de forma a possibilitar que o objetivo principal seja alcançado mais facilmente. No caso da Plataforma Integra, o uso de agentes, serviu como um mecanismo que visa identificar falhas (falta de sincronização entre as bases de dados) e resolvê-las de forma transparente para o usuário. Como vantagem principal dessa aplicação de agentes pode-se citar maior robustez da Plataforma, um vez que, com o módulo *Agent* ela possui um componente que se encarrega de automaticamente procurar e tratar falhas.

5. Conclusões

5.1. Considerações Finais

Neste trabalho foi abordado o tema de agentes de software de maneira a expor suas diversas definições e conceitos relacionados. Entre essas definições e conceitos foram comentados aspectos tais como características que agentes normalmente possuem, classificações e detalhes sobre a comunicação entre agentes. Toda essa revisão conceitual serviu como base para a confecção dos agentes propostos de forma a nortear o estudo de caso apresentado.

Atualmente vê-se muitas aplicações de agentes de software, porém, várias delas tratam-se de algoritmos isolados que resolvem apenas um determinado tipo ou classe de problema. O objetivo deste trabalho foi mostrar uma aplicação de agentes em um sistema real, de uma instituição de ensino, já em utilização por uma quantidade considerável de usuários que é a Plataforma Integra.

A Plataforma Integra, por sua vez, é uma aplicação que une regras de negócios e dados de sistemas distintos abstraindo a complexidade dessa união. Em cima do contexto da Plataforma que nasceu a idéia do módulo *Agent* e do estudo de caso visando mostrar a utilidade do uso de agentes em sistemas em funcionamento.

O estudo de caso teve como meta principal ilustrar como a colaboração entre agentes pode ser valiosa em sistemas (caracterizando um sistema multi-agente). Foram apresentados detalhes técnicos da implementação tais como arquitetura dos agentes, padrões de projeto, diagramas UML para mostrar de maneira clara como foi construído o módulo.

É importante ressaltar o ganho que um sistema pode ter com o uso de agentes principalmente quando estes resolvem problemas de forma autônoma. Uma vez identificada uma situação cuja solução possa ser construída através de agentes de software, não há dúvidas que será uma solução interessante no sentido que agentes resolvem problemas por si próprios não necessitando de intervenção externa. No caso da Plataforma Integra, a idéia de se ter componentes de software agindo independentemente de influências externas, e ainda, com objetivo de apresentar soluções para possíveis situações imprevistas, explicita bem o ganho do software com relação à sua confiabilidade, robustez e, conseqüentemente, qualidade.

5.2. Trabalhos Futuros

Apesar de todos os elementos apresentados que compõem o módulo *Agent* (conceitos, arquitetura de software, padrões de projeto) sejam bastante conhecidos e maduros, o trabalho

possui um caráter inovador por apresentar uma aplicação diferente de agentes das apresentadas na literatura. É possível destacar alguns trabalhos futuros a partir da base mostrada.

O primeiro é a ampliação do módulo em relação aos agentes implementados para contemplar outros tipos de agentes tal como aqueles que auxiliam um usuário durante o uso do software. Pode-se também estender a cognição dos agentes já existentes, uma vez que, o módulo *Agent* apenas resolve falhas, mas não as reporta para o usuário. Uma idéia seria criar um repositório de conhecimento onde seriam armazenados as falhas identificadas de forma a possibilitar que um usuário, em um momento oportuno, seja informado sobre a existência delas através de agentes de interfaces.

Pode ser explorada também a criação de agentes que possam realizar algum tipo de estudo estatístico como a freqüência em que ocorrem falhas na Plataforma ou em quais partes do sistema ocorrem mais falhas. Esses estudos estatísticos poderiam ser a base para inferir uma medida sobre o grau de confiabilidade do sistema, por exemplo.

6. Referências

1. GARCIA, A. et al. Software Engineering for Large Scale Multi-Agent Systems. Lecture Notes in Computer Science, Springer, No prelo, 2003.
2. GARCIA, A. et al. Software Engineering for Large-Scale Multi-Agent Systems – SELMAS 2002. (Post-Workshop Report) ACM Software Engineering Notes, August 2002.
3. BRADSHAW, J. An Introduction to Software Agents. In: BRADSHAW, J. (Ed.). Software Agents. Massachussets: MIT Press, 1997.
4. GENESERETH, N.; KETCHPEL, S. Software Agents. Communications of the ACM, v. 37, n. 7, Julho 1994.
5. JENNINGS, N.; SYCARA, K.; WOOLDRIDGE, M. A Roadmap of Agent Research and Development. Journal of Autonomous Agents and Multi-Agent Systems, v. 1, n. 1, 1998, p. 7-38.
6. JENNINGS, N. An Agent-based Approach for Building Complex Software Systems. Communications of the ACM, v. 44, n. 4, 2001, p. 35-41.
7. MINSKY, N.; UNGUREANU, V. Law-Governed Interaction: A Coordination and Control Mechanism for Heterogeneous Distributed Systems. ACM Transactions on Software Engineering and Methodology, v. 9, n. 3, Julho 2000, p. 273-305.
8. OMICINI, A.; PETTA, P.; TOLKSDORF, R. Engineering Societies in the Agents World II. In: International Workshop, 2, 2001, Praga, República Tcheca. Anais.
9. RIPPER, P. et al. V-Market: A Framework for e-Commerce Agent Systems. World Wide Web, v.3, n.1, 2000.
10. GUTTMAN, R.; MOUKAS, A.; MAES, P. Agent Mediated Electronic Commerce: A Survey. Knowledge Engineering Review, Junho 1998.
11. CUNHA, L. M.; FUKS, H.; LUCENA, C. Sistemas Multi-agente e Instrução Baseada na Web. In: International Conference on Engineering and Technology Education, 2002, Santos. Anais, 17-20.
12. JENNINGS, N.; WOOLDRIDGE, M. Applications of Intelligent Agents. In: Agent Technology: Foundations, Applications, and Markets, Springer-Verlag, Heidelberg, Germany, 1998, p. 3-28.

13. CABRI, G.; LEONARDI, L.; ZAMBONELLI, F. MARS: a Programmable Coordination Architecture for Mobile Agents. IEEE Internet Computing, v.4, n.4, 2000.

14. CABRI, G.; LEONARDI, L.; ZAMBONELLI, F. Mobile-agent Coordination Models for Internet Applications. IEEE Computer Magazine, v.33, n.2, 2000.

15. CABRI, G.; LEONARDI; ZAMBONELLI, F. The Impact of the Coordination Model in the Design of Mobile Agent Applications. In: Computer Software and Applications Conference, 22, 1998, Viena, Áustria. Anais.

16. DORIGO, M.; MANIEZZO, V.; COLORNI, A. The Ant System: Optimization by a colony of cooperating agents. IEEE Transactions on Systems, Man, and Cybernetics, v. 26, n.1, 1996, p. 29-41.

17. BROWN, S.; SANTOS, E.; BANKS, S.; STYTZ, M. Intelligent interface agents for intelligent environments. In: AAAI Spring Symposium on Intelligent Environments, 1998, Stanford, EUA. Anais.

18. LASHKARI, Y.; METRAL, M.; MAES, P. Collaborative Interface Agents. In: Twelfth National Conference on Artificial Intelligence, AAAI Press, Seattle, WA, 1994. Anais.

19. NWANA, H. Software Agents: An Overview. Knowledge Engineering Review, v. 11, n. 3, 1996, p. 1-40.

20. WOOLDRIDGE, M.; CIANCARINI, P. Agent-Oriented Software Engineering: The State of the Art. In: P. CIANCARINI AND M. WOOLDRIDGE (Eds.). Agent-Oriented Software Engineering. Springer-Verlag, Lecture Notes in AI, 2001.

21. ZAMBONELLI, F.; PARUNAK, H. Signs of a Revolution in Computer Science and Software Engineering. In: International Workshop Engineering Societies in the Agents World, 3, 2002, Madri, Espanha. Anais.

22. SHOHAM, Y. Agent-Oriented Programming. Artificial Intelligence, v. 60, 1993, p.24-29.

23. GARCIA, A. et al. Software Engineering for Large-Scale Multi-Agent Systems: A Roadmap. In: A. GARCIA, C. LUCENA, J. CASTRO, A. OMICINI, F. ZAMBONELLI (Eds). Software Engineering for Large-Scale Multi-Agent Systems. Springer, LNCS, No Prelo, 2003.

24. KARNIK, N.; TRIPATHI, A. Design Issues in Mobile-Agent Programming Systems. IEEE Concurrency, v. 6, n. 3, 1998, pp.52-61.

25. BELLIFEMINE, Fabio; CAIRE, Giovanni; TRUCCO Tiziana; RIMASSA Giovanni. Developing multi-agents system with JADE, 2003.
26. RUSSEL, Stuart J; PETER Norvig. Artificial Intelligence: A Modern Approach. Englewood Cliffs, Nova Jersey: Prentice Hall, 1995.
27. AYRES, Leonardo; Estudo e Desenvolvimento de Sistemas Multi-agentes usando JADE: Java Agent Development framework, 2003. Trabalho de conclusão para obtenção do grau de Bacharel.
28. FERBER, Jacques. Multi-Agent Systems: An Introduction to Distributed Artificial Intelligence, Addison-Wesley Pub Co; 1st Edition, February 1999.
29. BAKER, Albert. JAFMAS – A java-based agent framework for multiagent systems. Development and Implementation. Cincinnati: Department of Electrical & Computer Engineering and Computer Science University of Cincinnati, 1997. Doctoral thesis.
30. HILTZ, S. R. The Virtual Classroom: Learning without limits via computer network, Norwood, New Jersey: Ablex Publishing Corporation, 1994.
31. TUROFF, M.; HILTZ, S. R. Computer Support for Group versus Individual Decisions, IEEE Transactions on Communications, 1982, 30, (1), pp 82-91.
32. BENBUNAN-FICH R.; HILTZ, S. R. Impacts of Asynchronous Learning Networks on Individual and Group Problem Solving: A Field Experiment, Group Decision and Negotiation, 1999, Vol.8, pp. 409-426.
33. SCHÖN, D. A. The reflective practitioner: How professionals think in action, Basic Books, New York. 1983.
34. GAMMA, E. et. al. Design Patterns – Elements of Reusable Object-Oriented Software, Addison-Wesley Profissional Computing Series. Addison-Wesley Publishing Company, New York, NY, 1995.
35. GEROSA, M. A.; FUKS, Hugo.; LUCENA, Carlos José Pereira de. Elementos de percepção como forma de facilitar a colaboração em cursos via Internet, In: XII Simpósio Brasileiro de Informática na Educação - SBIE 2001, 2001, Vitória, ES. Anais do XII Simpósio Brasileiro de Informática na Educação, 2001. p. 194-202.
36. FOWLER, Martin. Patterns for Enterprise Application Architecture, Addison-Wesley, 2002.
37. GOOGLE. 2008. "SAML reference implementation". Disponível em: <http://code.google.com/apis/apps/sso/saml_reference_implementation.html>. Acesso em: abril de 2008
38. PROCACI, T. B. et. al. Projeto Integra: Uma Proposta de Desenvolvimento de Um Ambiente de Colaboração Integrado ao SIGA e aos Serviços Oferecidos pelo Google, In II

Workshop de Trabalhos de Iniciação Científica e de Graduação da Faculdade Metodista Granbery, 2008, Juiz de Fora – MG.

39. PRESSMAN, Roger S. Engenharia de Software, Makron, 6.ed, 2004.
40. FARIAS, A. F. ; ALMEIDA, V. C. ; SILVA, A. F. . Algoritmo para recuperação de falhas em sistemas de gerenciamento de workflow. In: XVI Seminário de Computação, 2007, Blumenau. Anais do XVI SEMINCO - Seminário de Computação. Blumenau : Universidade Regional de Blumenau, 2007. p. 31-38.
41. BRITO, P. H. Um método para modelagem de exceções em desenvolvimento baseado em componentes, 2005. Dissertação de mestrado – UNICAMP.
42. JAVA. 2008. "Java Platform Enterprise Edition (Java EE)". Disponível em: http://java.sun.com. Acesso em: junho 2008.
43. JADE. 2008. "Java Agent DEvelopment Framework ". Disponível em: http://jade.tilab.com. Acesso em: junho 2008.
44. SPRING. 2008. "The Leading Full-Stack Java/JEE Application Framework " Disponível em: http://www.springframework.org. Acesso em: junho 2008.
45. HIBERNATE. 2008. "Relational Persistence for Java and .NET" Disponível em: http://www.hibernate.org. Acesso em: junho 2008.
46. MYSQL. 2008. "The World's Most Popular Open Source Database". Disponível em:http://www.mysql.com . Acesso em: junho 2008.
47. JUNIT. 2008. "A Simple Framework to Write Repeatable Tests". Disponível em: http://www.junit.org. Acesso em: junho 2008.
48. ECLIPSE. 2008. "Eclipse - an open development platform". Disponível em: http://www.eclipse.org. Acesso em: junho 2008.
49. APACHE TOMCAT. 2008. "Servlet container developed by the Apache Software Foundation". Disponível em: http://tomcat.apache.org. Acesso em: junho 2008.
50. SIGA. 2008. "Sistema Integrado de Gestão Acadêmica – Universidade Federal de Juiz de Fora". Disponível em: http://siga.ufjf.br. Acesso em junho 2008.