Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

Confiabilidade em Sistemas Multiagentes

João Paulo de Freitas Matos

Monografia apresentada como requisito parcial
para conclusão do Bacharelado em Ciência da Computação

Orientador
Prof. Célia Ralha

Brasília
2012

Universidade de Brasília — UnB
Instituto de Ciências Exatas
Departamento de Ciência da Computação
Bacharelado em Ciência da Computação

Coordenador: Prof. Marcus Vinícus Lamar

Banca examinadora composta por:

Prof. Célia Ralha (Orientador) — CIC/UnB
Prof. Genaína Nunes — CIC/UnB
Prof. Professor II — CIC/UnB

CIP — Catalogação Internacional na Publicação

Matos, João Paulo de Freitas.

Confiabilidade em Sistemas Multiagentes / João Paulo de Freitas Matos. Brasília : UnB, 2012.
51 p. : il. ; 29,5 cm.

Monografia (Graduação) — Universidade de Brasília, Brasília, 2012.

1. Confiabilidade, 2. Sistemas Multiagentes

CDU 004.4

Endereço: Universidade de Brasília
Campus Universitário Darcy Ribeiro — Asa Norte
CEP 70910-900
Brasília–DF — Brasil

Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

Confiabilidade em Sistemas Multiagentes

João Paulo de Freitas Matos

Monografia apresentada como requisito parcial
para conclusão do Bacharelado em Ciência da Computação

Prof. Célia Ralha (Orientador)
CIC/UnB

Prof. Genaína Nunes
CIC/UnB

Prof. Professor II
CIC/UnB

Prof. Marcus Vinícus Lamar
Coordenador do Bacharelado em Ciência da Computação

Brasília, 12 de dezembro de 2012

Dedicatória

Dedico a....

Agradecimentos

Agradeço a....

Abstract

A ciência...

Palavras-chave: Confiabilidade, Sistemas Multiagentes

Abstract

The science...

Keywords: Reliability, Multiagent systems

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

Com o crescente desenvolvimento da computação que penetra cada vez mais em diversas áreas de conhecimento, a demanda por processamento tende a crescer rapidamente, tornando o desenvolvimento de aplicações cada vez mais complexo, exigindo cada vez mais desempenho e consequentemente poder de processamento. Para a execução dessas aplicações em tempo hábil, são necessários investimentos cada vez mais altos em *hardwares* melhores, existindo porém um fator limitante (custo ou tecnologia).

Além disso, atualmente a grande quantidade de informação disponível exige análises cada vez mais precisas e detalhadas da informação processada tendo em vista a ajuda de tomada de decisões. Dessa forma, aplicações que antes eram centralizadas em uma única máquina transformaram-se em aplicações distribuídas em várias máquinas que são concorrentes e assíncronas.

A partir dessa motivação, aplicações são projetadas para rodar de forma descentralizada, com componentes e serviços rodando em diversos lugares distintos e comunicando-se uma com as outras através de mensagens. Cada módulo pode ter um objetivo específico, como capturar e processar eventos no ambiente computacional, processar informações, persistir eventos no banco de dados, enfim, uma vasta gama de operações que são assíncronas e independentes. A arquitetura dessas aplicações é projetada objetivando o alto paralelismo, flexibilidade, interoperabilidade, dentre outros aspectos.

Diversos *frameworks* tentam lidar com o problema da computação distribuída, porém alguns usam arquiteturas que são baseadas em processamento ordenado: Os dados são processados ordenadamente, não usufruindo de todo o processamento que poderia ocorrer se fosse realmente paralelo. Outros *frameworks* são embasados em tecnologias que não são recomendáveis em um ambiente distribuído: O uso de recurso um compartilhado e blocante, que pode prejudicar o processamento em larga escola.

A computação distribuída toma formas ainda mais interessantes quando aplicada a contextos sensíveis a sociedade em geral. Áreas de atuação como Bolsa de Valores, Análise de Redes Sociais, Análise de Mídia Social, Informática na Educação, dentre outras. Em especial a esta última área, a possibilidade de auxílio no aprendizado do aluno eleva a importância deste setor na computação.

Na perspectiva da Informática na Educação (IE), a abordagem chamada Sistemas Tutores Inteligentes permite a representação de conhecimento de forma muito interessante. É possível a construção um modelo onde se representa o estudante (o objeto a quem se deve ensinar), o domínio do conteúdo (o conteúdo a ser ensinado) e o modelo peda-

gógico (a forma a ser ensinada). Esta abordagem permite o uso de vários conceitos da Inteligência Artificial para determinação de estilos de aprendizagem, dentre outras possibilidades. Dessa forma, aplicações tendem a ser bastante complexas vistas ao alto grau de processamento que este ambiente pode assumir.

1.1 Problema

Os ambientes educacionais de aprendizagem não possuem inferência de modelo do estudante, pois sua implementação seguindo modelo cliente-servidor não é apropriada para tal finalidade. Além disso, a alta carga da aplicação devido ao fato de muitos acessos dos alunos pode prejudicar uma aplicaçao centralizada, necessitando de um sistema que processasse em múltiplos locais.

1.2 Objetivos do Projeto

Tendo em vista o cenário atual apresentado, o objetivo geral deste trabalho é propor a arquitetura distribuída de um ambiente computacional, bem como sua construção, baseada em Sistemas Multiagentes, capaz de construir e manter um modelo do estudante, a partir do qual os estilos de aprendizagens desse estudante poderão ser identificados e informados ao docente por meio de uma interface web.

Os objetivos específicos serão devidamente justificados na próxima subseção, bem como idealizada a forma de atingi-los.

Usando tecnologias existentes e consolidadas, a arquitetura proposta irá usar o framework *JADE*, que é completamente desenvolvido na linguagem *JAVA* e simplifica a implementação de Sistemas Multiagentes (SMA) que cumprem as especificações FIPA. A arquitetura proposta também englobará uma interface web que utiliza a plataforma *open source JBoss Seam*, desenvolvida para auxiliar a construção de aplicações dinâmicas para a internet de forma simples e ágil.

1.3 Objetivos Específicos do Projeto

Este trabalho foi norteado pelo documento do projeto Frank [**?**], sendo construído como uma proposta ao problema nele proposto. A partir deste documento, é possível identificar os objetivos específicos derivados da construção do ambiente computacional para manutenção do modelo do aluno. Mais especificamente, os objetivos específicos deste trabalho são:

- Objetivo específico 1: Obter uma modelagem da arquitetura geral do SMA Frank utilizando-se da metodologia *Multiagent System Engineering* (MASE), proposta como uma solução de Engenharia de Software para o desenvolvimento de SMA;
- Objetivo específico 2: Obter uma implementação da arquitetura geral do SMA Frank;
- Objetivo específico 3: Obter uma implementação da arquitetura do agente assistente de cognição;

- Objetivo específico 4: propor uma interface do agente assistente de cognição com o estudante;
- Objetivo específico 5: propor uma interface do agente assistente de cognição com o docente;

A seguinte estrutura desse trabalho consiste na divisão de capítulos visando facilitar a leitura e organizar os conceitos que perfazem o desenvolvimento deste trabalho:

- O presente capítulo contém a introdução, onde o trabalho é justificado e os objetivos são esclarecidos.
- Capítulo 2 contém todos os fundamentos teóricos necessários para o desenvolvimento desse trabalho.
- Capítulo 3 contém a proposta de solução composta pela metodologia, modelagem da arquitetura e implementação.
- Capítulo 4 contém a experimentação e análise de resultados.
- Por fim, o capítulo 5 relata a conclusão e trabalhos futuros.

Capítulo 2

Fundamentos básicos

Este capitulo apresenta os principais conceitos e definições necessários para o entendimento deste trabalho. A seção 2.1 apresenta alguns conceitos básicos em *Unified Modeling Language* (UML), que são necessários para o entendimento da modelagem deste trabalho. A seção 2.2 disserta sobre conceitos a respeito da informática na educação. A seção 2.3 aborda a teoria sobre Sistemas Multiagentes necessária para este trabalho. A seção 2.4 contém a metodologia *Multiagent System Engineering*, desenvolvida para a criação de Sistemas Multiagentes A seção 2.5 e 2.6 abordam o funcionamento dos frameworks JADE e Jboss Seam, respectivamente. Por fim, a seção 2.7 detalha alguns trabalhos correlatos.

2.1 Unified Modeling Language

A Linguagem Unificada de Modelagem, *Unified Modeling Language* (UML) é uma linguagem visual que foi desenvolvida para a representação do software por meio de imagens, objetivando o entendimento dos artefatos de forma rápida e clara e resultando em uma semântica para o projeto em questão. De acordo com [3] o UML faz parte de uma família de notações gráficas que ajudam na descrição e concepção de sistemas de software, principalmente em sistemas concebidos utilizando o paradigma da orientação à objetos (OO).

O UML é um padrão não proprietário, controlado pelo consórcio *Object Management Group* [1]. Seu nascimento é datado em 1997 [3], surgindo a partir da união de diversas linguagens e ferramentas de surgiram na década de 80 e 90. A linguagem ajuda o entendimento de como o software foi projetado, como ocorre a comunicação entre seus objetos, como suas classes são organizadas, quais são os atores que são envolvidos na utilização do software, dentre outras possibilidades de representação.

Em [3], é possível separar o uso do UML de três formas distintas, diferindo entre as modelagens utilizadas e o objetivo de uso. As três formas são: Rascunho, planta de software e como linguagem de programação.

A utilização como rascunho é utilizada para facilitar a comunicação entre as pessoas envolvidas no projeto, sejam desenvolvedores discutindo funcionalidades do software ou gestores explicando funcionalidades em alto nível. O objetivo neste uso é a comunicação de alguns aspectos do sistema de forma rápida, sem a necessidade de formalizar artefatos para o projeto.

A utilização do UML como planta de software são documentos detalhados que são criados para documentação do software, sendo divida em duas sub-categorias: Engenharia reversa e engenharia avante. Na engenharia reversa, os diagramas são gerados a partir de uma ferramenta que faz a leitura do código fonte e gera os diagramas desejados, que são utilizados para auxiliar o leitor no entendimento do sistema. Na engenharia avante, a idéia é modelar o sistema detalhadamente antes de qualquer desenvolvimento, prevendo quais serão os módulos do sistema, bem como a sua a comunicação.

No uso como linguagem de programação, o UML é utilizado para geração de código executável por ferramentas avançadas de modelagem. Esse modo requer a modelagem de estado e comportamento do sistema, para fins de detalhar todo o comportamento e lógica do sistema em código.

2.1.1 Diagramas UML

O UML 2 descreve 13 tipos de diagramas que podem ser categorizados por estruturais e comportamentais. A imagem 2.1 ilustra essa categorização de diagramas.

Apesar da grande quantidade diagramas envolvidos no UML, nem todos os processos de desenvolvimentos de software utilizam todos eles. A maioria das pessoas utiliza-se de um conjunto de poucos diagramas para modelagem do sistema. Nas próximas subseções, serão detalhados apenas os seguintes diagramas, que são necessários para o entendimento deste trabalho:

- Diagrama de Caso de uso
- Diagrama de sequência

Diagrama de Caso de Uso

Casos de uso são relatos textuais que são utilizados para descobrir e descrever os requisitos do sistema. Consiste da descrição de como um ator utiliza uma funcionalidade do sistema para atingir algum objetivo relacionado. De acordo com [4], os casos de uso devem ser prioritariamente desenvolvidos de forma textual e o seu respectivo diagrama deve ser desenvolvido de forma secundária, somente para ilustrar o relato textual.

Um dos objetivos do caso de uso é a facilidade do levantamento dos requisitos, tanto para os analistas de um sistema, quanto para os clientes envolvidos. A definição de uma modelagem em comum facilidade entre as partes faz do caso de uso uma boa maneira de simplificar o entendimento do comportamento do sistema [4], bem como envolver todos as partes interessadas do sistema(*stakeholders*) na construção do mesmo. De acordo com [2], o caso de uso é um contrato de como será o comportamento do sistema. Este contrato será feito por meio dos atores que existirão, da sua interação com o sistema, bem como os cenários existentes.

Duas definições fazem-se necessárias para o entendimento do caso de uso. A primeira delas é o "Ator"do caso de uso. Ele é um objeto com um comportamento definido no sistema. É possível definir o ator como uma pessoa, organização ou mesmo o próprio sistema (quando utiliza serviços do próprio sistema), desde que tenham sempre um papel relacionado. Existem três tipos de atores relacionados ao sistema:

- Ator Principal: Seus objetivos são satisfeitos por meio da utilização do sistema.

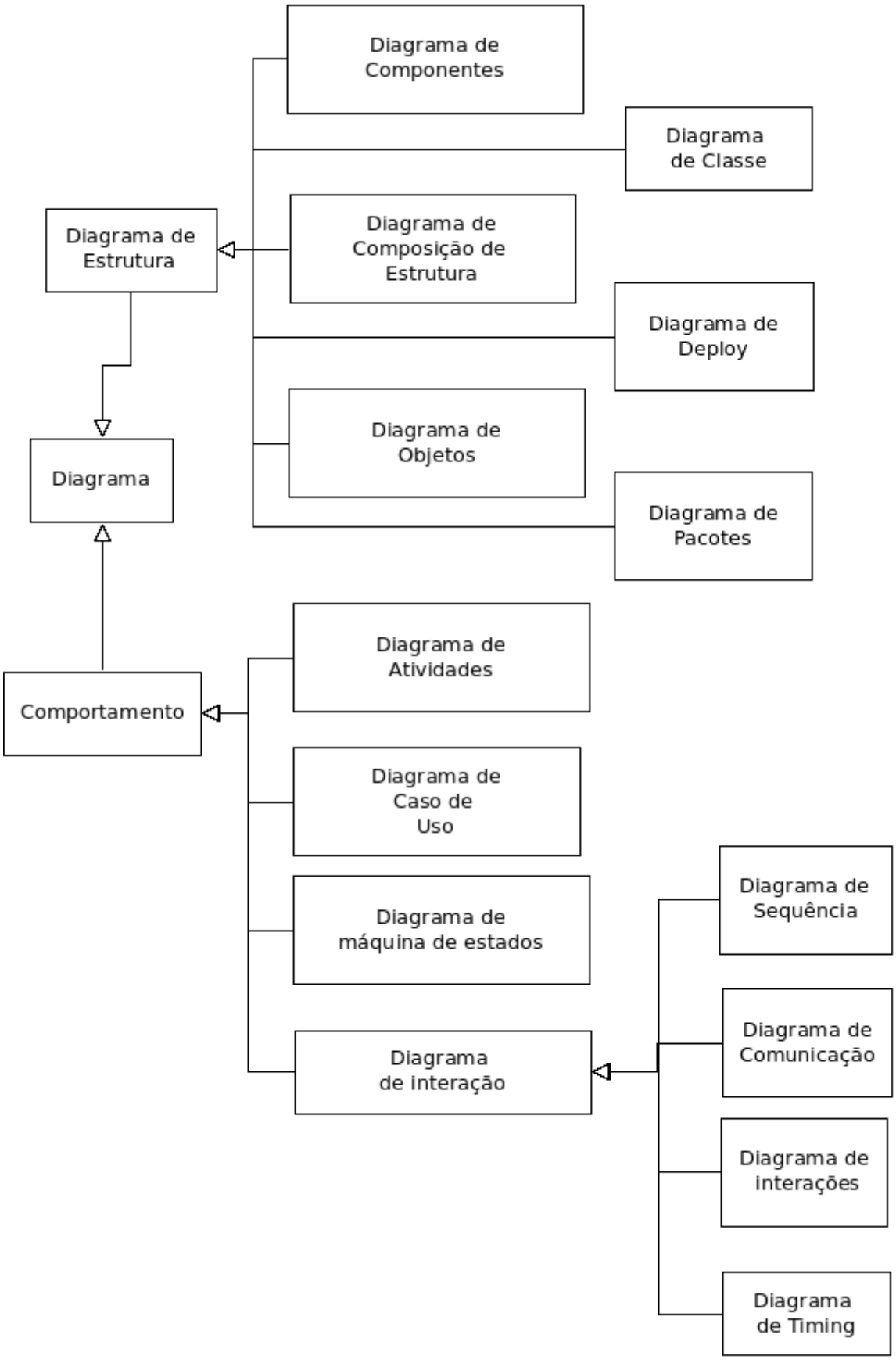

Figura 2.1: Diagramas categorizados por Estrutura e Comportamento. Fonte [3].

Tabela 2.1: Estruturação Detalhada de Caso de Uso

Seção do Caso de Uso	**Significado**
Nome do Caso de Uso	Nome do caso de uso, iniciando-se com um verbo
Escopo	Escopo descrito pelo caso de uso
Nível	Podem ser níveis de objetivo de usuário (quando descrevem os cenários para atingir o objetivo do usuário) ou nível de subfunção (subpassos para dar suporte a um objetivo de usuário)
Ator Principal	O ator que procura os serviços para atingir seus objetivos
Interessados e Interesses	Significado
Pré-Condições	Condições que antecedem o caso de uso e são necessárias para atingir os objetivos
Garantia de Sucesso	Objetos que podem ser analisados após a execução do caso de uso a fim de validar a correta execução do sistema
Cenário de Sucesso Principal	Chamado também de fluxo básico, este cenário descreve o fluxo principal do sistema que satisfaz os interesses dos interessados.
Extensões	Chamado também de fluxos alternativos, são fluxos auxiliares ou cenários de erros que são relacionados ao cenário de sucesso principal
Requisitos Especiais	Registram requisitos não funcionais do sistema e que estão relacionados com o caso de uso
Lista de Variantes Tecnologias de Dados	Listagem de dificuldades técnicas, desafios técnicos que valem a pena registrar no caso de uso
Frequência de ocorrência	Frequência de ocorrência deste caso de uso

- Ator de Suporte: Fornece algum serviço para o sistema.
- Ator de Bastidor: Expressa algum interesse pelo comportamento do caso de uso.

A segunda definição envolvida é a de cenário. Um cenário é uma sequência de interações entre os atores e o sistema. Os cenários são separados por ações de interesses de atores. Logo o caso de uso pode ser considerado como um conjunto de cenários de interações de atores com o sistema.

Dessa forma, o caso de uso deve deixar claro os requisitos funcionais do sistema, bem como o seu comportamento. Existem três formas de se escrever um caso de uso, diferindo em seu nível de formalidade e formatos: Resumido, informal e completo. Este trabalho usará o nível de caso de uso completo, devido ao fato de ser estruturado e mostrar mais detalhes. Os casos de uso são estruturados de diversas formas de acordo com a literatura, sendo a mais famosa [4] a estrutura utilizada por Alistar Cockburn [2] e detalhada na tabela 2.1

A diagramação do caso de uso por UML é uma forma de representação do sistema, mostrando fronteiras do sistema, comunicação e comportamento entre os atores. A figura 2.2

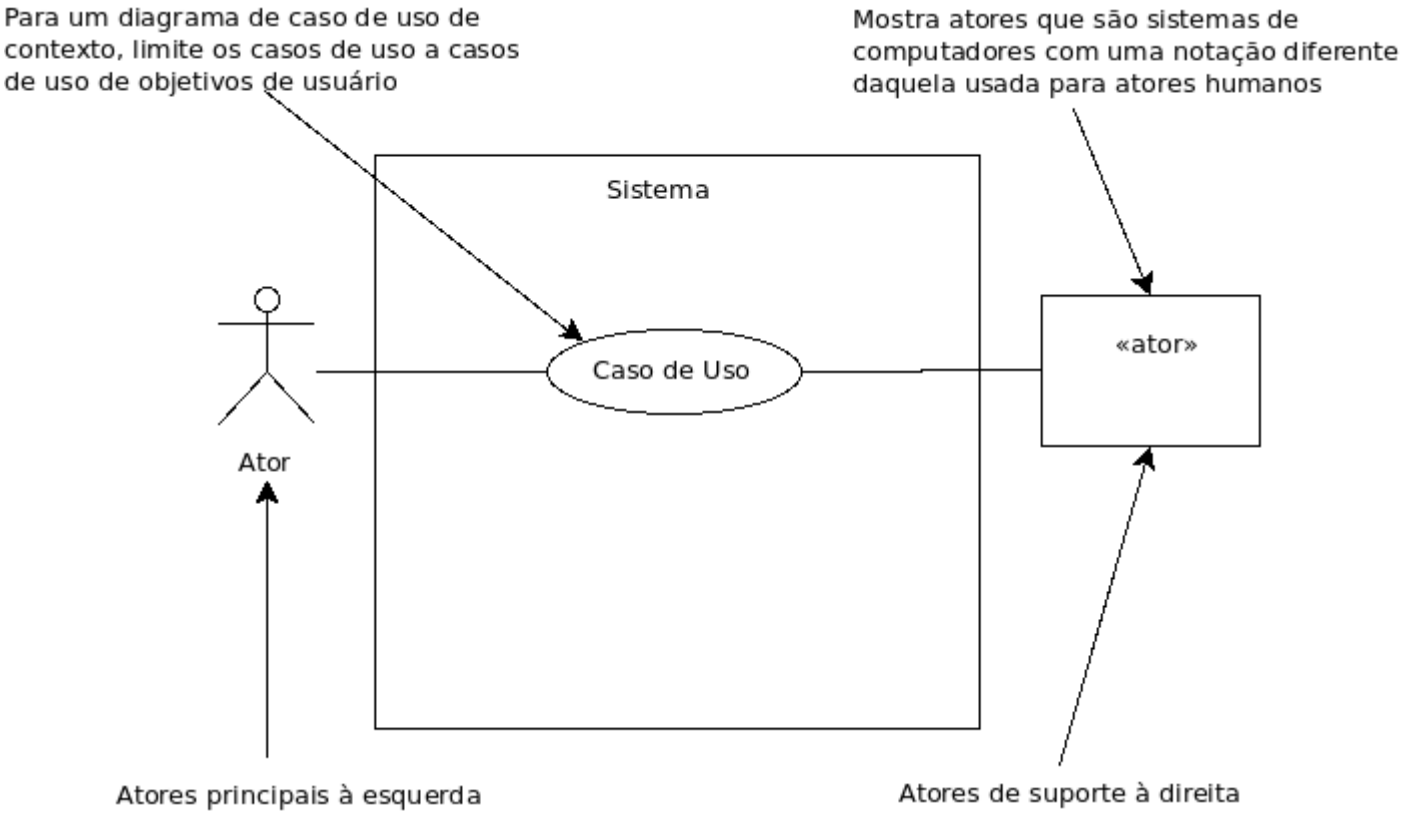

Figura 2.2: Sugestões de notação de caso de uso proposto por [4]

mostra a sugestão de diagramação do caso de uso, propondo forma de representação de atores, casos de uso e atores auxiliares.

Diagrama de Sequência

O Diagrama de Sequência é um documento criado para ilustrar os eventos descritos em um caso de uso de forma sequencial e temporal, mostrando a interação de atores externos ao sistema e os eventos que eles geram durante essa interação. A UML possui uma notação específica para este diagrama, onde todos os elementos são representados.

Neste diagrama, são representados para cada cenário do caso de uso os eventos que os atores geram, bem como a ordem da sua interação. No diagrama os atores são representados na parte superior (com a mesma notação do diagrama de caso de uso). Abaixo dos atores é apresentada a linha de tempo, crescendo de cima para baixo.

Durante a interação do ator com o sistema, eventos de sistema são gerados e iniciam toda a execução do cenário do caso de uso, ou operação do sistema. A execução dos eventos ocorre até o último evento cronológico na linha do tempo. Os eventos gerados pela interação entre os atores ocorrem na linha do tempo de forma cronológica e ordenada com a mesma ordem dos eventos do cenário do caso de uso.

A nomenclatura dos eventos deve sempre iniciar com um verbo, podendo ser seguida de um substantivo. Além disso, deve-se sempre expressar a nomenclatura em níveis genéricos verbais, nunca detalhando a funcionalidade do sistema.

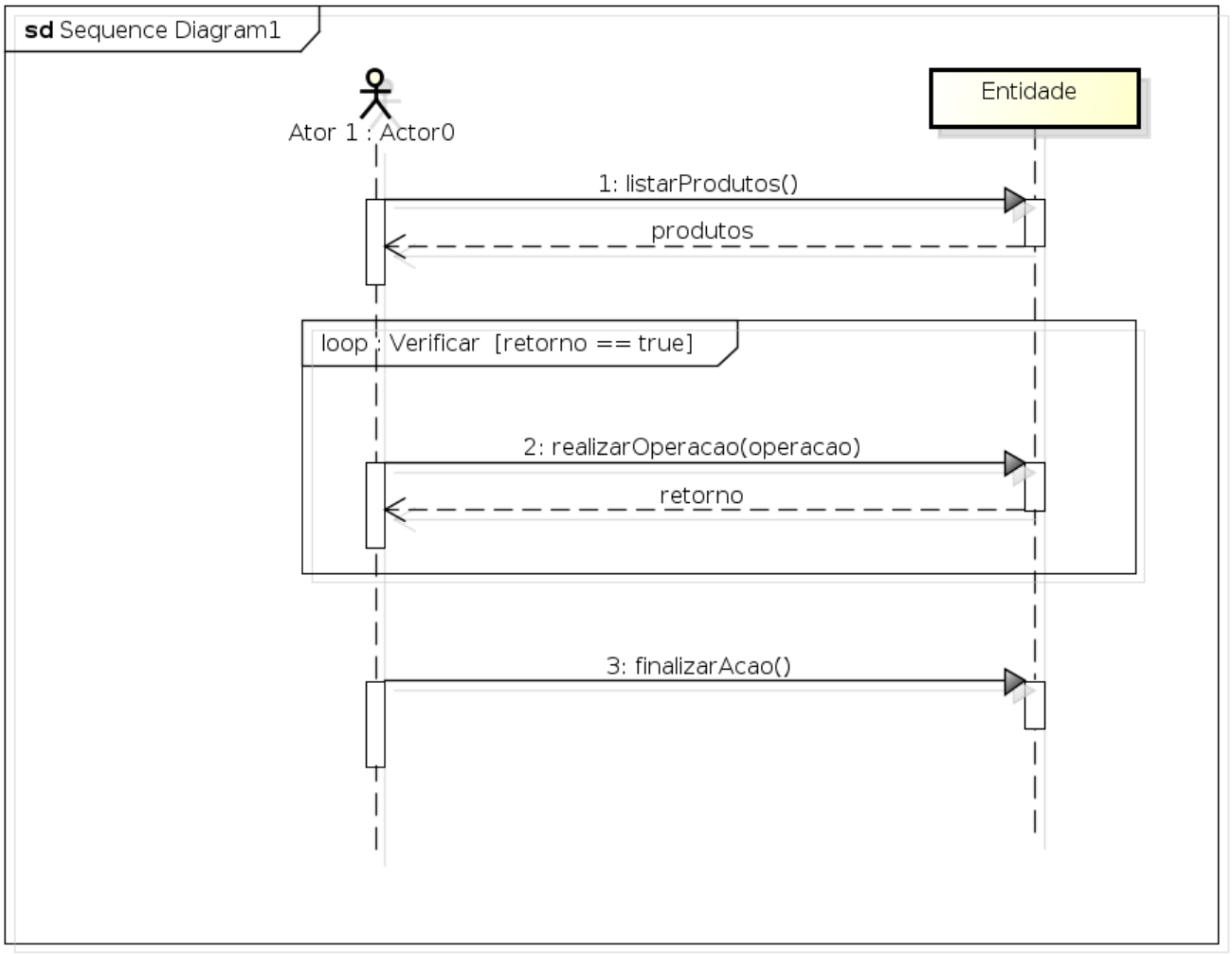

Figura 2.3: Notação de diagrama de sequência, exibindo a comunicação de um ator e uma entidade (sistema)

A imagem 2.3 ilustra a notação de diagramas de sequência com todos os pontos que foram até agora expostos. Nela, é possível identificar a interação entre um ator e uma entidade do sistema. É fácil identificar que os eventos estão ocorrendo de forma ordenada de cima para baixo.

O primeiro evento é iniciado pelo ator, onde o método (listarProdutos()) da Entidade é invocado. Esse método gera a resposta (produtos) para o ator. A interação seguinte acontece dentro de um retângulo, do tipo *loop* e de nome "Verificar". Esse retângulo permite que o diagrama de sequências represente um loop, onde todos os eventos serão repetidos enquanto a condição de guarda for verdadeira, no caso do exemplo, enquanto retorno for igual a *true*. O UML permite diversos operadores além do loop, como por exemplo a negação, a assertiva, o *break*, dentre outros.

Dentro do retângulo, o ator chama o método (realizarOperacao), enviado o parâmetro *operação*. A entidade retorna um valor, que será testado na guarda para a continuidade da conversação. Por fim, o ator chama o método *finalizarAcao* da entidade. Por não haver outra interação em seguida, o cenário é considerado como encerrado.

A importância do desenho de um diagrama de sequência está no fato de detalhar os eventos do sistema que são gerados pela interação de atores externos, pois assim é possível ter uma análise comportamental do sistema com base nesses eventos. Neste nível de análise, é possível estudar e projetar o comportamento do sistema no nível de projetar o que um sistema faz, porém sem necessariamente explicar como o faz **??**.

O diagrama de sequência geralmente está relacionado com o caso de uso, primeiramente pelo fato de descrever um cenário de caso de uso, mas também pelo fato de o caso de uso conter todos os detalhes do cenário. Ele apenas deixará claro a interação entre os atores e os eventos derivados dessa interação.

2.2 Informática na Educação

2.3 Sistemas Multiagentes

Este capítulo visa introduzir o conceito de sistemas multiagentes (SMA). Para tanto é necessário mostrar conceitos que são base para o entendimento de SMA, iniciando pela apresentanção alguns conceitos a cerca de Inteligencia Artificial (IA). Em seguida o trabalho disserta sobre a teoria relacionada à Agente, bem como suas arquiteturas. Só então são apresentados os conceitos de Sistemas Multiagentes (SMA), comunicação em um ambiente SMA e por fim a teoria sobre ontologias.

2.3.1 Inteligência artificial

De acordo com [6] identificamos que a definição de IA pode variar em duas dimensões principais. Usando a definição de sistemas computacionais que agem racionalmente temos:

> *Computational Intelligence is the study of the design of intelligent agents.*

Nessa definição, é importante ressaltar que o agente é uma entidade que atua racionalmente, esperando-se que essa racionalidade e outras características o diferencie de simples programas.

Com o crescimento dos estudos relacionados a este campo a inteligência artificial ganhou várias áreas de atuação, permitindo assim a resolução de diversos desafios relacionados à aplicações modernas. Uma das áreas de atuação é o auxílio na execução de aplicações que resolvem problemas de alta complexidade.

Atualmente várias aplicações podem exigir demais de *hardwares* mais modestos tornando inviável o tempo de execução daquela aplicação, sendo necessário um investimento maior e consequentemente encarecendo o seu valor. Dessa forma outras abordagems fazem-se necessárias, como a distribução da aplicação em vários computadores que dividem a sua execução. Este é o campo de estudo da Inteligência Artificial Distribuída: Sistemas que são compostos por vários agentes coletivos, ou seja, distribuem o trabalho uns com os outros. Cada agente pode possuir uma capacidade diferente, sendo possível realizar a tarefa de modo paralelo.

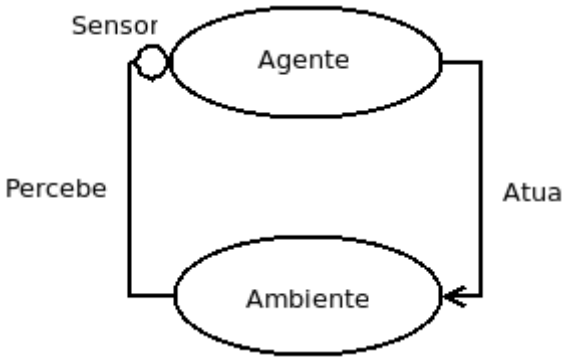

Figura 2.4: Esquematização do funcionamento básico de um agente em um ambiente.

2.3.2 Agente

De acordo com [7], agentes são entidades (reais ou virtuais) que funcionam de forma autônoma em um ambiente, ou seja, não necessitam de intervenção humana para realizar processamento. Esse ambiente de funcionamento do agente geralmente contém vários outros agentes e é possível a comunicação entre eles através do ambiente por meio de troca de mensagens.

Em geral o funcionamento de agentes acontece de forma a perceberem o ambiente em que estão por meio de sensores, fazem análises com base nessa interação inicial e por fim podem agir sobre o ambiente de forma a modifica-lo por meio de efetuadores. A figura 2.4 é uma ilustração do que foi dito.

Alguns agentes seguem o princípio de racionalidade básico: sempre objetivam suas ações pela escolha da melhor ação possível segundo seus conhecimentos. Logo é possível inferir que a ação de um agente nem sempre alcança o máximo desempenho, sendo desempenho o parâmetro definido para medir o grau de sucesso da ação de um agente com base nos seus objetivos. São estes os chamados Agentes Racionais.

Como dito anteriormente, agentes estão presentes em um ambiente. O agente não tem controle total do ambiente, ele pode no máximo influenciá-lo com a sua atuação ou criar outros agentes. Podemos separar ambientes em classes: Software, Físico e Relidade virtual (simulação de ambientes reais em software). De acordo com [9] temos, em geral, ambientes com propriedades inerentes à seu funcionamento:

- Observável: Neste tipo de ambiente, os sensores dos agentes conseguem ter percepção completa do ambiente. Por exemplo, um sensor de movimento consegue ter visão total em um ambiente aberto.
- Determinística: O próximo estado do ambiente é sempre conhecido dado o estado atual do ambiente e as ações dos agentes. O oposto do ambiente determinístico é o estocástico, quando não temos certeza do estado do ambiente. Por exemplo, agentes dependentes de eventos climáticos.
- Episódico: A experiência do agente é dividida em episódios, onde cada episódio é a percepção do agente e a sua ação.
- Sequêncial: A ação tomada pelo agente pode afetar o estado do ambiente e ocasionar na mudança de estado

Tabela 2.2: Listagem de sistemas multiagentes com propriedades de medida de performance, ambiente, atuadores e sensores

Tipo de agente	**Medida de performance**	**Ambiente**	**Atuadores**	**Sensores**
Sensores de estacionamento	Avarias no veículo	Carro e garagens	Freio do carro, controle de velocidade	Sensor de proximidade
Jogos com oponente computador	Quantidade de vitórias	Software	Realizar jogada	Percepção do tabuleiro
Agentes hospitalares	Saúde do paciente	Paciente, ambiente médico	Diagnósticos	Entrada de sintomas do paciente

- Estático: O ambiente não é alterado enquanto um agente escolhe uma ação.
- Discreto: Existe um número definido de ações e percepções do agente para o ambiente em cada turno.
- Contínuo: As percepções e ações de um agente modificam-se em um espectro contínuo de valores. Por exemplo, temperatura de um sensor muda de forma contínua.

Na tabela 2.2 mostramos alguns exemplos de agentes, apresentando as suas características já discutidas nesse trabalho.

A primeira linha da tabela 2.2 apresenta um exemplo de agente atuando em um veículo como um sensor de estacionamento. Responsável por auxiliar o motorista no ato de estacionar o carro, o seu ambiente é da classe físico (considerando o carro e o ambiente onde está o carro). Seu sensor de proximidade é a percepção do ambiente e, caso detecte que está próximo de um obstáculo, pode atuar nos freios dos carros diminuindo a velocidade e evitando colisões. Avarias no carro podem indicar um mal funcionamento do sensor.

A segunda linha da tabela é apresentado o exemplo de agente atuando em um jogo avulso. Esse ambiente é dito dinâmico, pois a cada jogada de um oponente (real ou não), o agente deve analisar a jogada feita pelo seu oponente, irá calcular sua próxima jogada e a realizará. O objetivo principal do agente é a vitória. O ambiente que o agente atua é um software e o seu atuador é um algum mecanismo que permite que ele realize a jogada. O sensor é o mecanismo no qual o agente irá perceber a jogada realizada pelo oponente.

Por fim, a última linha da tabela 2.2 expõe um exemplo de um agente médico atuando em um ambiente estático: Um paciente. Esse ambiente é classificado como estático por não ser alterado pelo agente nesse exemplo, porém é possível ser diferente em outras situações. O objetivo principal é monitorar a saúde do paciente, logo a medida de performance será a aproximação ou não do diagnóstico médico. Seu atuador não será diretamente no ambiente (corpo humano), será na forma de relatórios médicos e seus sensores podem variar de acordo com a doença a ser monitorada.

Conforme podemos encontrar em [9], podemos definir algumas noções gerais de agentes. A primeira, chamada de noção fraca, contém a maior parte dos agentes. Ela compreende os aspectos de *reatividade*, *proatividade* e *habilidade social*. O conceito de reatividade

está ligado com o agente perceber o ambiente e reagir. Proatividade é a característica do agente tomar a iniciativa e agir sem a necessidade de nenhum estímulo. Habilidade social é a capacidade de interação com outros agentes.

Já a noção forte de agente envolve os seguintes aspectos: Mobilidade, veracidade, benevolência, racionalidade e cooperação. As definições são:

- Mobilidade: O Agente deve pode mover-se no ambiente, por exemplo, em uma rede.
- Veracidade: Agente não comunica informações falsas.
- Benevolência: Agente ajudará os outros.
- Racionalidade: O agente não irá agir de forma a impedir a realização de seus objetivos.
- Cooperação: O agente coopera com o usuário.

2.3.3 Arquitetura de agentes

A arquitetura de agentes varia de acordo com a complexidade da sua autonomia, ou seja, com a capacidade de reagir aos estímulos do ambiente. Conforme verificado no livro de [7], os tipos de arquitetura são: orientadas à tabela, reflexiva simples, reflexiva baseado em modelo, baseada em objetivo, baseada em utilidade.

A primeira arquitetura a ser explorada é o agente orientado à tabelas. Todas as ações dos agentes dessa arquitetura são conhecidas e estão gravadas em uma tabela. Assim, quando o agente receber o estímulo ele já terá a ação a ser tomada previamente gravada em sua memória. Logo para construir esse tipo de agente, fica claro que além de saber todas percepções possíveis, será necessário definir ações apropriadas para todas. Isso levará a tabelas muito complexas e o tamanho pode facilmente passar a ordem de milhões dependendo do número de entradas.

A arquitetura reflexiva simples é um dos tipos mais simples de agente. Nele, o agente seleciona a ação com base unicamente na percepção atual, desconsiderando assim uma grande tabela de decisões. A decisão é tomada com base de regras condição-ação: Se uma condição ocorrer, uma ação será tomada. Por exemplo, vamos supor um agente médico que determina o diagnóstico de uma doença no paciente caso exista alguma anomalia no organismo (Por exemplo, paciente com febre). Uma condição-ação poderia ser:

if anomalia-organismo then diagnóstico-médico

Esse tipo de agente é bastante simples, o que é uma vantagem comparado à arquitetura de tabela. Porém, essa abordagem requer um ambiente totalmente observável, visto que esse tipo de agente possui uma inteligência bastante limitada. No exemplo do agente médico existem diversas maneiras de se detectar uma anomalia no organismo do paciente, seria necessário conhecer todas as formas para usarmos uma abordagem reativa simples.

A arquitetura reflexiva baseada em modelos funciona de maneira similar a anterior. Nessa abordagem, é levado em conta a parte do ambiente que não é visível neste momento. E para saber o "momento atual" de um agente, é necessário guardar a informação de estado consigo. Para atualizar o estado do agente, é necessário conhecer como o mundo desenvolve-se independente do agente (no caso do exemplo, como o organismo funciona) e é necessário saber as ações dos agentes no ambiente. Esses dois conhecimentos do ambiente

são chamados de **modelo do mundo**. O agente que usa esse tipo de abordagem é chamado de agente baseado em modelo.

Na arquitetura reflexiva baseada em objetivo, as ações do agente são tomadas apenas se o aproximam de alcançar um objetivo. Para isso, será necessário algo além do estado atual do ambiente: Será necessário informações do objetivo a ser atingido. Assim o agente pode combinar as informações do estado e o objetivo para determinar se deve ou não agir sobre o ambiente. Essa arquitetura porém é obviamente mais complexa e de certa forma ineficiente. Porém ela permite uma maior flexibilização das ações em determinados ambientes, visto que suas decisões são representadas de forma explícita e podem ser modificadas. É interessante notar que esse tipo de arquitetura não trata ações com objetivos conflitantes.

E por fim, a arquitetura reflexiva baseada em utilidade não utiliza apenas objetivos para realizar a próxima decisão, mas dá ao agente a capacidade de fazer comparações sobre o estado do ambiente e as ações a serem tomadas: Quais delas são mais baratas, confiáveis, baratas, rápidas do que as outras. A capacidade de avaliação do agente chama-se função de utilidade, que mapeia uma sequência de estados em um número real que determina o grau de utilidade. Esse mecanismo possibilita a decisão racional de escolha entre vários objetivos conflitantes. Por exemplo, escolher entre um objetivo mais barato ao invés de escolher entre o mais rápido.

2.3.4 Sistemas Multiagentes

Sistemas multiagentes são sistemas compostos por vários agentes capazes de se comunicar, possuindo uma linguagem de alto nível para isso. O agente deve ser conhecimento para realizar uma determinada tarefa e pode ou não cooperar com outros agentes para realizá-la.

Fica claro nessa definição que sistemas multiagentes

De acordo com [8], podemos encontrar as seguintes características principais de ambientes em SMAs:

- Ambientes SMAs fornecem protocolos específicos para comunicação e interação. Cada ambiente tem as suas particularidades: Alguns são em uma única máquina, outros são compartilhados com o mundo real e outros são distribuídos. Cabe a cada ambiente definir um protocolo onde todos agentes devem obedecer para comunicar-se.
- SMAs são tipicamente abertos.
- SMAs contém agentes que são autônomos e individualistas.

Capítulo 3

Metodologia de desenvolvimento

Para a realização deste trabalho, foi planejada uma metodologia de desenvolvimento na qual objetivou-se um estudo sobre um módulo da plataforma JAMA. Inicialmente, faz-se necessário um levantamento do estado da arte para levantar os principais trabalhos que são relacionados à área de dependabilidade em sistemas distribuídos, para a orientação desse trabalho.

Em seguida é necessário levantar os requisitos do JAMA, descrevendo e modelando detalhadamente o módulo escolhido para a análise. A modelagem será feita com base na linguagem *Unified Modeling Language* (UML). De acordo com [5], UML consegue expressar o significado semântico de um projeto de software à todos os *stakeholders* utilizando, na grande maioria dos casos, notação gráfica.

Após esse levantamento inicial, será necessário também levantar requisitos de dependabilidade. Como já visto anteriormente, ambientes distribuídos possuem diversas características de dependabilidade relativos à arquitetura. Serão analisados no trabalho características que talvez não sejam adequados para outras arquiteturas.

O próximo passo é a caracterização das falhas que um ambiente distribuído pode ter. Falhas como erro na transmissão de mensagens para outros pares, perda de mensagens na rede, segurança na autenticação dos pares, ataques de negação de serviços da rede, propagação de vírus, dentre outros, devem ser requisitos considerados na análise a ser feita neste trabalho.

Em seguida se dá o desenvolvimento de um modelo probabilístico de estados com base em cadeias de *Markov*, para determinar as operacões do sistema que tem algum tipo de relação com os requisitos de dependabilidade acima mencionados.

O próximo passo é a implementação do modelo através do sistema de checagem de modelo *PRISM*. Em seguida a modelagem para análise formal utilizando pctl (*probabilistic computational tree logic*). Com todos essas análises e modelagens, será possível enfim realizar uma análise de sensibilidade mais detalhada sobre o componente do JAMA em questão.

Após a análise, caso seja necessário, serão propostas melhorias visando diminuir o grau de dependência de alguns componentes com o módulo avaliado. A hipótese a ser considerada é que o JAMA possuirá um baixo nível de sensibilidade de componentes do sistema. Essa melhoria pode ser no nível de estudos ou mesmo uma implementação em nível de código.

Por fim, essa solução deverá ser testada e avaliada afim de garantir a sua melhoria em relação à versão anterior da plataforma antes de ser realiazado esse estudo.

Referências

[1] Object management group. http://www.omg.org/. Accessed: 05/01/2013. 4

[2] A. Cockburn. *Writing effective use cases*. Agile software development series. Addison-Wesley, 2001. 5, 7

[3] Martin Fowler. *UML Distilled: A Brief Guide to the Standard Object Modeling Language*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 3 edition, 2003. vi, 4, 6

[4] C. LARMAN. *Utilizando UML e Padrões*. Bookman, 3 edition, 2008. vi, 5, 7, 8

[5] C. LARMAN. *Utilizando UML e Padrões*. BOOKMAN COMPANHIA ED, 2008. 15

[6] David Poole, Alan Mackworth, and Randy Goebel. *Computational Intelligence: A Logical Approach*. Oxford University Press, 1998. 10

[7] Stuart Russel and Peter Novig. *Artificial Intelligence - A Modern Approach*. Number 1. Pearson Education, Upper Saddle River, New Jersey 07458, 1995. 11, 13

[8] Carlos Vinícius Sarmento Silva. Agentes de mineração e sua aplicação no domínio de auditoria governamental. Acessado em 18/11/2011. Disponível em http://monografias.cic.unb.br/dspace/handle/123456789/318, Abril 2011. 14

[9] Michael Wooldridge. *An Introduction to MultiAgent Systems*. Number 1. John Wiley and Sons Ltd, Krst Sussex POI0 IlJD, England, 2004. 11, 12