UNIVERSIDADE DE CAXIAS DO SUL
CENTRO DE COMPUTAÇÃO E TECNOLOGIA DA INFORMAÇÃO
BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

LUCAS RAFAEL COSTELLA PESSUTTO

Proposta de Arquitetura Multiagente para o Portal de Algoritmos da UCS (AlgoUCS)

Marcos Eduardo Casa
Orientador

Caxias do Sul, Dezembro de 2013

Proposta de Arquitetura Multiagente para o Portal de Algoritmos da UCS (AlgoUCS)

por

Lucas Rafael Costella Pessutto

Projeto de Diplomação submetido ao curso de Bacharelado em Ciência da Computação do Centro de Computação e Tecnologia da Informação da Universidade de Caxias do Sul, como requisito obrigatório para graduação.

Projeto de Diplomação

Orientador: Marcos Eduardo Casa
Banca examinadora:

Ricardo Vargas Dorneles
CCTI/UCS
Elisa Boff
CCTI/UCS

Projeto de Diplomação apresentado em
09 de dezembro de 2013

Daniel Luís Notari
Coordenador

"Se eu cheguei longe, foi por ter subido nos ombros de gigantes."
Isaac Newton

AGRADECIMENTOS

Gostaria de agradecer aos meus pais Sérgio e Tania, ao meu irmão Leonardo e a toda a minha família e todos os meus amigos. Sem vocês este trabalho não seria possível, obrigado pela dedicação, confiança e incentivo em toda a minha jornada.

Agradeço ao professor Marcos Casa que dedicou seu valioso tempo para me guiar em cada passo deste trabalho. Sua orientação, seu grande conhecimento e incentivo serviram de inspiração para a viabilização desta monografia.

Dedico este trabalho a todos os professores que participaram da minha educação, desde os que me ensinaram a ler aos que me ensinaram a programar, graças a profissionais tão brilhantes este trabalho pode se tornar realidade.

Agradeço aos meus colegas de curso, por deixarem mais fácil e mais leve a caminhada na universidade e pelas trocas de conhecimento e experiência que tivemos.

E finalmente agradeço a Deus, por proporcionar estes agradecimentos a todos que tornaram minha vida melhor, além de ter me dado uma família maravilhosa e amigos sinceros.

Lucas Rafael Costella Pessutto

SUMÁRIO

LISTA DE ACRÔNIMOS

ACL	*Agent Communication Language*
ADL	*Action Description Language*
AID	*AgentIdentifier*
AMS	*Agent Management System*
API	*Application Programming Interface*
BDI	*Believe-Desire-Intention*
BPML	*Business Process Modeling Language*
BPMN	*Business Process Model and Notation*
CORBA	*Common Request Broker Architecture*
CT	*Container Table*
DF	*Directory Facilitator*
FIPA	*Foundation for Intelligent Physical Agents*
GADT	*Global Agent Descritor Table*
HTML	*HyperText Markup Language*
IDE	*Integrated Development Environment*
JADE	*Java Agent Development Kit*
Jadl++	*JIAC agent description language*
JIAC	*Java-based Intelligent Agent Componentware*
JMS	*Java Message Service*

JSP *Java Server Pages*

JVM *Java Virtual Machine*

LADT *Local Agent Descritor Table*

SOA *Service Oriented Architecture*

UCS *Universidade de Caxias do Sul*

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE TRECHOS DE CÓDIGO

RESUMO

O ensino de algoritmos tornou-se um grande desafio ao longo do tempo, pois esta disciplina requer do aluno raciocínio lógico e abstração de alguns conceitos necessários ao aprendizado de uma linguagem de programação. Esses fatores somados a uma educação básica precária recebida por alguns alunos contribuíram para o alto índice de reprovação da disciplina de algoritmos.

Como forma de diminuir os índices de reprovação e fornecer aos alunos uma ferramenta de estudo, um grupo de professores da Universidade de Caxias do Sul desenvolveu o Portal de Algoritmos da *UCS* (*Universidade de Caxias do Sul*), que é basicamente uma *IDE* para a linguagem Português Estruturado, ensinado nas aulas de algoritmos, aliado a um banco de problemas que permite aos alunos submeter suas soluções e avaliá-las, determinando se estão corretas e obtendo o custo de sua solução. Com esse número o sistema elabora um ranking dos usuários que resolveram mais problemas e das melhores soluções apresentadas para cada problema.

Este trabalho irá propor uma arquitetura multiagente que suporte as funcionalidades do portal de algoritmos presentes atualmente na plataforma. Essa abordagem terá como objetivo permitir uma futura integração de novos agentes à sociedade, tendo em vista as diversas possibilidades de novas funcionalidades que a ferramenta oferece, principalmente em termos de Inteligência Artificial.

Num primeiro momento será feito um levantamento bibliográfico da área de Agentes e Sistemas multiagentes, um estudo de alguns frameworks para o desenvolvimento de Sistemas Multiagentes e um estudo do portal de algoritmos. Após esse estudo inicial será proposta a arquitetura multiagente para a plataforma e posteriormente será realizado um estudo de caso e implementação do sistema. Por fim, serão analisados os resultados obtidos a fim de determinar a corretude da arquitetura proposta inicialmente.

Palavras-chave: Inteligência Artificial, Algoritmos, Sistemas Multiagentes, Engenharia de Software Multiagente, Portal de Algoritmos da UCS.

Proposal for a Multiagent Architecture for the Portal de Algoritmos da UCS (AlgoUCS)

ABSTRACT

Teaching algorithms has become a great challenge over the time, because this course requires the student's logical reasoning and abstraction of some concepts needed to learning a programming language. These factors added up to a precarious basic education received by some students contributed to the fact of the high failure rate of the discipline of algorithms.

As a form of decrease the failture rates and provide to the students a tool to study, a group of professors from the Universidade de Caxias do Sul developed the Portal de Algoritmos da *UCS*, which is basically an *IDE* for Structured Portuguese language, taught in algorithms' classes, combined with a database of problems that allows students to submit their solutions and evaluates them, determining if they are correct and getting the cost of their solution. With this number the system produces a ranking of the users who solved more problems and the best solutions for each problem.

This paper will propose a multiagent architecture that supports the functionalities of the Portal de Algoritmos currently present on the platform. This approach will aim to allow future integration of new agents to the society, in view of the many possibilities of new features that the tool provides, especially in terms of Artificial Intelligence.

At first time, will be made a bibliographical survey of agents and multiagent systems area, a study of some frameworks to development of multiagent systems and a study of the Portal de Algoritmos. After this initial study will be proposed multiagent architecture for the platform and later will be done a case study and implementation of the system. Finally, will be analyzed the results achieved to determine the correctness of the proposed architecture initially.

Keywords: Artificial Intelligence, Algorithms, Multiagent Systems, Multiagent Software Engeneering, Portal de Algoritmos da UCS.

1 INTRODUÇÃO

A disciplina de algoritmos é uma disciplina básica dos cursos da área de informática e tecnologia, e tem como objetivo principal ensinar o acadêmico a utilizar o raciocínio lógico-matemático para buscar a resolução de um problema e posteriormente formalizar essa resolução em uma linguagem de programação. (JúNIOR; RAPKIEWICZ, 2005). Essa disciplina introdutória apresenta elevados índices de reprovação, devido à dificuldade dos estudantes em assimilar as abstrações envolvidas no processo de aprendizagem de algoritmos e pela deficiência em seu raciocínio lógico-matemático.(JúNIOR et al., 2006).

Buscando uma melhora no desempenho dos alunos das disciplinas de algoritmos, surgiu o Portal de Algoritmos da *UCS* (*Universidade de Caxias do Sul*) (AlgoUCS), que é um objeto de aprendizagem, desenvolvido pelos professores Ricardo Vargas Dorneles e Delcino Picinin Júnior da Universidade de Caxias do Sul, que possui um editor de código fonte e um compilador para a linguagem algorítmica utilizada nas aulas e um banco de problemas dos diversos conteúdos trabalhados na disciplina, podendo o aluno submeter sua solução e compará-la com a de outros colegas.

Este trabalho fará uma análise do portal de algoritmos e posteriormente será proposto um modelo de arquitetura multiagente que deverá suportar as funcionalidades que já existem nesse ambiente, tendo como objetivo mapear os possíveis agentes existentes no ambiente e suas respectivas funções. Além disso, pretende-se com este tipo de arquitetura facilitar o processo de inserção de novos agentes a esta sociedade, definindo um protocolo comum de comunicação entre os agentes.

Primeiramente, será realizado um levantamento bibliográfico acerca do assunto a ser estudado (Inteligência Artificial, Agentes Inteligentes, Sistemas Multiagentes), visando buscar um aprofundamento maior nessa área. Também será feito um levantamento de alguns frameworks utilizados para implementar sistemas multiagentes, avaliando sua aplicabilidade na resolução do problema proposto. Paralelamente a esse levantamento bibliográfico, será estudado o Portal de Algoritmos da *UCS*, buscando um entendimento acerca de seu funcionamento, suas funcionalidades e os serviços disponibilizados e a estruturação e armazenamento das informações pelo

AlgoUCS.

Como produto deste estudo será produzida uma descrição da arquitetura multiagente para o Portal de Algoritmos, contemplando uma descrição dos agentes, do comportamento de cada agente no ambiente e dos serviços que cada agente fornecerá, além de modelar as interações entre os agentes e a tecnologia empregada no desenvolvimento desses agentes. Em seguida, será feito um estudo de caso sobre essa arquitetura, implementando-se parte do que foi proposto na arquitetura, com o objetivo de avaliar a corretude da solução proposta e sua aplicabilidade na solução do problema de pesquisa desse trabalho.

1.1 Objetivos

1.1.1 Objetivo geral

Modelagem e desenvolvimento de uma arquitetura multiagente que suporte as funcionalidades do Portal de Algoritmos da UCS.

1.1.2 Objetivos específicos

- Pesquisar e avaliar frameworks voltados à programação de sistemas multiagentes e sua aplicabilidade ao problema proposto.
- Modelar a arquitetura multiagente, de acordo com os estudos realizados no AlgoUCS e através do levantamento bibliográfico.
- Implementar um estudo de caso para avaliar se a arquitetura proposta pode ser aplicada ao Portal de Algoritmos.

1.2 Estrutura do trabalho

No Capítulo 2 são apresentados conceitos básicos sobre Inteligência Artificial, Agentes Inteligentes, Sistemas Multiagentes, necessários para definir os conceitos que nortearão este trabalho.

No Capítulo 3 serão descritos dois frameworks para o desenvolvimento de Sistemas Multiagentes e posteriormente será feita uma comparação entre tais plataformas.

No Capítulo 4 será realizada uma descrição das funcionalidades presentes no Portal de Algoritmos da *UCS*.

No Capítulo 5 é descrita formalmente a arquitetura multiagente proposta para o Portal de Algoritmos e que posteriormente será implementada.

No Capítulo 6 é apresentada a implementação da arquitetura proposta e a avaliação de seu desempenho.

2 REFERENCIAL TEÓRICO

Inteligência Artificial é o ramo da Ciência da Computação que se preocupa em automatizar o comportamento inteligente, através da representação de conhecimento e técnicas que aplicam este conhecimento na resolução de problemas. (LUGER, 2004)

Para se obter uma definição de conhecimento, há que se entender o que são dados e informações, e como estes se relacionam com o conceito de conhecimento, segundo (REZENDE, 2003) :

- Dados são conjuntos de símbolos que podem ser quantificados e que representam algum evento, mas sozinhos não nos oferecem nenhum entendimento do que eles representam. Por exemplo, dizer que o faturamento de determinada empresa foi de 1 milhão de reais no mês de janeiro. Note que este número somente ilustra o valor do faturamento de uma empresa em determinado mês, não nos permitindo aferir nada sobre esse fato.
- Uma Informação é a contextualização e a análise de um conjunto de dados, permitindo que seja possível a interpretação destes (dentro de determinado contexto). Como exemplo, pode-se avaliar o conjunto formado pelos dados correspondentes ao faturamento de uma empresa no mês de janeiro, durante o período de dez anos. Percebe-se que esses dados permitem uma interpretação dos valores, por exemplo, pela comparação dos mesmos.
- Já o conhecimento se dá pela comparação e combinação de informações (que se encontram na forma de fatos, heurísticas e regras) que possuem alguma utilidade e um significado, e que permitem que sejam aferidas novas informações a partir dessas comparações. Por exemplo, se considerarmos o exemplo anterior referente ao faturamento de uma empresa, podemos, entender o porquê de um faturamento baixo em determinado ano se compararmos este a dados referentes à economia global, por exemplo.

São adotadas, basicamente, duas abordagens distintas quando se estuda Inteligência Artificial: as que pretendem imitar o pensamento humano e as que giram

em torno do conceito de racionalidade. (RUSSELL; NORVIG, 2004)

As abordagens que procuram imitar o pensamento humano atuam de forma empírica para resolver um determinado problema: primeiramente se observa como a mente humana desenvolve determinado raciocínio, em seguida é escrito um algoritmo com base nas observações feitas, que será codificado em um programa de computador. Após isso, são realizados testes com o intuito de verificar se, em mesmas condições, a saída do programa corresponde ao comportamento humano correspondente.

Por outro lado, as técnicas centradas no conceito de racionalidade se utilizam da lógica para encontrar a solução de um determinado problema, e ao contrário da abordagem anterior esta não admite erros, ou seja, com base nas informações que o sistema inteligente possui, este sempre encontrará a melhor resposta possível para o seu problema.

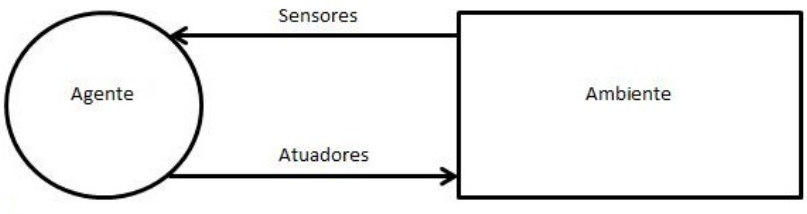
2.1 Agentes Inteligentes

Com base na abordagem racionalista, desenvolveu-se o conceito de agente racional: que é um programa de computador que age em determinado ambiente, percebendo-o e modificando-o, através de ações autônomas que este toma, visando sempre o melhor resultado possível para resolver seus problemas. (RUSSELL; NORVIG, 2004)

O foco desse estudo está em o que um agente faz, ou seja, avalia-se o modo de agir do agente para julgar se o agente é inteligente ou não. Um agente é considerado inteligente quando: (POOLE; MACKWORTH, 2010)

- Suas ações forem apropriadas às circunstâncias e estiverem de acordo com seus objetivos;
- Ele é flexível para alterar tanto o ambiente quanto os seus objetivos;
- Ele aprende de acordo com suas experiências, e
- É capaz de tomar as decisões corretas, dada sua percepção do ambiente.

Figura 2.1: Interação Agente / Ambiente



Fonte: (RUSSELL; NORVIG, 2004)

Como ilustra a Figura 2.1, um agente percebe o ambiente no qual está inserido através de sensores, que irão servir de entrada para o processamento que ele irá executar, e sua saída será uma ação no ambiente que se dará através de atuadores que modificarão o ambiente, de maneira a atingir o objetivo deste agente.

A percepção de um agente pode ser dada com base no conjunto de percepções do ambiente, ou somente com base na sua percepção atual. Damos o nome de Função de Agente ao conjunto de percepções necessárias que um agente deve fazer para que seja desencadeada determinada ação. (RUSSELL; NORVIG, 2004)

Para que um agente possa ser considerado racional, ele deve sempre escolher a melhor solução possível, dentre suas percepções do ambiente e de seu conhecimento interno. Mas, como garantir que a solução escolhida é a melhor possível? É necessário, ao projetar um agente racional, definir uma medida objetiva de desempenho, que avalie a qualidade das ações do agente. (RUSSELL; NORVIG, 2004)

Um agente racional pode ser definido em função de quatro itens básicos: ambiente, sensores, atuadores e a medida de desempenho desse agente. Logo, para cada sequência de percepções do agente e de seu conhecimento interno, o agente deve escolher a ação que venha a maximizar sua medida de desempenho, e externando essa ação através de seus atuadores. Para que um agente execute corretamente sua função, deve ser projetado de modo a contemplar da maneira mais abrangente possível esses quatro itens. (RUSSELL; NORVIG, 2004)

Além disso, para que o agente realize efetivamente seu trabalho é importante que sejam implementados métodos de realizar a coleta das informações do ambiente, de maneira que o agente se cerque de toda informação possível para tomar a melhor decisão. Uma vez de posse de tais informações, o agente deve extrair o máximo de conhecimento que ele puder, ajudando na tomada de decisão. É interessante também que o agente aprenda ao longo do tempo, garantindo com isso uma certa autonomia ao agente. (RUSSELL; NORVIG, 2004)

2.1.1 Estrutura Interna de um Agente Inteligente

Após a análise do que são agentes inteligentes, passa-se agora a uma definição abstrata de um agente, para então tratarmos como é estruturado internamente um agente, e quais os principais tipos de implementação de agentes.

Definiremos, primeiramente, que um agente (Ac) é representado formalmente por um conjunto finito de ações possíveis que podem alterar o estado do ambiente (WOOLDRIDGE, 2002):

$$Ac = \{\alpha, \alpha', \ldots\} \tag{2.1}$$

E um ambiente (E) é definido por (WOOLDRIDGE, 2002) como sendo um

conjunto de estados finitos que podem ser alcançados através das ações de um agente. Note que ambientes contínuos podem ser mapeados através de conjuntos discretos com um certo nível de precisão.

$$E = \{e, e' \ldots\} \tag{2.2}$$

Tendo a descrição de um agente dada em 2.1 e um ambiente como o exposto em 2.2, (WOOLDRIDGE, 2002) descreve agora como ocorre a interação entre tais entidades: Inicialmente o ambiente encontra-se em determinado estado, então o agente escolhe uma ação para ser executada. Como resultado dessa ação, o ambiente responde com um ou mais possíveis estados, mas como ao final da execução somente poderá existir um estado final, o agente escolhe outra ação para ser executada, e assim por diante, até que reste somente um único estado. Como resultado desse processo temos que uma execução (r) é dada por:

$$r = e_0 \overset{\alpha_0}{\rightarrow} e_1 \overset{\alpha_1}{\rightarrow} e_2 \overset{\alpha_2}{\rightarrow} \ldots \overset{\alpha_{n-1}}{\rightarrow} e_u \tag{2.3}$$

Chamamos de R ao conjunto formado por todas as sequências possíveis de execução de um determinado agente Ag sobre um ambiente E. E deste conjunto derivam-se dois subconjuntos: R^{Ac} que contém todas as execuções que se encerram em uma ação, e R^e que possui as execuções terminadas em estados do ambiente. (WOOLDRIDGE, 2002)

Para se representar o efeito das ações de um agente em determinado ambiente utiliza-se o conceito de Função Transformadora de Estado (τ) que é uma função que mapeia uma determinada execução (pertencente ao conjunto R^{Ac}) em um conjunto de possíveis estados do ambiente, que podem ser resultado de alguma ação do agente (WOOLDRIDGE, 2002). Formalmente temos:

$$\tau = R^{Ac} \rightarrow \wp(E) \tag{2.4}$$

Nota-se, pela equação 2.4, que o ambiente é dependente da história, ou seja, para se determinar o estado de um ambiente leva-se em conta, além da ação realizada pelo agente e o estado atual do ambiente, todas as ações realizadas anteriormente naquele ambiente.

Um ambiente pode ser representado formalmente como uma tripla: $Env = \{E, e_0, \tau\}$, onde E é o conjunto dos estados do ambiente, $e_0 \in E$ e representa o estado inicial do ambiente e τ é a Função de Transformação do Ambiente definida em 2.4 (WOOLDRIDGE, 2002).

Finalmente, a representação formal de um agente é definida como sendo uma função que mapeia uma execução (definida em 2.3) em uma ação (definida em 2.1) (WOOLDRIDGE, 2002):

$$Ag : R^E \rightarrow Ac \tag{2.5}$$

2.1.2 Tipos Básicos de Agentes

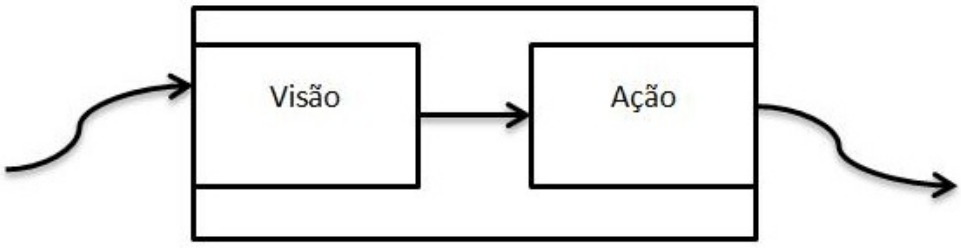
Será feita agora uma análise interna do funcionamento de um agente, dividindo-se o conjunto de agentes em cinco tipos básicos que possuem as características de praticamente todos os sistemas inteligentes, proposta por (RUSSELL; NORVIG, 2004):

2.1.2.1 Agentes Puramente Reativos

Os Agentes Puramente Reativos são agentes que se baseiam unicamente na percepção atual que estão tendo do ambiente para realizar sua tomada de decisão. Normalmente esse tipo de agente é construído especificando-se um conjunto de regra condição-ação, onde para cada percepção do agente se desencadeia uma (ou mais) ações correspondentes no agente.

Internamente, um agente desse tipo pode ser subdividido em dois subsistemas menores: visão e ação (WOOLDRIDGE, 2002), conforme ilustra a Figura 2.2:

Figura 2.2: Arquitetura Interna de um Agente Puramente Reativo



Fonte: (WOOLDRIDGE, 2002)

A Função Visão é responsável por modelar a capacidade do agente de observar o ambiente e refletir isso em uma percepção. Formalmente, ela é uma função (WOOLDRIDGE, 2002):

$$Visao : E \rightarrow Per \tag{2.6}$$

Uma propriedade da função 2.6 que merece destaque é a seguinte: dados dois estados do ambiente e_1 e e_2, sendo que $\{e_1, e_2\} \in E$ e tendo $e_1 \neq e_2$, mas $visao(e_1) = visao(e_2)$. Com isso, conclui-se que dois estados diferentes de um ambiente podem mapear a mesma percepção, sendo indistinguíveis para o agente. Isso ocorre quando determinado estado mapeia duas ou mais características do ambiente e alguma(s) delas não tem relevância nenhuma para o agente, quando isso ocorre diz-se que os estados são considerados equivalentes: $e_1 \sim e_2$. (WOOLDRIDGE, 2002)

Se dividirmos os estados de determinado ambiente E em classes de estados naturalmente indistinguíveis, temos que: (WOOLDRIDGE, 2002)

- Se $|\sim| = |E|$, ou seja, se o número de percepções do agente é igual ao número de possíveis estados do ambiente, o agente tem uma percepção perfeita do ambiente distinguindo todos os estados possíveis.
- Se $|\sim| = 1$, o agente não possui a capacidade de percepção, sendo que ele considera todos os estados como sendo idênticos.

(RUSSELL; NORVIG, 2004) ainda definiram um algoritmo básico para um agente reativo simples que sintetiza a função desse tipo de agente como segue abaixo:

Trecho de Código 2.1: Algoritmo Agente Puramente Reativo

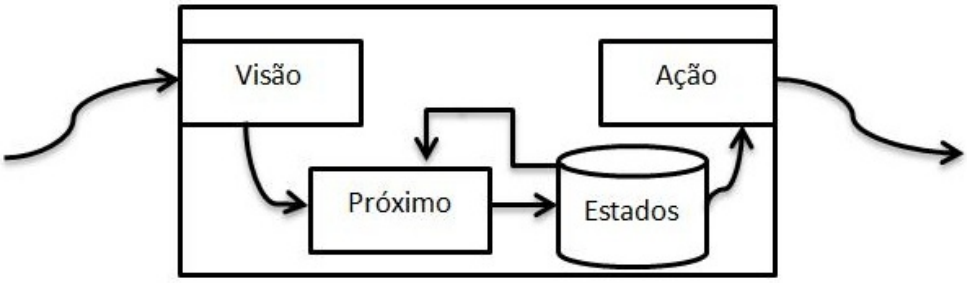
```
1 Função AGENTE-REATIVO-SIMPLES (percepção) retorna ação
2     variáveis estáticas: regras, um conjunto de regras condição-ação
3
4     estado <- INTERPRETAR-ENTRADA (percepção)
5     regra <- REGRA-CORRESPONDENTE (estado, regras)
6     ação <- AÇÃO-DA-REGRA [regra]
7     retornar ação
8 fimfunção
```

2.1.2.2 Agentes Reativos Baseados em Modelo

Nessa abordagem de construção de agentes, é armazenado em uma estrutura de dados interna o histórico das percepções do agente e algum conhecimento sobre o funcionamento do ambiente no qual o agente está inserido, de modo que o agente possa prever o comportamento de determinadas estruturas. Para tomar alguma decisão o agente leva em conta todo o histórico de percepções ou parte dele.

Podemos expandir o modelo definido para representar o Agente Puramente Reativo para definir o comportamento interno dos Agentes Baseados em Modelo:

Figura 2.3: Estrutura Interna de um Agente Baseado em Modelo



Fonte: (WOOLDRIDGE, 2002)

No modelo apresentado na Figura 2.3, a função Visão mapeia um estado do ambiente a partir de uma percepção. Essa percepção atual é combinada com o

histórico anterior armazenado no agente e produz um estado, através da função Próximo, e por fim, a função Ação recebe como parâmetro um estado interno do agente e o transforma em uma ação.

Um algoritmo que descreve esse tipo de agente pode ser assim definido:(RUSSELL; NORVIG, 2004)

Trecho de Código 2.2: Algoritmo Agente Reativo Baseado em Modelo

```
1 Função AGENTE-REATIVO-SIMPLES (percepção) retorna ação
2     variáveis estáticas: estado, uma descrição do  estado atual do mundo,regras,
           um conjunto de regras condição-ação, ação mais recente inicialmente
          nenhuma
3
4     estado <- ATUALIZAR-ESTADO (estado, ação, percepção)
5     regra <- REGRA-CORRESPONDENTE (estado, regras)
6     ação <- AÇÃO-DA-REGRA [regra]
7     retornar ação
8 fimfuncao
```

2.1.2.3 Agentes Baseados em Objetivo

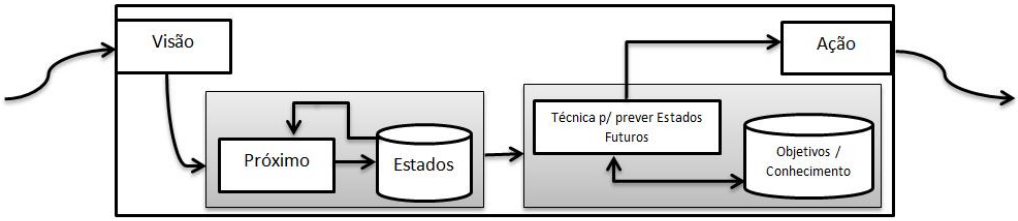
Para um agente baseado em objetivo mais do que somente reagir a determinado evento, é importante que se avalie se aquela ação conduz o agente para uma situação desejável. Assim, o agente combina o conjunto de ações possíveis (retornados pelo ambiente a cada ação executada pelo agente) com seus objetivos mapeados sob a forma de conhecimento explícito dentro da estrutura do agente.

Esse tipo de agente se difere de um agente reativo simples devido à tomada de decisões considerar um cenário de longo prazo: um agente baseado em objetivo pode se utilizar de mecanismos como por exemplo técnicas de busca e planejamento que são mais avançadas do que regras condição-ação para prever uma sequência maior de ações para encontrar uma maneira de atingir seus objetivos.

Outra vantagem dessa abordagem sobre os agentes reativos simples é que como o conhecimento interno do agente está mapeado explicitamente o próprio agente, em tempo de execução, pode alterar esse conhecimento de acordo com a realidade em que se encontra, enquanto que as regras condição-ação não admitem essa característica. Para alterar o comportamento do agente, deve-se reescrever ou adicionar novas regras ao programa de agente.

A Figura 2.4 mostra de maneira esquematizada a estrutura interna de um agente baseado em objetivo, nota-se uma estrutura mais complexa se comparada às anteriores e percebe-se que num primeiro momento ocorre a percepção do ambiente e o mapeamento do estado interno do agente, após isso, é aplicada alguma técnica que através dos estados anteriores e dos objetivos e conhecimentos presentes neste agente irá escolher a ação que aproxima mais o agente de seus objetivos.

Figura 2.4: Estrutura Interna de um Agente Baseado em Objetivo



2.1.2.4 Agentes Baseados em Utilidade

Os agentes baseados em utilidade incorporam aos agentes baseados em objetivos uma função de utilidade que é usada para medir a qualidade de determinado estado (ou solução). A tarefa de um agente baseado em utilidade, então é mover-se aos estados que irão maximizar sua utilidade.

Uma função de utilidade basicamente mapeia estados do ambiente em um valor numérico real. Tal função é definida de acordo com a tarefa a ser executada pelo agente. Por exemplo, se fizermos uma análise pessimista podemos calcular a utilidade global do agente como sendo a pior utilidade dos estados que ele pode alcançar, ou também podemos definir a utilidade de um agente como a média dos estados que podem ser alcançados, entre outras técnicas que podem ser utilizadas.

Uma desvantagem das funções de utilidade está em atribuir uma utilidade para um estado local, pois é difícil especificar uma visão de longo prazo atribuindo utilidades para estados individualmente. Uma maneira melhor de definir uma utilidade é atribuir uma utilidade para uma execução inteira.

Por outro lado, essa abordagem se mostra vantajosa quando o agente possui diversos objetivos que são contraditórios, pois uma função de utilidade estabelece as prioridades do agente em relação aos seus objetivos. E quando não há certeza de sucesso, pode-se ponderar através da função de utilidade essa probabilidade com a importância desses objetivos.

2.1.2.5 Agentes Baseados em Aprendizagem

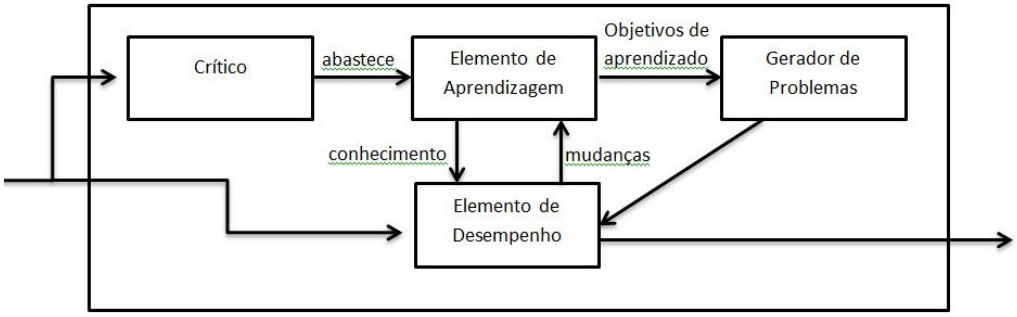
(RUSSELL; NORVIG, 2004) destacam os quatro componentes integrantes de agente baseado em aprendizado:

- Elemento de Aprendizado: A função desse componente é analisar o desempenho do agente na execução das suas tarefas e modificar o elemento de desempenho de modo a melhorar a performance do agente.
- Elemento de Desempenho: Esse item se resume ao que era considerado o agente nos itens anteriores: ele analisa as percepções recebidas e decide as ações do agente no futuro.

- Crítico: Elemento responsável por avaliar o funcionamento do agente através de um padrão fixo de desempenho. É com ele que o elemento de aprendizado avalia o sucesso de determinada ação no ambiente.
- Gerador de Problema: Este componente se ocupa em fazer o agente explorar novas e diferentes experiências com o objetivo de fazer com que ele não se restrinja às soluções que são óbvias e explore soluções que parecem piores no momento, mas que podem se mostrar ótimas em longo prazo.

O funcionamento interno de um agente com aprendizagem e a integração dos quatro módulos internos desse tipo de agente está ilustrado na Figura 2.5.

Figura 2.5: Estrutura Interna de um Agente Baseado em Aprendizagem.



Fonte: (RUSSELL; NORVIG, 2004)

O elemento de aprendizado pode ser implementado, de uma maneira simplificada, com aprendizado direto: através da observação de pares de estados o agente aprende como suas ações afetam o ambiente. Quanto mais informações ele retirar do ambiente, mais fácil será sua aprendizagem. E através das medidas de desempenho executadas pelo crítico, ele recompensará (ou penalizará) as percepções obtidas e isso irá retroalimentar a qualidade do comportamento do agente.

2.2 Ambientes

Nesta seção serão apresentadas as principais classificações que podem ser atribuídas a um ambiente segundo (RUSSELL; NORVIG, 2004).

2.2.1 Completamente Observável x Parcialmente Observável

Um ambiente é dito completamente observável se um agente pode obter informações completas, precisas e atualizadas sobre o estado do ambiente em determinado momento. Sendo assim, um agente que atua nesse tipo de ambiente não necessita armazenar internamente nenhuma informação do ambiente. Caso o ambiente não permita essas condições ele é dito parcialmente observável. Quanto mais observável for um ambiente mais fácil será construir agentes que atuem nele, pois

como há mais oferta de informações, torna-se mais fácil para o agente tomar decisões mais acertadas nesse tipo de ambiente.

2.2.2 Determinístico x Estocástico

Num ambiente determinístico o agente pode prever o resultado de suas ações no ambiente, se houver incerteza no ambiente dizemos que ele é estocástico. Ainda, se o ambiente é determinístico para um agente exceto pelas ações de outros agentes, dizemos que o ambiente é estratégico.

Um ambiente estocástico captura o fato de que os agentes tem uma esfera de influência limitada sobre o ambiente, ou seja, ele somente controla parte do ambiente. Além disso, esse tipo de ambiente considera o fato de que determinadas ações podem falhar ao se tentar alcançar determinado estado desejado do ambiente.

2.2.3 Episódico x Sequencial

Em ambientes episódicos cada ação do agente é dividida em episódios atômicos que consistem de uma percepção do agente e a execução de uma única ação, sendo que cada episódio é independente dos demais. Tarefas de classificação em geral são episódicas. Já em ambientes sequenciais, cada decisão do agente tem o poder de afetar as futuras decisões por ele tomadas. Nota-se que é mais fácil a implementação de agentes para atuarem em ambientes episódicos, tendo em vista que o agente não necessita pensar à frente, preocupando-se somente com cada momento em particular.

2.2.4 Estático x Dinâmico

Um ambiente é considerado estático se ele somente altera seu estado com as ações do agente e é chamado de dinâmico se outros processos alteram a configuração do ambiente sem que o agente controle isso. Nesse tipo de ambiente o agente precisa se preocupar com a passagem do tempo. Quando o ambiente permanece inalterado, mas o agente melhora seu desempenho com o passar do tempo ele é chamado de semidinâmico.

2.2.5 Discreto x Contínuo

Em ambientes discretos, há um número finito de estados, ações e percepções possíveis, caso contrário o ambiente é considerado contínuo. Ambientes contínuos são mais complexos de implementar devido ao fato do computador ser uma máquina discreta, logo ambientes contínuos devem ser simulados em computadores com algum grau de precisão e isso acarreta em perda de informação.

2.2.6 Agente único x Multiagente

Um ambiente é considerado multiagente se mais de um agente estiver atuando nesse ambiente em um mesmo intervalo de tempo, caso contrário esse ambiente é de agente único.

O agente deve observar quais entidades estão tentando maximizar alguma medida de desempenho para poder classifica-la como um agente. Quando maximizar essa medida de desempenho por consequência minimiza a medida de desempenho de outros agentes, esse ambiente é dito multiagente competitivo, e quando todos os agentes buscam maximizar a mesma medida de desempenho ele é dito multiagente cooperativo. Ambientes multiagente são mais complexos, pois devem implementar meios de interação entre os agentes daquele ambiente.

2.3 Sistemas Multiagentes

Conceitua-se um Sistema Multiagente como sendo o conjunto de um ambiente de dois ou mais agentes que atuam nesse ambiente, sendo que tais agentes interagem entre si para solucionarem seus problemas, produzindo um comportamento global inteligente (WOOLDRIDGE, 2002). Esse tipo de abordagem se popularizou recentemente devido ao crescente uso de sistemas distribuídos e do avanço nas tecnologias de rede e das tentativas de padronização como *CORBA* (*Common Request Broker Architecture*). (BITTENCOURT, 2006)

Cada agente que constitui o sistema possui uma limitada esfera de influência sobre o sistema, podendo esta coincidir com a esfera de influência de outros agentes (e divergir da de outros agentes), criando assim relações de dependência entre eles. (WOOLDRIDGE, 2002)

O comportamento de um agente dentro de um ambiente multiagente, acaba por influenciar os outros agentes desse sistema naturalmente, mesmo que ele não tenha consciência da existência destas entidades. A tal fato chamamos de Interferência Social e esta pode ser classificada como: (REZENDE, 2003)

- Positiva: quando a ação de um agente aproxima outro de seu objetivo.
- Negativa: quando a ação do agente afasta outro de seu objetivo.
- Neutra: quando não há interferência.

Uma das consequências da interferência social é a mudança dos objetivos atingíveis de um agente, pois como está em sociedade, outro agente poderá influenciar positivamente no comportamento deste agente, fazendo com que ele atinja novos estados que sozinho ele não conseguiria atingir, criando uma relação de dependência entre estes agentes. Nesse estágio um agente pode-se utilizar de planejamento social para influenciar outro agente para que ele execute uma determinada ação que venha a

beneficiar o agente que o influenciou. Quando um agente influencia a outro ele pode oferecer ajuda para que este agente atinja o mesmo objetivo (cooperação) ou para que ele atinja outro objetivo (escambo social). (REZENDE, 2003)

Mesmo buscando um mesmo objetivo podem ocorrer certas divergências entre os agentes envolvidos, sendo necessário que os agentes negociem e entrem em acordo em relação ao que será executado. Através de negociação é possível que os agentes tomem uma decisão em conjunto beneficiando ambos os agentes. Para que isso ocorra é necessário estabelecer protocolos de comunicação entre os agentes para agilizar o processo de comunicação entre eles. (REZENDE, 2003)

O primeiro protocolo desse tipo foi chamado de Rede Contratual (SMITH, 1980): Nesse protocolo quando um agente necessita que determinado serviço seja realizado ele faz um anúncio e os agentes que podem executar esse serviço fazem uma oferta, com base em alguma métrica o agente escolhe um dos ofertantes e fecha uma parceria com este para a execução da tarefa.

Nas situações em que dois agentes trabalham juntos buscando atingir objetivos comuns usando de cooperação, é necessário que haja coordenação das tarefas que serão executadas, qual ação será executada por cada agente, como se dará a troca de informações e o compartilhamento de recursos. (REZENDE, 2003)

(REZENDE, 2003) faz uma relação entre o estudo das interações entre agentes comparando com o estudo feito por (SCHIEL, 2002) referente à interação entre os indivíduos de uma população, aplicando a nomenclatura da biologia ao contexto computacional dos sistemas multiagentes, conforme segue:

- Neutralismo ($0 \leftrightarrow 0$): Não ocorre interação nenhuma entre os agentes.
- Competição ($- \leftrightarrow -$): Nesse caso, há competição entre dois agentes por determinado recurso, e com isso ambos acabam prejudicados.
- Amensalismo ($- \leftrightarrow 0$): Ocorre quando um agente sofre as consequências negativas da ação não proposital de outro agente.
- Parasitismo ($+ \leftrightarrow -$): Um agente parasita se beneficia de outro (hospedeiro), agindo intencionalmente com o intuito de prejudicar o hospedeiro.
- Predação ($+ \leftrightarrow -$): Na interação de predação, o agente predador busca eliminar o agente presa para obter seus objetivos.
- Comensalismo ($+ \leftrightarrow 0$): Essa relação é parecida com o parasitismo, com a diferença que o agente comensal não prejudica o agente hospedeiro.
- Proto-cooperação ($+ \leftrightarrow +$): Numa relação de proto-cooperação dois agentes cooperam para a realização de uma tarefa que estes poderiam realizar sozinho, mas se realizarem em equipe farão essa tarefa de forma otimizada.
- Simbiose ($+ \leftrightarrow +$): Nesse caso, ao contrário da proto-cooperação, os agentes trabalham em conjunto pois precisam fazer isso, ou seja, eles não podem re-

solver determinado problema sozinhos e então recorrem ao trabalho em equipe para fazê-lo.

2.3.1 Tipos de Sistemas Multiagentes

Segundo (Knapik,1998 apud (FERREIRA; GIRARDI, 2000)) existem três arquiteturas básicas para construção de um sistema multiagente:

- Simples: Existe somente um único agente atuando no ambiente.
- Moderada: O sistema é composto por n agentes, que podem ser locais ou estar distribuídos pela rede. Mas todos os agentes do sistema são iguais, logo realizam as mesmas tarefas (podendo pertencer a usuários diferentes).
- Complexa: Nesse tipo de arquitetura os agentes são diferentes entre si, ou seja, possuem funções diferentes, e podem cooperar (ou competir) entre si para resolver algum problema.

Existem ainda outras formas de classificação de Sistemas Multiagentes, que levam em conta o nível de coordenação ou de cooperação apresentado pelos agentes que o compõe.

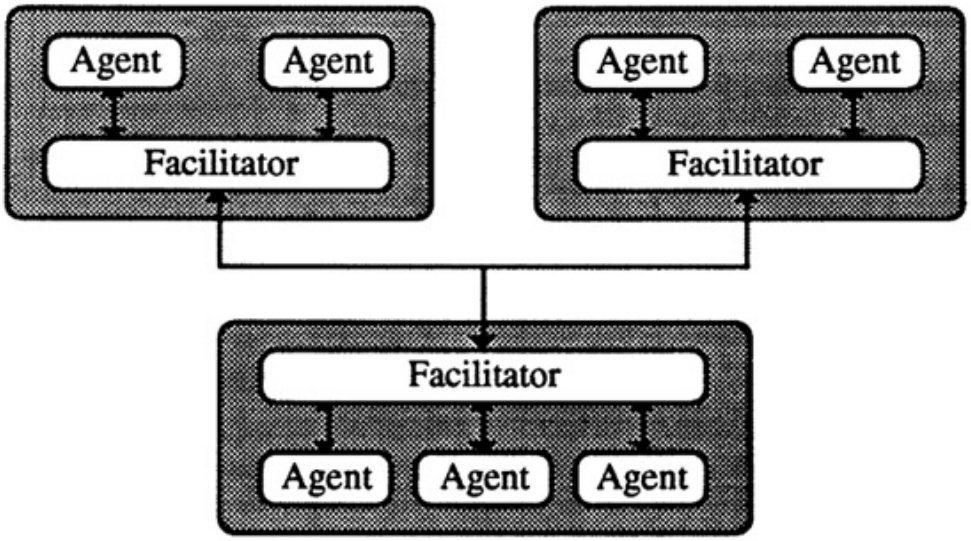
Em sistemas de arquitetura complexa e moderada é necessário que os agentes cooperem entre si para que seja possível resolver determinado problema. (FERREIRA; GIRARDI, 2000) apresentam algumas formas de implementação de cooperação entre agentes:

- Arquitetura quadro-negro: Nesse tipo de arquitetura existe uma estrutura de dados única (quadro-negro) utilizada por todos os agentes para escrever e ler mensagens. Quando um agente necessita de determinado conhecimento ele pode buscar diretamente no quadro-negro e se não encontrar ele registra sua solicitação nessa estrutura e aguarda até outro agente responde-la. Com o aumento do número de agentes que utilizam a estrutura o processo de gravar / recuperar informações pode se tornar complexo e inviabilizar o uso desse tipo de estrutura.
- Arquitetura Troca de Mensagens: Essa arquitetura determina que os agentes troquem mensagens assíncronas diretamente uns com os outros, sem a necessidade de uma estrutura intermediária, podendo haver a presença de um agente que facilite o processo comunicativo.

  Para que esse processo seja implantado, é necessário que os agentes saibam como localizar os agentes que compõem toda a plataforma, através de algum tipo de endereçamento. Além disso, se torna imprescindível o uso de um protocolo de comunicação que estabeleça as regras utilizadas para troca de mensagens.

- Arquitetura Federativa: Esse tipo de arquitetura foi desenvolvida com o objetivo de diminuir a quantidade de mensagens trocadas entre os agentes de uma sociedade. Nela são criados grupos de agentes, onde cada grupo possui um agente facilitador que é responsável por receber uma mensagem e encaminhá-la ao correto destinatário, evitando assim que sejam disparadas mensagens broadcast a diversos agentes. A Figura 2.6 ilustra esse tipo de arquitetura.

Figura 2.6: Arquitetura de Coordenação Federativa



Fonte: (GENESERETH; KETCHPEL, 1994)

Em relação à coordenação de um sistema multiagente (PARAISO (1997) apud FERREIRA; GIRARDI (2000)) apresentam duas possíveis classificações:

- Arquitetura Mestre-Escravo: Nesse tipo de estrutura, existem basicamente dois tipos de agentes: os Mestres que gerenciam o trabalho e distribuem as tarefas, e os escravos que são responsáveis por executar algum processamento e apresentar um resultado ao mestre.
- Mecanismo de Mercado: Em uma arquitetura com mecanismo de mercado todos os agentes conhecem os serviços disponibilizados pelos outros agentes da plataforma e, com isso, solicitam diretamente ao agente que execute tal tarefa para ele diretamente.

2.3.2 Padrões de Construção de Sistemas Multiagentes

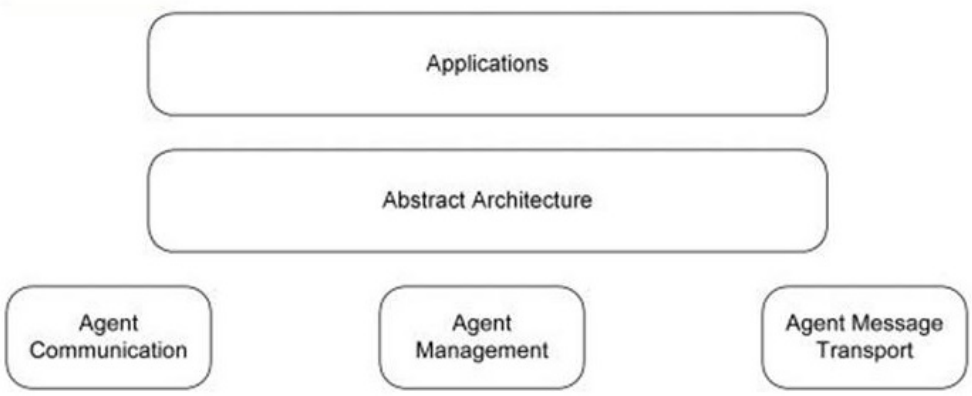
Diversos esforços foram feitos pela comunidade acadêmica para padronizar a construção de Sistemas Multiagentes. A padronização fornece suporte para os agentes executarem, para gerenciarem a sua execução, para acessarem os recursos do sistema, e para garantir a integridade e proteção dos agentes e da própria plataforma.

Seu uso facilita a construção de Sistemas Multiagentes, pois ela prevê diversos mecanismos para que sejam tratadas questões comuns nesse tipo de sistema, como por exemplo, como se estruturarão os agentes e como se dará a comunicação entre eles, facilitando também a comunicação entre diferentes programas (interoperabilidade). (WEISS, 1999), (DICKINSON, 1997)

Entre as diversas formas de padronização destaca-se o padrão proposto pela *FIPA* (*Foundation for Intelligent Physical Agents*), cujo objetivo é fornecer padrões para o desenvolvimento de agentes, entre eles: propor uma terminologia e uma arquitetura básicas para serem referência na construção de agentes, fornecer padronização para determinadas tecnologias de agentes, sendo elas normativas, informativas (informado como se deve aplicar essas normas) e testar a viabilidade dessas normas. (THE FOUNDATION OF INTELLIGENT PHYSICAL AGENTS , FIPA)

No padrão *FIPA* um agente é um software que contém sua própria thread de controle, um estado interno e um certo comportamento. Além disso, ele é capaz de interagir com outros agentes. (POSLAD; BUCKLE; HADINGHAM, 2000) As especificações principais contidas na *FIPA*, se constituem de uma arquitetura abstrata de Sistemas Multiagentes, modelos de gerenciamento de agentes e finalmente uma linguagem de comunicação entre agentes, conforme ilustra a Figura 2.7: (THE FOUNDATION OF INTELLIGENT PHYSICAL AGENTS , FIPA)

Figura 2.7: Classes de Especificações do padrão FIPA



Fonte: (THE FOUNDATION OF INTELLIGENT PHYSICAL AGENTS - FIPA, Acesso em 13 de maio de 2013)

Arquitetura Abstrata: As especificações desse módulo definem as entidades abstratas que são necessárias para que seja possível construir serviços e ambientes de agentes.

Comunicação de Agentes: Essa camada da arquitetura define como se dará a comunicação entre os agentes. Essa categoria lida com a troca de mensagens entre agentes na linguagem *ACL* (*Agent Communication Language*), através de protocolos de interação, das diferentes declarações feitas em ACL (chamadas de Atos de

Comunicação - Communicative Acts), que estabelecem formas mais complexas de comunicação entre agentes e por fim, trata das formas de representação das mensagens em ACL.

Gerenciamento de Agentes: Nesta camada são definidas as formas de gerenciamento dos agentes, seja dentro de uma plataforma ou entre plataformas diferentes.

Transporte de Mensagens de Agente: Definem-se aqui como as mensagens em ACL serão representadas e transportadas em diferentes tipos de protocolos de transporte e de rede.

3 FRAMEWORKS PARA O DESENVOLVIMENTO DE SISTEMAS MULTIAGENTES

Segundo (PREE; SIKORA, 1997) Frameworks são arquiteturas de software semiacabadas que permitem o reuso de código-fonte, componentes simples e principalmente do design da arquitetura proposta pelo framework, conseguindo com isso explorar todo o potencial do desenvolvimento de software Orientado a Objetos e alcançando um nível alto de reusabilidade de código.

Normalmente um framework é representado por um conjunto de classes abstratas e as interações entre essas classes, uma classe abstrata define simplesmente uma interface para determinados métodos, quando um usuário for utilizar determinado framework ele deve criar subclasses dessas classes abstratas que implementem os métodos descritos por elas. (JOHNSON; FOOTE, 1988)

3.1 Frameworks para desenvolvimento de sistemas multiagentes

Existem diversos frameworks disponíveis na Internet que possibilitam a construção de sistemas multiagentes, cada qual com características peculiares. Este trabalho pretende analisar frameworks para desenvolvimento de sistemas multiagentes buscando descrever suas características e compará-los, para futuramente decidir qual se aplica melhor ao problema proposto, e que futuramente participará da arquitetura final. Pelo fato de haver diversos frameworks disponíveis para esse fim, foram escolhidos dois para serem analisados nesse trabalho: *JADE* (*Java Agent Development Kit*) e *JIAC* (*Java-based Intelligent Agent Componentware*).

3.1.1 Framework *JADE*

A plataforma *JADE* (Java Agent Development Framework) é um framework que possibilita o desenvolvimento de Sistemas Multiagentes de maneira facilitada, visto que ele fornece uma *API* (*Application Programming Interface*) amigável e simples, um middleware robusto que permite a distribuição das aplicações de ma-

neira facilitada, além disso, Jade fornece um conjunto de ferramentas que auxiliam o desenvolvimento e depuração desses sistemas.

JADE fornece ao programador um sistema totalmente distribuído, fornecendo comunicação transparente entre os agentes (o programador não sentirá diferença na forma de comunicar agentes locais ou agentes remotos), cumprimento total das especificações da *FIPA*, transporte assíncrono de mensagens de maneira eficiente, implementação do serviço de páginas amarelas e páginas brancas e gerenciamento eficaz do ciclo de vida dos agentes, mobilidade de agentes (possibilidade de migração do código e do estado de um agente entre máquinas e processos de maneira transparente para os demais agentes), mecanismo de subscrição (para que agentes e aplicações sejam notificados de eventos externos), conjunto de ferramentas gráficas, suporte a ontologias e linguagens de representação de conteúdo de maneira automática, biblioteca de protocolos de interação, interface para rodar a plataforma de uma aplicação externa e kernel extensível (permite que se adicionem novos serviços ao kernel da plataforma).

Segundo (BELLIFEMINE; CAIRE; GREENWOOD, 2007) a plataforma *JADE* é fundamentada nas seguintes abstrações de agentes:

- Agentes são entidades autônomas e proativas: Cada agente possui sua própria thread de execução, o que permite que cada um tenha autonomia para decidir quando e quais ações serão executadas. Sendo assim, um agente não pode ser passado como referência a fim de obter ajuda com a execução de determinada rotina.
- Fraco acoplamento entre agentes: A única forma de comunicação entre os agentes é através de mensagens. Elas podem ter um destinatário ou mais, que são identificados por um nome (não se utiliza referências a objetos), podendo um agente enviar mensagens para agentes desconhecidos, pois *JADE* possui mecanismos que permitem, por exemplo, enviar uma mensagem para um agente que forneça determinado serviço.
- Sistema Ponto a Ponto: Globalmente cada agente possui um nome único global *AID* (*AgentIdentifier*), conforme definido nas especificações da *FIPA*, podendo inclusive mudar de plataforma durante a execução do sistema.

No momento em que a plataforma JADE é inicializada, criam-se automaticamente dois agentes especais:

- Sistema de Gerencia de Agentes - *AMS* (*Agent Management System*): Este agente possui como função manter o controle de toda a plataforma, fornecer o serviço de páginas brancas (usado quando um agente deseja buscar por outro agente), e também se responsabiliza por fazer o registro automático dos agentes

quando estes são criados (este procedimento é necessário para que um agente receba seu *AID*).

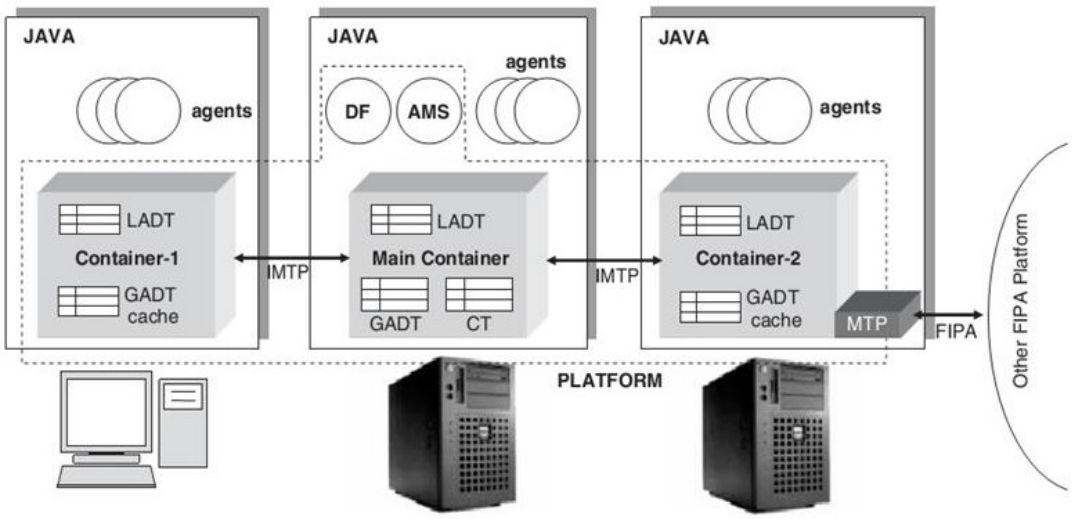
- Facilitador de Diretórios (*DF* (*Directory Facilitator*)): Sua função é fornecer o serviço de páginas amarelas, que é usado quando um agente quer encontrar outro agente que ofereça determinado serviço, podendo também registrar seus próprios serviços ou mesmo ser avisado sempre que um serviço é registrado ou que uma notificação é feita.

A Figura 3.1 ilustra a arquitetura do framework *JADE*: Uma aplicação nessa plataforma é composta de um ou mais containers, que tem como função hospedar e executar agentes. Para que seja possível saber quais são os agentes que integram um container, cada um possui uma tabela local de descritor de agentes - *LADT* (*Local Agent Descritor Table*).

Existe também um container principal, que é responsável por inicializar a plataforma e é através dele que os demais containers ingressam no sistema. Suas principais funções são:

- Ele contém a Tabela de Containers - *CT* (*Container Table*) que armazena os endereços de todos os containers da aplicação.
- Contém também a Tabela Global de Descritor de Agente - *GADT* (*Global Agent Descritor Table*), que possui os registros de todos os agentes da plataforma, incluindo seu status atual e sua localização.
- Hospedar os agentes: *AMS* (Agent Management System) e *DF* (Directory Facilitator) que possuem funções importantes na plataforma.

Figura 3.1: Arquitetura do Framework *JADE*



Fonte: (BELLIFEMINE; CAIRE; GREENWOOD, 2007)

O container principal apesar de ser um centralizador das informações da plataforma, foi desenvolvido para não se tornar um gargalo do sistema com a implementação de caches locais da tabela global de agentes (*GADT*s cache). Em caso de ocorrer alguma falha no container principal, o framework fornece o Main Replication Service que é um serviço que controla o nível de falha, de escalabilidade e distribuição da plataforma.

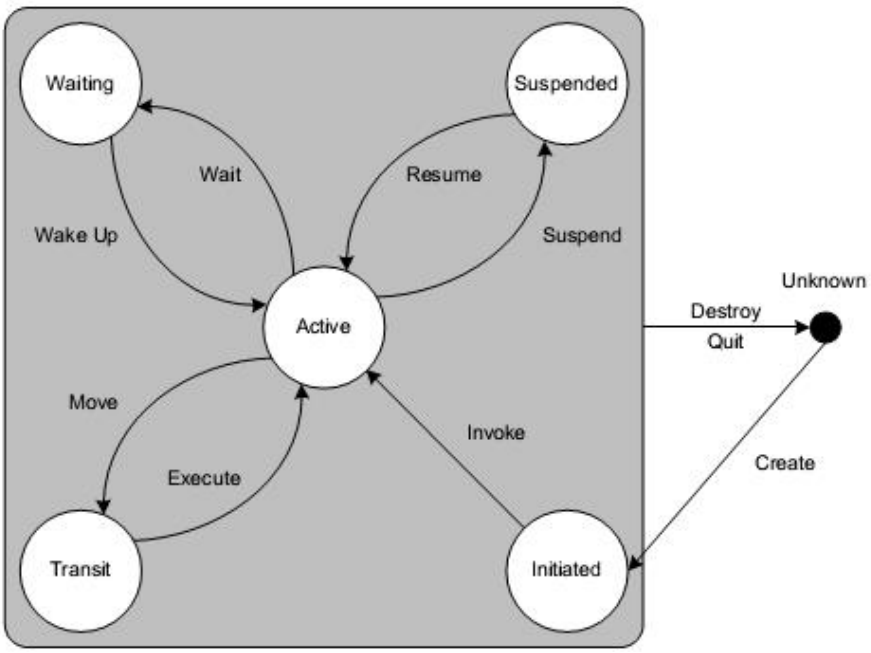
3.1.1.1 Estrutura de um agente na plataforma JADE

Um agente pode ser criado na plataforma *JADE* estendendo-se a classe `Agent` que se encontra no pacote `jade.core`. Além disso, o programador deve implementar o método `setup()` inserindo nesse método os comandos que devem ser executados no momento da inicialização do agente.

A Figura 3.2 ilustra os estados do ciclo de vida de um agente, que são definidos pelo framework *JADE* conforme segue:

- INICIADO: Nesse estado o agente é somente construído, não possui nome, endereço e nem pode se comunicar com outros agentes.
- ATIVO: Quando é ativado o agente recebe um nome, endereço e pode utilizar as funcionalidades do framework.
- SUSPENSO: Quando é suspenso, a thread interna do agente é pausada e ele não executa nenhum comportamento.
- AGUARDANDO: Este estado suspende a thread atual até que determinado evento aconteça, despertando a thread.
- DELETADO: Quando um agente é deletado, ele perde seu registro no *AMS* e sua thread é encerrada.
- TRÂNSITO: Nesse estado um agente móvel está em processo de mudança para outro local. Nesse estado ele ainda pode receber mensagens que serão armazenadas em um buffer e posteriormente serão enviadas à nova localização do agente.

Figura 3.2: Ciclo de vida de um agente na plataforma *JADE*



Fonte: (BELLIFEMINE; CAIRE; GREENWOOD, 2007)

Para que seja possível transitar entre os estados definidos para um agente a classe Agent do framework *JADE* dispõe de métodos que realizam tais transformações:

- `doActivate()`: Transforma o estado SUSPENSO em ATIVO ou AGUARDANDO, dependendo do estado anterior do agente.
- `doDelete()`: Faz a transição de ATIVO, SUSPENSO ou AGUARDANDO para DELETADO, ocasionando na destruição do agente na sequência.
- `doSuspend()`: Transita do estado ATIVO ou AGUARDANDO para SUSPENSO, salvando o estado atual para ser restaurado quando o agente for novamente ativado.
- `doWait()`: Altera o estado ATIVO para AGUARDANDO. Podendo ser passado como parâmetro um valor de tempo para que a thread somente seja suspensa por tempo determinado.
- `doWake()`: Muda o estado de AGUARDANDO para ATIVO.

Quando um agente é inicializado a plataforma controla a execução do código do agente da seguinte forma: executa-se o método construtor do agente, ele recebe um *AID*, em seguida ele é registrado no *AMS*, seu status é alterado para ATIVO e então é executado o método `setup()`, que deve ser usado pelo programador para realizar a inicialização dos parâmetros do agente. Deve-se dar preferência a este método na realização das inicializações ao invés de utilizar o construtor do agente, pois no momento da execução do método `setup()` o agente já encontra-se registrado

no *AMS*, já possui um *AID* e seu status está ATIVO, o que não ocorre no método construtor.

Após o método setup ter sido executado, o agente começa a atuar no ambiente, com a execução de um ou mais comportamentos. Tais comportamentos devem ser criados como uma classe que estende a classe `jade.core.behaviours.Behaviour`. Cabe ressaltar que a execução de um comportamento por um agente ocorre de forma cooperativa e não concorrente. Ou seja: cabe ao programador definir quando um comportamento deve encerrar sua execução para que outro possa executar.

Ao programar um comportamento um programador deve implementar dois métodos, o primeiro deles `action()` é um método que contém o comportamento que o agente deverá executar e o segundo é o método `done()` que avalia se o comportamento foi finalizado e retorna um valor booleano com esta informação.

3.1.2 Framework *JIAC*

A plataforma *JIAC* (Java-based Intelligent Agent Componentware) é um framework para o desenvolvimento de sistemas multiagentes de alto desempenho, de maneira fácil e eficiente, que atualmente encontra-se em sua 5ª versão. *JIAC* foi desenvolvido sobre os seguintes princípios básicos: Distribuição Transparente ao usuário, interação entre seus componentes através de serviços, uso de ontologias para descrever os serviços, mecanismos de segurança e gerência de agentes, facilidades em ambientes distribuídos, tais como migração de agentes, tolerância a falhas, reconfiguração de componentes.

Existem três componentes básicos na plataforma *JIAC* que integram de forma explícita uma aplicação nessa linguagem: Serviços, Regras e Ações. Um serviço pode ser modelado na plataforma através de *BPML* (*Business Process Modeling Language*), sendo posteriormente convertidos para a linguagem da plataforma *Jadl++* (*JIAC agent description language*). Os serviços da plataforma podem ser classificados como: Serviços Compostos que combinam diversos serviços em um só, trazendo melhorias para o sistema e Serviços de Infraestrutura que são serviços especiais que são utilizados por agentes, serviços compostos ou ações. Regras são fatos que desencadeiam ações, serviços ou mudanças na base de fatos do sistema e finalmente, ações são as funções desempenhadas por um agente no sistema.

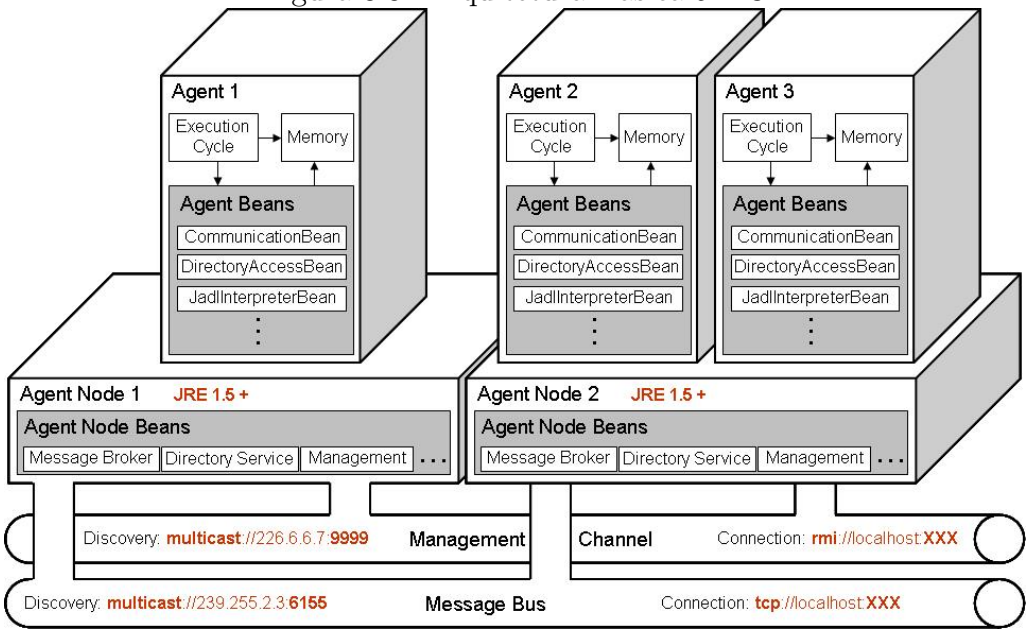
Uma aplicação desenvolvida com o framework *JIAC* é constituída de um ou mais "nodos de agente", que se encontram distribuídos pela rede e que fornecem um ambiente de execução aos agentes da plataforma. É permitida a inserção de novos agentes, serviços ou até mesmo nodos de agente dinamicamente. O framework se utiliza do paradigma *SOA* (*Service Oriented Architecture*) para realizar a comunicação entre agentes, ou seja, toda a interação é feita por chamada de serviço, seja pelo envio de mensagens para um ou mais agentes, seja através de protocolos de

interação complexos.

A arquitetura da plataforma *JIAC*, mostrada na Figura 3.3 é composta por um ou mais nodos de agente (que basicamente são uma *JVM* (*Java Virtual Machine*) com a base necessária para os agentes executarem), e esta por sua vez contêm um número de agentes. Esses agentes são formados por diversos componentes, alguns deles são básicos e comuns a todos os agentes, e outros ainda podem ser adicionados através de AgentBeans, que por sua vez são o nível no qual as funcionalidades são implementadas pelos desenvolvedores.

Por sua vez, um Nodo de agente, possui um NodeBean, que acrescenta funcionalidades a este nodo como um todo, da mesma forma que um AgentBean acrescenta funções em um Agente. Por padrão um NodeBean possui um Diretório que lista os diferentes agentes que compõem este nodo e os diferentes serviços ofertados por cada um. Além disso, ele conta com um MessageBroker, que permite o envio de mensagens entre os nodos da aplicação.

Figura 3.3: Arquitetura Básica JIAC



Fonte: (JIAC - JAVA-BASED INTELLIGENT AGENT COMPONENTWARE, Acesso em 20 de Maio de 2013)

3.1.2.1 Estrutura de um agente na plataforma JIAC

Um agente *JIAC* constitui-se basicamente de um conjunto de componentes cujas funcionalidades variam desde execução de tarefas básicas para o agente até habilidades mais complexas, como comportamento inteligente e interação com o ambiente. A linguagem *Jadl++*(Java Agent Description Language) é utilizada para fazer a descrição de um agente. Essa linguagem é baseada em *ADL* (*Action Description Language*) e na lógica de primeira ordem.

Tanto um agente como seus componentes possuem um estado discreto que se altera conforme o ciclo de vida do agente e é através dele que se determina todo o comportamento do agente. Os possíveis estados de um agente são:

- VOID: Este é o estado inicial e final de todos os agentes / componentes. Um componente nesse estado não é considerado parte de um agente.
- STOPPED: Este é o primeiro estado que deveria ser alcançado por um agente após o estado VOID, pois é nele que é feita a primeira inicialização do agente.
- INITIATED: Nesse estado é executado todo o código necessário antes do agente receber o status de ativo.
- ACTIVE: Quando é ativado o componente passa a ser parte do agente.
- SUSPEND: As atividades do agente / componente são temporariamente suspensas.
- STEPPING: Execução de instruções em modo passo a passo
- SERIALIZED: Este estado indica que o componente está pronto para ser serializado.
- TRANSIENT: O agente está migrando entre duas plataformas.

A alteração do estado de um agente ou de um componente é feita através do método `changeState()`.

3.2 Análise Comparativa dos Frameworks *JADE* e *JIAC*

Antes de detalhar os requisitos da aplicação a ser desenvolvida, será feita uma análise prévia das funcionalidades presentes nos frameworks escolhidos. Esse roteiro foi adaptado de (BORDINI et al., 2009) que define em seu estudo um roteiro de comparação entre frameworks para implementação de Sistemas Multiagentes:

Tabela 3.1: Comparação entre os frameworks *JADE* e *JIAC*

Critério	***JADE***	***JIAC***
Funcionalidade: A linguagem suporta a criação de que tipos de agentes? (Como reativos, cognitivos, etc.)	A plataforma *JADE* prevê uma estrutura básica de agente, que permite sua extensão para qualquer arquitetura de agente.	Permite a criação de três tipos de agente: agentes reativos simples, agentes do tipo *BDI* (*Believe-Desire-Intention*) e agentes com aprendizado e planejamento.
Continua na próxima página		

Tabela 3.1 – Continuação da página anterior

Critério	*JADE*	*JIAC*
Comunicação: A linguagem conta com uma interface de alto nível de comunicação entre os agentes?	A linguagem prevê o uso de mensagens *FIPA ACL*. Ela ainda conta com o envio de mensagens multicast e broadcast.	A comunicação ocorre através de mensagens ou de serviços. Cada agente possui a sua própria caixa de mensagens e ele ainda pode receber mensagens broadcast ou multicast através de grupos de agente.
Simplicidade: A linguagem é fácil de entender e utilizar?	*JADE* não possui uma linguagem própria, por isso basta que o programador se familiarize com sua *API* para utilizá-la.	Além da linguagem Java, o framework possui uma linguagem própria para descrição de agentes (JADL) o que faz com que o programador tenha que se familiarizar com essa linguagem. Além disso, o framework é utilizado conjuntamente com a ferramenta Apache Maven, necessitando do programador conhecimento na configuração e utilização desta.
Expressividade: A linguagem permite a implementação de sistemas em diversos domínios ou ela é feita para algum propósito específico?	O propósito da linguagem *JADE* é o desenvolvimento de aplicações nos mais diversos domínios.	A linguagem script JADL do framework facilita a criação de aplicações orientadas a serviço. Porém, como sua integração com Java é fácil, pode-se desenvolver aplicações em diversos domínios.

Continua na próxima página

Tabela 3.1 – Continuação da página anterior

Critério	*JADE*	*JIAC*
Implantação: É fornecido ao programador documentação suficiente para ele utilizar a plataforma?	*JADE* fornece documentação em seu site, tanto para instalação e configuração da plataforma bem como para a programação (possuindo um manual escrito em português inclusive) e possui um fórum para troca de experiências entre usuários. Além disso, existe informação sobre a linguagem disponível em outros sites na internet, contando ainda com livros publicados sobre a linguagem.	O site do framework conta com um guia de instalação e um guia para programadores, e possui uma descrição da arquitetura e um fórum disponíveis online, além de diversos artigos publicados recentemente sobre o framework. Em alguns pontos essa documentação é deficiente: o guia de instalação prevê familiaridade do programador com o uso da ferramenta Apache Maven, não há muito aprofundamento nas funcionalidades mais avançadas do *JIAC* nos manuais de programação, e os exemplos de código estão incompletos no site do framework.
Portabilidade: A plataforma requer algum tipo de ambiente computacional específico (arquitetura computacional, sistema operacional, biblioteca) para ser utilizado?	Por ser escrito em Java, o framework roda em sistemas que suportem o uso da *JVM*, abrangendo um grande número de dispositivos, de servidores a telefones celulares (que rodem Java Microedition).	O framework é escrito em Java, com o uso de bibliotecas abertas (Spring Framework e ActiveMQ), rodando em sistemas que suportem tais tecnologias. Possui versão para dispositivos móveis (Micro-JIAC).

Continua na próxima página

Tabela 3.1 – Continuação da página anterior

Critério	***JADE***	***JIAC***
Conformidade com Padrões: O framework é aderente a algum padrão de construção de sistemas multiagente (como FIPA, MASIF, etc)?	*JADE* possui suporte total às especificações contidas na *FIPA*.	*JIAC* suporta parcialmente os padrões da *FIPA*.
Ferramentas Disponibilizadas: Que ferramentas são disponibilizadas com a plataforma para a gerência, log, monitoramento e depuração das aplicações desenvolvidas?	*JADE* oferece uma interface de administração do sistema chamada Remote Management Agent, que possui agentes de depuração (Dummy Agent, Sniffer Agent, Introspector Agent e Log Manager Agent).	Disponibiliza um plug-in para a *IDE* (*Integrated Development Environment*) Eclipse chamado *JIAC* Toolipse que é composto por: Visual Service Design Tool (Editor *BPMN* (*Business Process Model and Notation*) utilizado para modelar e criar serviços no framework), Agent World Editor (Editor Gráfico utilizado para projetar sistemas multiagentes) e JADL Edit (Editor de código fonte para a linguagem JADL).
Integração de Aplicações: Em aplicações existentes, quais ferramentas são utilizadas para realizar a integração com os programas escritos com o framework?	*JADE* oferece integração com os mais diversos domínios de aplicação: Sistemas de Produção de Regras (Jess e Drools), Tecnologias Web (servlets, *JSP* (*Java Server Pages*) e Applets) e ferramentas de gestão de Ontologias (Protegé e Jena).	*JIAC* possui suporte a Webservices, tratando cada nodo de agente como um webserver e permitindo a criação de interfaces através de *JSP*s e utilizando para comunicação *JMS* (*Java Message Service*).

Continua na próxima página

Tabela 3.1 – Continuação da página anterior

Critério	*JADE*	*JIAC*
Performance: Qual o número máximo de agentes que rodam na plataforma de maneira eficiente em uma única instância da plataforma? Há um limite no número de mensagens com as quais a plataforma pode lidar?	*JADE* permite a criação de milhares de agentes, podendo estes trocarem grandes quantidades de mensagens entre eles.	É possível rodar cerca de 10 mil agentes em uma máquina local.
Versão: Qual é o estado atual da plataforma? (protótipo, versão beta, estável)? A plataforma sofre atualizações ou foi descontinuada?	A versão mais recente de *JADE* é a 4.3.0 (estável). Sua última atualização ocorreu em 29 de março de 2013, e o framework sofre atualizações anuais.	A última versão estável do framework é a 5.1.3 de outubro de 2012, e encontra-se em fase beta a versão 5.1.4. Ele sofre atualizações constantes.
Organização do Sistema Multiagente: Como se dá a organização dos agentes da plataforma? (Controle e estrutura organizacional dos agentes)	Possui controle centralizado (cada plataforma de agente é gerenciada pelo agente *AMS*), seguindo a especificação da *FIPA*.	O framework permite a criação de grupos de agentes e o controle distribuído dos agentes (pois cada nodo de agente controla os agentes que estão inseridos nele e uma aplicação é formada por um ou mais nodos).
Funcionalidades para Sistemas Multiagentes: A plataforma oferece bibliotecas extras para a construção de Sistemas Multiagentes? (protocolos de interação, modelos para a construção de agentes, componentes de agente reutilizáveis).	Pelo fato de serem escritos em Java, todos os componentes da plataforma *JADE* são reutilizáveis. Além disso, sua comunidade de usuários constrói e compartilha bibliotecas e extensões para serem utilizadas para os mais diversos fins.	Encontram-se ainda em fase de desenvolvimento.

Continua na próxima página

Tabela 3.1 – Continuação da página anterior

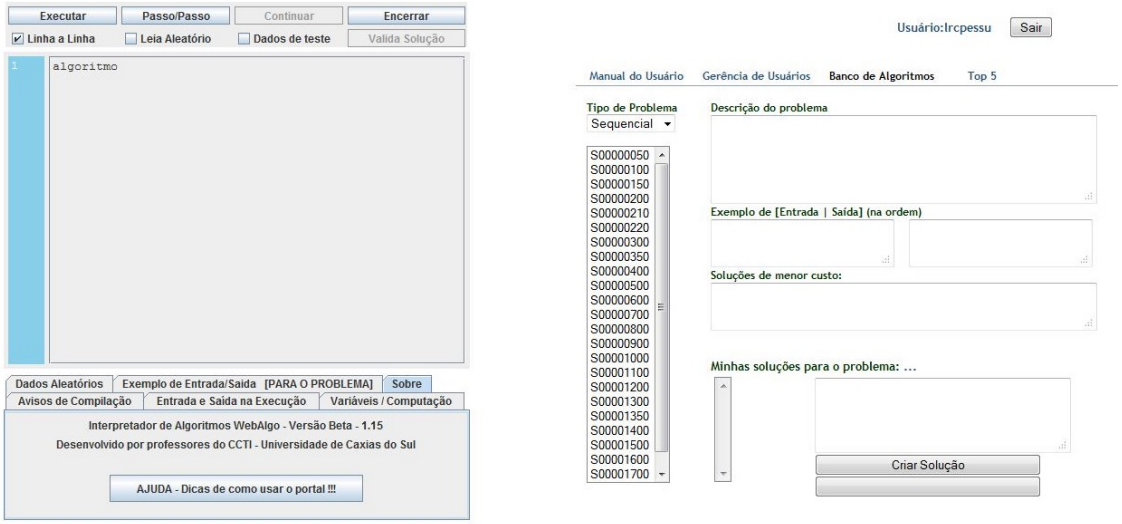
Critério	***JADE***	***JIAC***
Aplicações Típicas: Que tipo de aplicações tem sido desenvolvidas com o uso da plataforma? (Toy problems, solução de problemas reais, aplicações industriais)	Usado para desenvolver tanto aplicações reais como para a indústria.	Usado para desenvolver tanto aplicações reais como para a indústria. Suas principais aplicações incluem simulação e execução de serviços, controle de energia e automação residencial.
Domínio Foco: A plataforma foi desenvolvida com vistas a atingir um domínio de aplicação específico?	A plataforma cobre um domínio extenso de aplicações, tais como: suporte a trabalho colaborativo, e-learning, gerência de redes de computadores, entretenimento, gestão do conhecimento, aplicações industriais.	A plataforma independe de domínio, mas seu uso se volta a aplicações orientadas a serviço, ambientes de telecomunicação e aplicações industriais.

4 PORTAL DE ALGORITMOS DA UCS (ALGOUCS)

O Portal de Algoritmos da *UCS* nasceu da necessidade de um objeto de aprendizagem que auxiliasse os professores da disciplina de Algoritmos da universidade em suas aulas. Ele foi desenvolvido pelos professores Ricardo Vargas Dorneles e Delcino Picinin Júnior no ano de 2009. Atualmente a ferramenta conta com 1.101 usuários em sua base de dados, 152.000 algoritmos cadastrados e atende a turmas de algoritmos dos cursos de Ciência da Computação, Sistemas de Informação e Tecnologias Digitais do Centro de Computação e Tecnologia da Informação (CCTI) e também atende aos cursos de engenharia do Centro de Ciências Exatas e Tecnologia (CCET) e aos alunos do Ensino Médio do CETEC da universidade.

Pode-se acessar o Portal de Algoritmos pelo endereço vposeidon9.ucs.br. Ao efetuar login no sistema o usuário depara-se com a tela apresentada na Figura 4.1.

Figura 4.1: Tela Inicial do Sistema



O sistema encontra-se dividido em duas partes: ao lado esquerdo encontra-se o editor / compilador de algoritmos e ao lado direito tem-se o banco de problemas e o gerenciador de usuários. Nesse ambiente um usuário pode inserir algoritmos

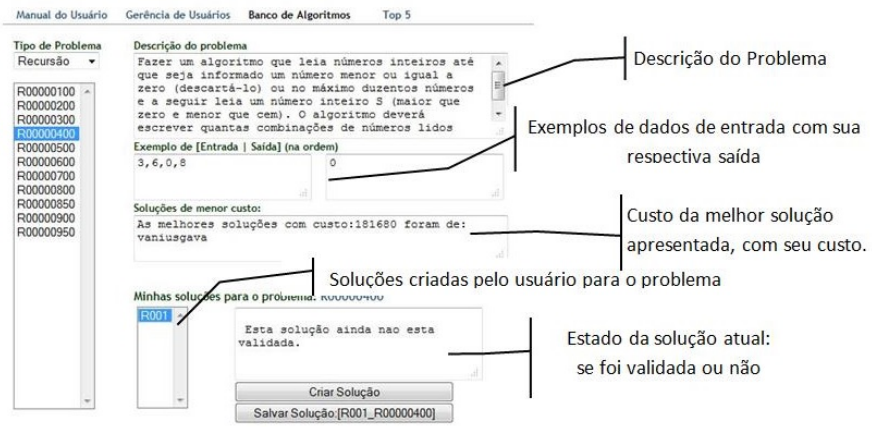
no editor e executá-los, sempre respeitando a sintaxe da linguagem Português Estruturado suportada pela ferramenta. Caso ocorra algum erro de compilação, será mostrada uma mensagem na aba Avisos de Compilação. O editor conta ainda com outras funcionalidades frequentemente presentes em outras *IDE*s: execução passo a passo, execução de algoritmos com dados aleatórios, uso de breakpoints e realce de código.

Além das funções de *IDE*, o Portal de Algoritmos é um objeto de aprendizagem sendo que ele conta com um banco de problemas que é dividido em diversas categorias, de acordo com a sequência didática de conteúdos adotado nas disciplinas de algoritmos da universidade: sequencial, condicional, iterativo, vetor, matriz, função, recursão e geral. Os usuários do sistema podem submeter suas soluções à correção da ferramenta, que além de avaliar a corretude dessa solução, calcula o número de instruções executadas pelo algoritmo criado e com essa informação ele faz um ranking das melhores soluções apresentadas pelos usuários, o que mostrou-se uma forma de estimular aos alunos, conforme expõe (DORNELES; JR.; ADAMI, 2011):

> A identificação dos algoritmos mais eficientes se tornou um importante mecanismo motivador. Estudantes eram estimulados a procurar por algoritmos mais eficientes para superar o desempenho da melhor solução (em termos de número de operações) até o momento em cada problema. Pode-se notar isso pelas repetidas submissões do mesmo algoritmo com número decrescente de operações executadas.

Para inserir uma solução o usuário deve seguir os seguintes passos: Primeiramente ele deve selecionar um dos problemas disponíveis, em seguida ele deve clicar no botão criar solução. Nesse momento o editor é habilitado e possibilita a inserção do algoritmo, além disso outras informações são mostradas no banco de algoritmos como mostra a Figura 4.2:

Figura 4.2: Informações contidas no Banco de Problemas



Após digitar a sua solução, o usuário pode executar seu algoritmo, para verificar se ele atende ao que foi solicitado clicando no botão Executar do editor de algoritmos, que efetuará a análise léxica e sintática e disparará a execução do algoritmo. Além disso, são oferecidas três opções de execução ao usuário: Linha a Linha, Leia Aleatório e Dados de Teste. Se a opção linha a linha estiver marcada o algoritmo executará normalmente, mas será mostrada no editor qual a linha está sendo executada no momento, e será feita uma pequena pausa após a execução de cada instrução.

Marcando as opções Leia Aleatório ou Dados de Teste (pode-se somente marcar uma destas opções ou nenhuma delas) a ferramenta não solicita a inserção de dados de entrada para o algoritmo que está sendo executado, sendo gerados valores aleatórios para as variáveis (no caso da opção Leia Aleatório) ou serão usados os dados de teste do exemplo do problema no banco de algoritmos, ou mesmo inserindo os próprios dados de teste fornecidos pelo usuário na aba correspondente no editor de código.

O usuário pode optar pela execução Passo a Passo do algoritmo, clicando no botão com este nome ao invés de escolher o botão Executar. Nesse modo de execução, uma instrução só será executada quando o usuário clicar no botão Passo a Passo. Tal modo de execução é útil quando se deseja avaliar o estado do programa durante sua execução. Se o usuário desejar executar normalmente o algoritmo a partir de certo ponto, ele pode clicar no botão Continuar, que executará normalmente o algoritmo a partir daquele ponto.

Quando desejar submeter sua solução à correção o usuário deve clicar no botão "Valida Solução". Se o algoritmo estiver correto, ou seja, para as entradas de teste ele produzir as saídas adequadas, então será mostrada a mensagem "Executou corretamente todas as instâncias" na aba "Avisos de Compilação". Caso contrário serão mostradas mensagens informando algum erro de compilação, no caso de haver algum problema de sintaxe com o algoritmo, ou ainda mostrará a mensagem "Valor gerado é diferente do esperado", caso as saídas geradas para os dados de teste sejam diferentes do esperado, ou ainda ocorrerá um erro de execução, como por exemplo a leitura de valores a mais ou a menos do que o esperado.

A qualquer momento o usuário pode salvar no banco de problemas sua solução, bastando clicar no botão "Salvar Solução: [Nome da solução]", onde Nome da Solução corresponde ao nome dado ao algoritmo. Se o algoritmo estiver validado, será mostrado uma caixa de texto que informará ao usuário o custo de sua solução.

4.1 Estrutura do Sistema

O Portal de Algoritmos foi escrito em duas linguagens de programação diferentes: o editor de código / compilador[1] é escrito em Java e roda no Browser através de um applet. Já o banco de problemas e o gerenciador de usuários foram escritos em Python com o uso do framework Django. A aplicação conta também com um banco de dados MySQL que armazena as descrições dos problemas existentes na ferramenta, os usuários do sistema e os problemas resolvidos por cada usuário.

Com base na análise realizada no código-fonte do Portal de Algoritmos, destacam-se as seguintes entidades componentes da ferramenta atualmente:

- Compilador: Composto por um *scanner* e um *parser*, ele é responsável por fazer a análise léxica e sintática dos algoritmos escritos pelo usuário e também por executar tais algoritmos.
- Interface Gráfica do Usuário: Responsável pela interação entre o usuário e o sistema. Ela é composta pelo editor de código, que é escrito em Java, e também por formulários *HTML* (*HyperText Markup Language*), que contém o cadastro de usuários, os problemas contidos na ferramenta e também o ranking de usuários, e a interface para os professores e administradores do portal.
- Gerenciador de Problemas: Parte do sistema que possibilita manter os problemas da plataforma, fornecendo as operações de inserção, alteração, consulta de problemas na plataforma.
- Gerenciador de Usuários: Permite a inserção de novos usuários ao sistema, bem como permite a definição das permissões destes, e também possibilita o gerenciamento dos problemas resolvidos por cada usuário e o ranking de usuários.
- Ajuda do Sistema: Documento que auxilia o usuário principalmente no que diz respeito à linguagem algorítmica.

Após o levantamento das entidades componenentes da plataforma foram definidos quais são as funcionalidades presentes no sistema, juntamente com a entidade que é responsável pela sua execução:

[1] O termo Compilador é utilizado somente para facilitar a nomeclatura da estrutura responsável por realizar a análise léxica e sintática do código fonte do Portal de Algoritmos. É sabido que a ferramenta não gera um executável e que a execução dos algoritmos é simulada pela ferramenta

Tabela 4.1: Funcionalidades presentes no Portal de Algoritmos

Item	**Funcionalidade**	**Responsável**
1	Reconhecimento de Sintaxe	Compilador e Interface Gráfica
2	Execução Normal	Compilador
3	Execução Passo a Passo	Compilador e Interface Gráfica
4	Encerramento (de um problema em execução)	Compilador
5	Execução com dados de teste	Compilador
6	Execução com dados aleatórios	Compilador
7	Validação de uma solução	Compilador
8	Execução de um algoritmo com Breakpoints	Compilador e Interface Gráfica
9	Calcular o número de instruções executadas	Compilador
10	Ajuda	Ajuda
11	Inserção de Usuários	Gerenciador de Usuários
12	Trocar a senha de um usuário	Gerenciador de Usuários
13	Envio de nova senha (Esqueci minha senha)	Gerenciador de Usuários
14	Ranking de Usuários	Gerenciador de Usuários
15	Manutenção e Consulta de Problemas resolvidos por cada usuário	Gerenciador de Usuários
16	Alteração dos dados de um usuário	Gerenciador de Usuários
17	Consulta aos problemas que foram resolvidos por um aluno	Gerenciador de Usuários
18	Consulta aos alunos que resolveram determinado problema	Gerenciador de Usuários
19	Banco de Algoritmos: operações de inserção, alteração e consulta	Gerenciador de Problemas
20	Interação com o usuário	Interface Gráfica

4.2 Cenários de uso do Portal de Algoritmos

Sabe-se que existem quatro perfis básicos de usuários do Portal de Algoritmos: Usuários eventuais, que utilizam a plataforma sem ter efetuado login no sistema, Usuários do Sistema, que efetuaram login na plataforma, professores e administra-

dores do sistema. Abaixo foram criados cenários de utilização, que demonstram de que maneira cada grupo de usuários se relaciona com o Portal de Algoritmos.

4.2.1 Usuário Eventual

Qualquer pessoa conectada à internet pode acessar e utilizar o Portal de Algoritmos, sendo que ela pode inserir algoritmos na ferramenta e executá-los. Mas se ela não possuir um cadastro no sistema, ela não poderá resolver os problemas cadastrados na base de dados, nem salvar os algoritmos que digitar na plataforma. Ou seja, ela terá acesso somente às funções de *IDE* que o Portal de Algoritmos oferece.

4.2.2 Usuário do Sistema

Quando estiver de posse de um login e senha para o Portal, o usuário amplia as funcionalidades contidas na plataforma. Com isso, além das funções de *IDE* é possível que um usuário resolva os problemas da ferramenta e submeta suas soluções à análise dela. Além disso, o usuário poderá salvar as soluções que forem criadas e acessá-las posteriormente e pode comparar seu desempenho com o de outros usuários através da ferramenta Ranking presente na plataforma.

4.2.3 Professor

Um professor pode utilizar as funcionalidades de gerenciamento para avaliar o desempenho de seus alunos, com isso ele tem acesso a uma interface que permite que ele consulte todas as soluções postadas para cada problema da ferramenta, e também aos problemas resolvidos por cada usuário do sistema. Há a possibilidade de filtrar os problemas e os usuários, diminuindo assim a quantidade de dados apresentados. Mas não há o conceito de turma nessa interface, logo um professor que use a plataforma verá todos os usuários cadastrados no sistema. Além disso, não há a integração entre a ferramenta e o portal, necessitando que o professor acesse a aplicação em um espaço externo ao Portal de Algoritmos.

4.2.4 Administrador do Sistema

Os administradores do sistema possuem acesso a uma interface que permite que se gerencie os problemas cadastrados, permitindo a inserção de novos problemas, sua alteração ou a adição de novos dados de teste ao conjunto de dados já existente. Além disso, eles podem alterar as permissões dos usuários na plataforma, diferenciando assim os professores, os alunos e os administradores.

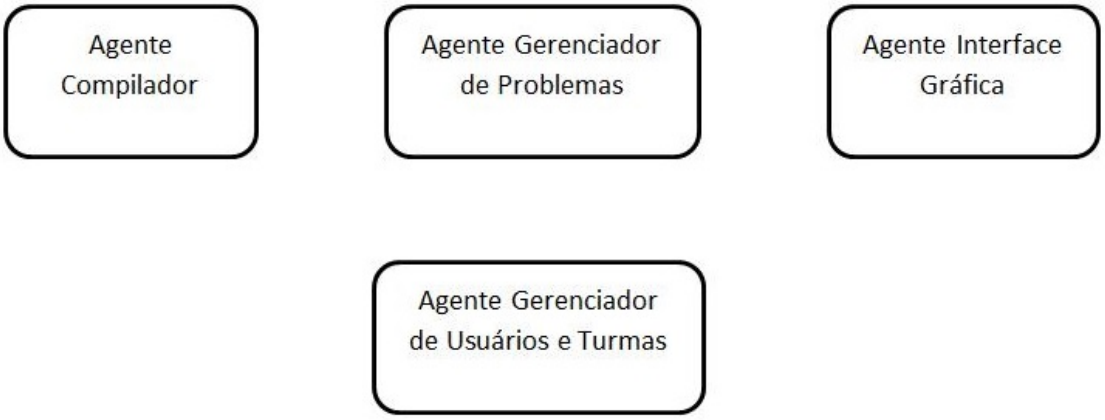
5 ARQUITETURA MULTIAGENTE PARA O PORTAL DE ALGORITMOS

Baseando-se no referencial teórico levantado anteriormente no capítulo 2, no estudo dos frameworks para implementação de sistemas multiagentes apresentado no capítulo 3 e na análise do Portal de Algoritmos descrita no capítulo 4, foi elaborada arquitetura multiagente, apresentada na próxima seção, para modelar o problema do Portal de Algoritmos da UCS.

5.1 Estrutura de Agentes

Depois do estudo realizado no Portal de Algoritmos, identificando as principais funcionalidades da plataforma e as entidades presentes no sistema, foram identificados os seguintes agentes para a arquitetura do sistema:

Figura 5.1: Agentes propostos para o Portal de Algoritmos



Foram mantidas as mesmas entidades componentes identificadas na análise da estrutura atual do Portal de Algoritmos inserindo-se o conceito de Turma na sociedade de agentes. Tal atitude foi motivada devido ao fato de que um professor que acompanha seus alunos através da interface do portal tem acesso a todos os usuários

cadastrados na plataforma, sendo custoso para o professor encontrar seus alunos entre os 900 usuários cadastrados no sistema, e principalmente pela ferramenta não oferecer uma forma de comparação de desempenho que permita ao professor avaliar o desempenho de cada aluno perante a sua turma. Esses dados podem servir como uma importante ferramenta para o professor das disciplinas de algoritmos, tendo em vista que ele pode fazer um planejamento de aula diferenciado para cada turma, com enfoque nas dificuldades apresentadas por ela no uso do Portal de algoritmos e ainda permite que o professor realize um atendimento especializado para cada aluno, a partir da análise de seu desempenho individual.

Definidos os agentes que farão parte da plataforma, será feita uma descrição das responsabilidades gerais de cada um deles, bem como a definição dos comportamentos que deverão ser implementados.

5.2 Descrição dos Agentes componentes da Arquitetura

5.2.1 AGENTE 01: Agente Interface Gráfica

O Agente Interface Gráfica será responsável pelo controle de execução de toda a plataforma, visto que ele controla toda a interação com o usuário. Ou seja: para cada comportamento dos demais agentes, ele capturará um evento de Interface Gráfica e ativará tal comportamento no agente correspondente, passando os argumentos necessários, e aguardará um retorno da execução deste agente para apresentar o resultado ao usuário.

Os comportamentos desse agente não serão detalhados aqui, porque eles estão descritos juntamente com os demais comportamentos nas seções seguintes.

5.2.2 AGENTE 02: Agente Compilador

Esse agente se ocupa de realizar a análise léxica e sintática dos algoritmos inseridos no editor de algoritmos, bem como da execução de um algoritmo e do cálculo do número de instruções executadas. Ele engloba as funcionalidades nº 1 ao 9 descritas nas funcionalidades da plataforma de algoritmos (tabela 4.1). Os comportamentos presentes nesse agente são:

5.2.2.1 Comportamento 01: Execução Normal

Restrições: Nenhuma
Entrada:

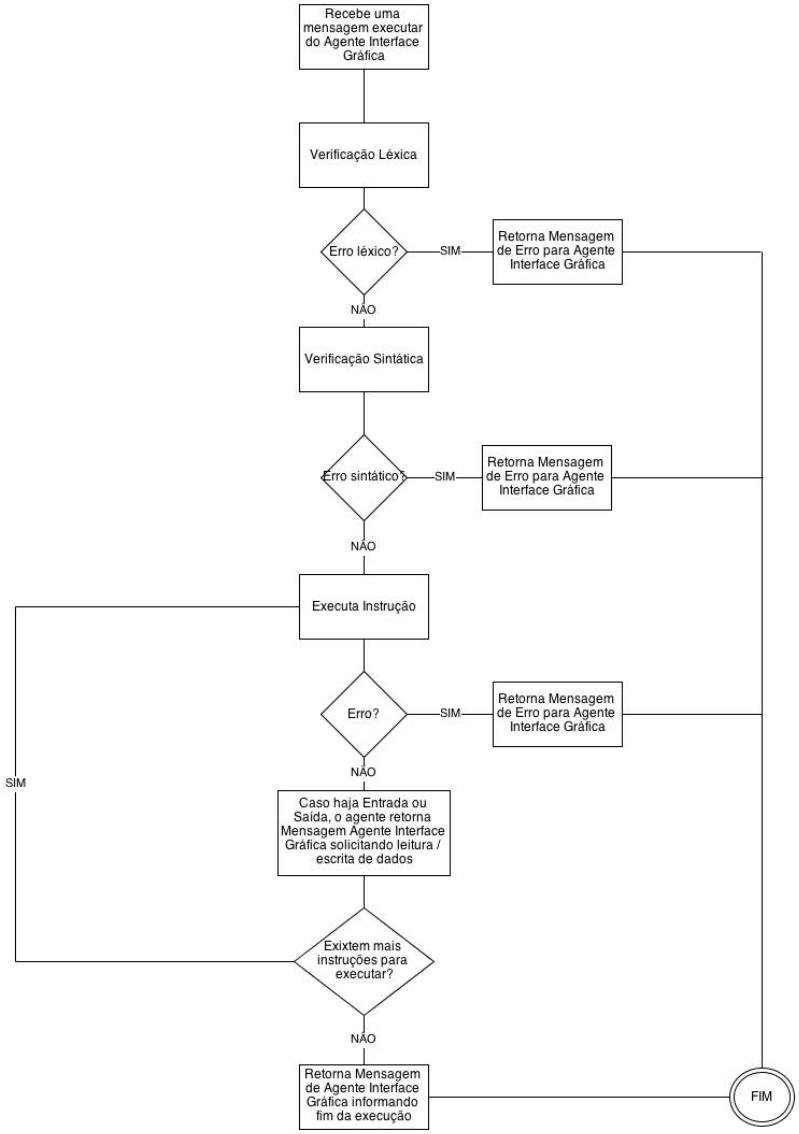
1. Mensagem do agente Interface Gráfica
2. Algoritmo a ser executado
3. Identificação de Execução Normal

Saída:

1. Execução com sucesso do algoritmo
2. Erro Léxico ou Sintático
3. Erro em Tempo de Execução

Comportamento:

Figura 5.2: Comportamento: Execução Normal



5.2.2.2 Comportamento 02: Execução Passo a Passo

Restrições: Nenhuma

Entrada:

1. Mensagem do agente Interface Gráfica
2. Algoritmo a ser executado

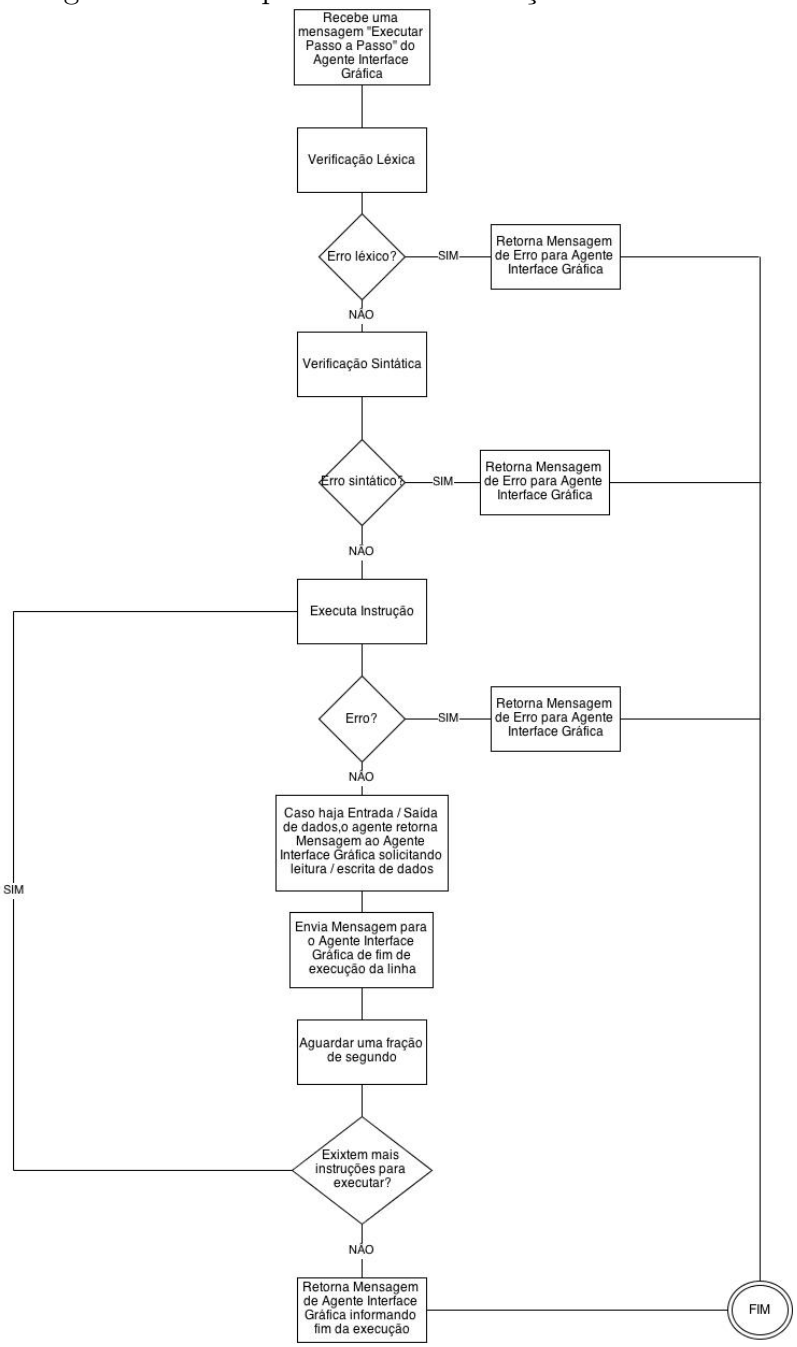
3. Identificação de execução passo a passo

Saída:

1. Execução com sucesso do algoritmo
2. Erro Léxico ou Sintático
3. Erro em Tempo de Execução

Comportamento:

Figura 5.3: Comportamento: Execução Passo a Passo



5.2.2.3 Comportamento 03: Execução com Dados de Teste

Restrições: Nenhuma
Entrada:

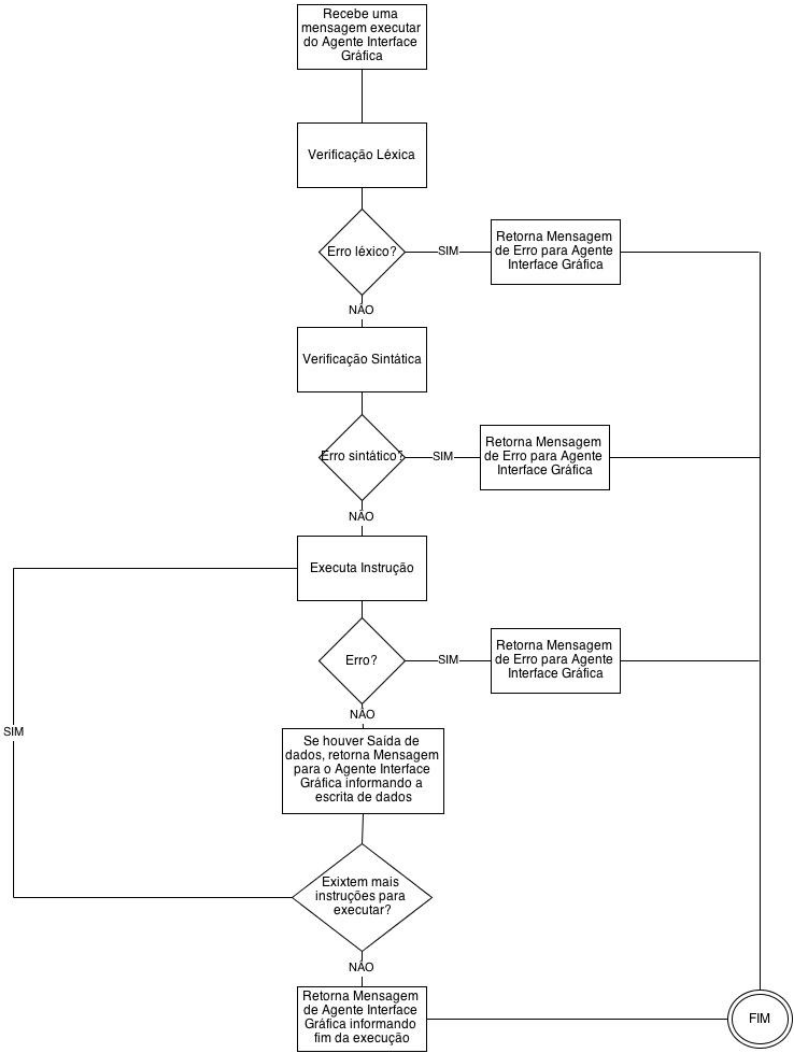
1. Mensagem do agente Interface Gráfica
2. Algoritmo a ser executado
3. Conjunto de Dados de Teste
4. Identificação de execução com Dados de Teste

Saída:

1. Execução com sucesso do algoritmo
2. Erro Léxico ou Sintático
3. Erro em Tempo de Execução

Comportamento:

Figura 5.4: Comportamento: Execução com Dados de Teste



5.2.2.4 Comportamento 04: Execução com Dados Aleatórios

Restrições: Nenhuma
Entrada:

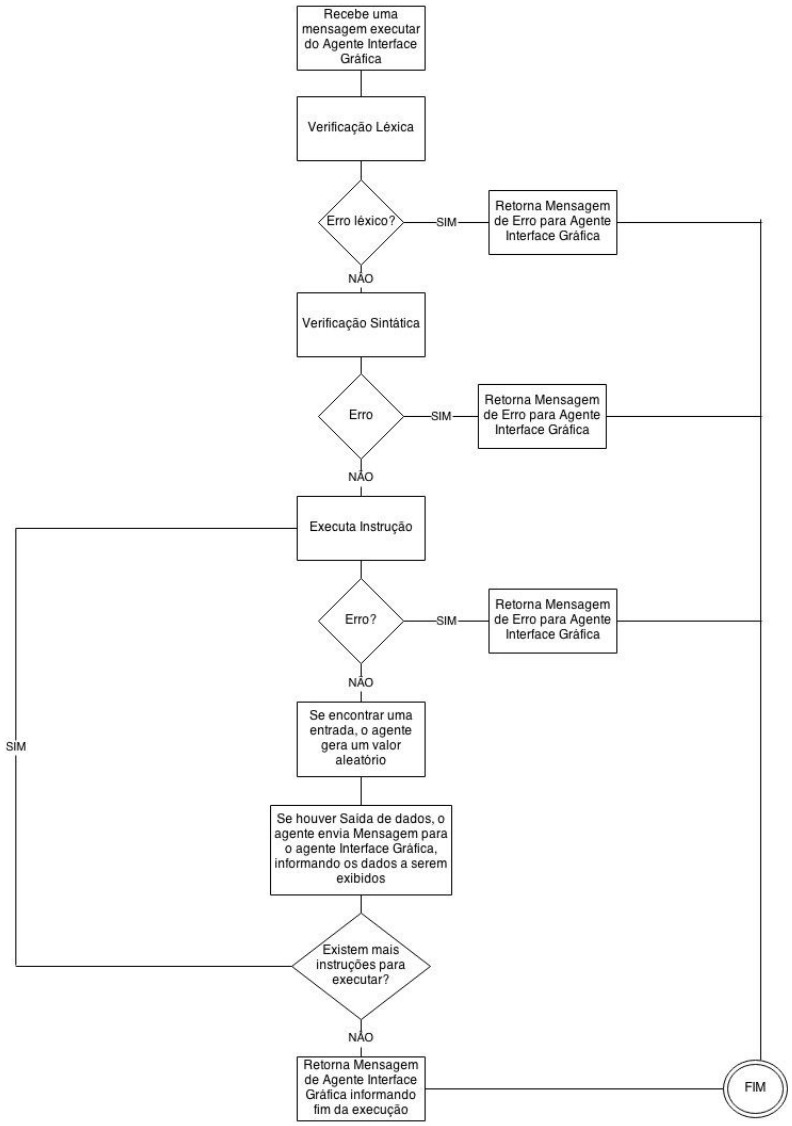
1. Mensagem do agente Interface Gráfica
2. Algoritmo a ser executado
3. Identificação de execução com Dados Aleatórios

Saída:

1. Execução com sucesso do algoritmo
2. Erro Léxico ou Sintático
3. Erro em Tempo de Execução

Comportamento:

Figura 5.5: Comportamento: Execução com Dados Aleatórios



5.2.2.5 Comportamento 05: Execução com Breakpoints

Restrições: Nenhuma
Entrada:

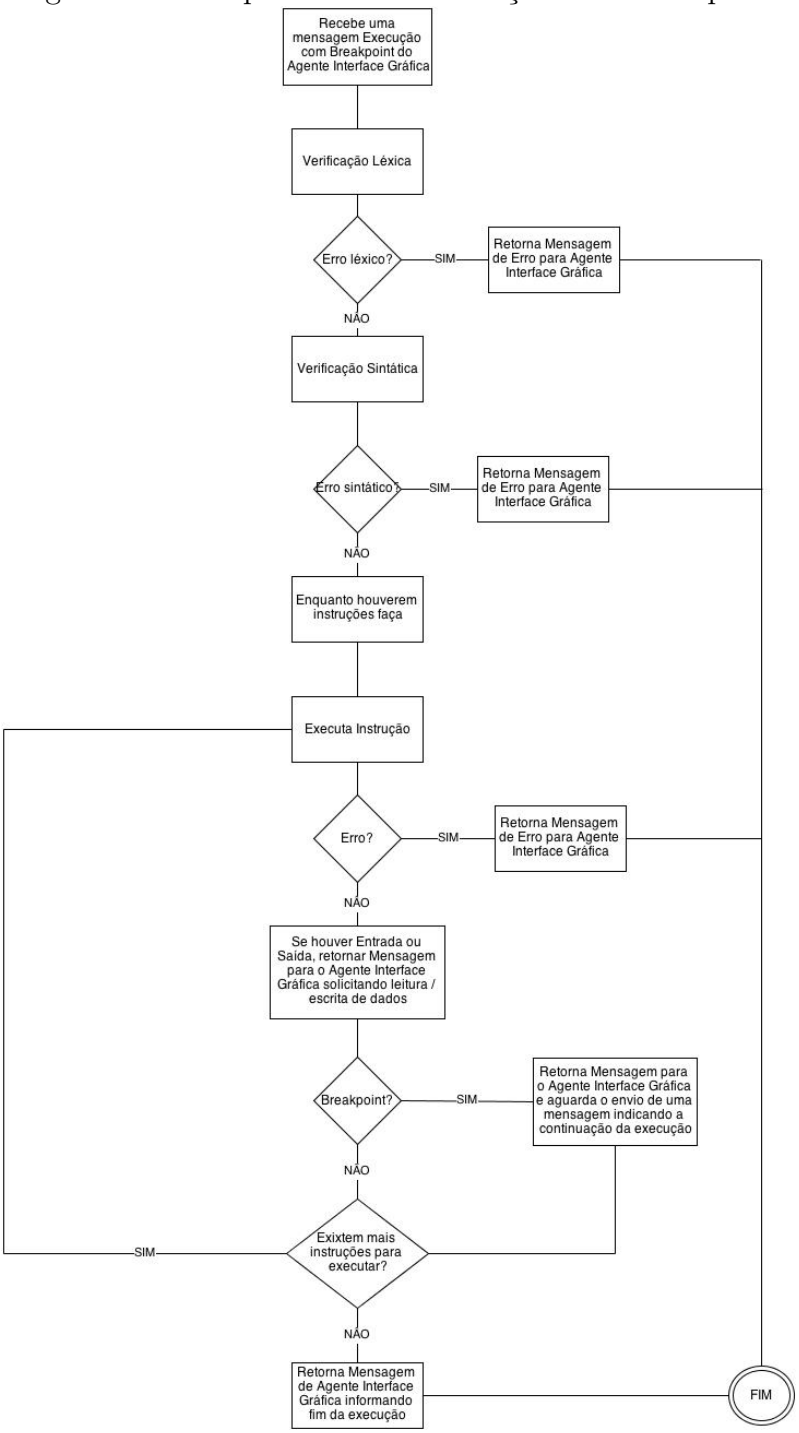
1. Mensagem do agente Interface Gráfica
2. Algoritmo a ser executado
3. Lista de linhas que contém um breakpoint
4. Identificação de execução com breakpoints

Saída:

1. Execução com sucesso do algoritmo
2. Erro Léxico ou Sintático
3. Erro em Tempo de Execução

Comportamento:

Figura 5.6: Comportamento: Execução com Breakpoints



5.2.2.6 Comportamento 06: Encerramento de um Problema em Execução

Restrições: Deve haver algum algoritmo em execução (em qualquer modo)
Entrada:

1. Mensagem do Agente Interface Gráfica

Saída:

1. Encerramento da Execução

Comportamento:

Figura 5.7: Comportamento: Encerramento de um Problema em Execução

5.2.2.7 Comportamento 07: Validação de Solução

Restrições: Nenhuma
Entrada:

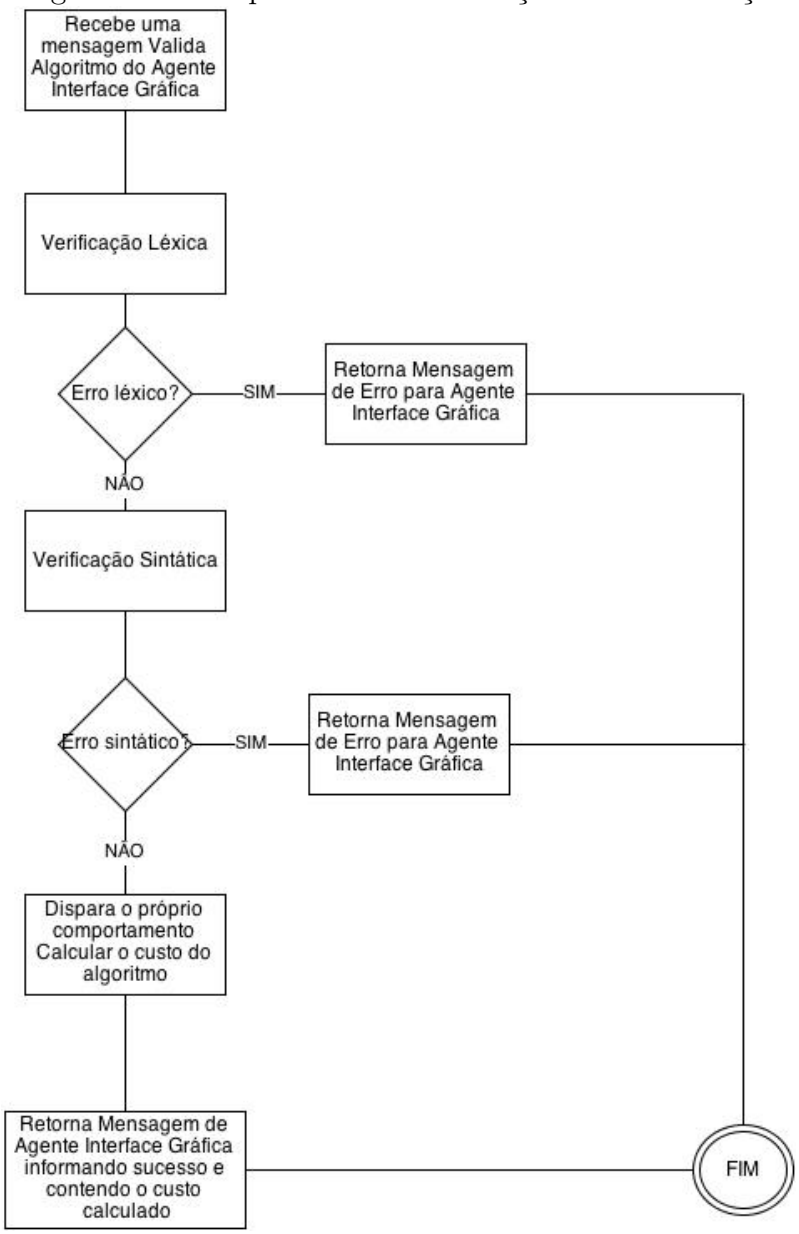
1. Mensagem do Agente Interface Gráfica
2. Algoritmo a ser executado

Saída:

1. Resultado da Validação do Algoritmo (Sucesso ou Falha)
2. Custo da solução (se algoritmo for validado com sucesso)
3. Erro Léxico, Sintático ou de execução (Se não for possível validar)

Comportamento:

Figura 5.8: Comportamento: Validação de uma Solução



5.2.2.8 Comportamento 08: Cálculo do número de instruções executadas

Restrições: Nenhuma
Entrada:

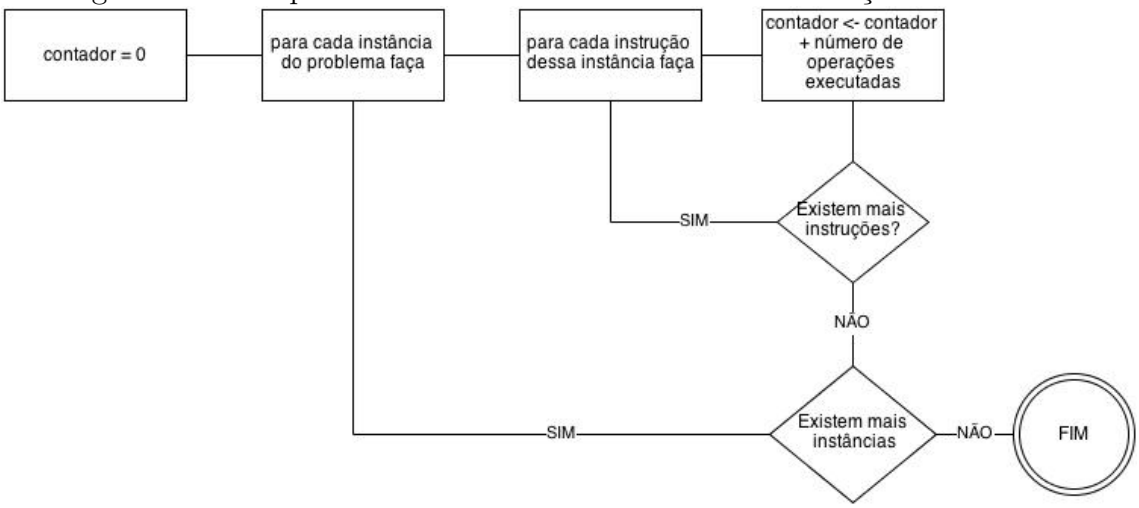
1. Algoritmo que se deseja calcular

Saída:

1. Inteiro que representa a quantidade de instruções executadas

Comportamento:

Figura 5.9: Comportamento: Cálculo do número de instruções executadas



5.2.3 AGENTE 03: Agente Gerenciador de Problemas

O Agente Gerenciador de Problemas será o responsável pelas ações de inserção, alteração e recuperação de problemas na base de dados do Portal de Algoritmos (funcionalidade 19 da tabela 4.1) e o fornecimento de informações para outros agentes na plataforma.

5.2.3.1 Comportamento 01: Inserção de um novo Problema

Restrições: Usuário deve ser do tipo Administrador ou Professor
Entrada:

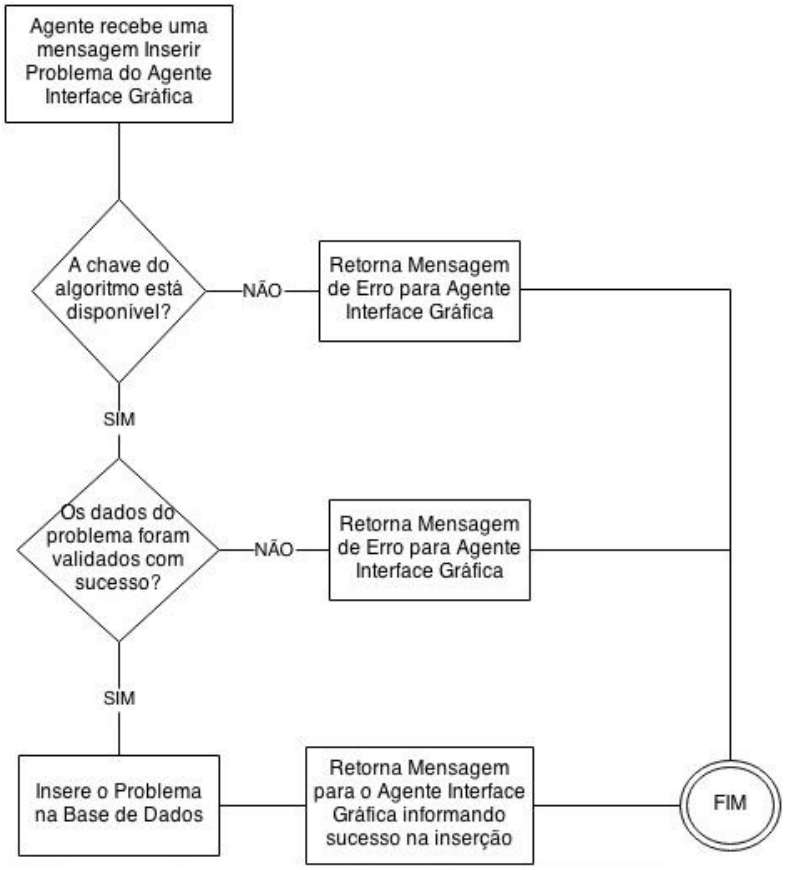
1. Mensagem do Agente Interface Gráfica
2. Dados do novo problema

Saída:

1. O problema é cadastrado com sucesso
2. Erro: Código do problema já utilizado
3. Erro: Falta de alguma informação do Problema

Comportamento:

Figura 5.10: Comportamento: Inserção de um novo Problema



5.2.3.2 Comportamento 02: Recuperar informações de um problema

Restrições: Nenhuma
Entrada:

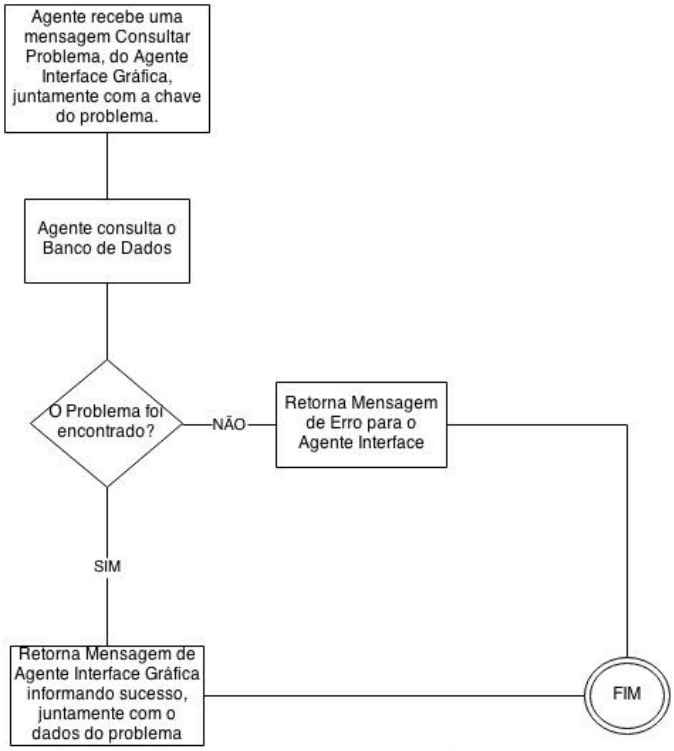
1. Mensagem do Agente Interface Gráfica
2. Chave do problema

Saída:

1. Consulta executada com sucesso, retorno das informações do problema
2. Erro: Problema não encontrado

Comportamento:

Figura 5.11: Comportamento: Recuperar informações de um problema



5.2.3.3 Comportamento 03: Recuperar Lista de Problemas cadastrados na Plataforma

Restrições: Nenhuma
Entrada:

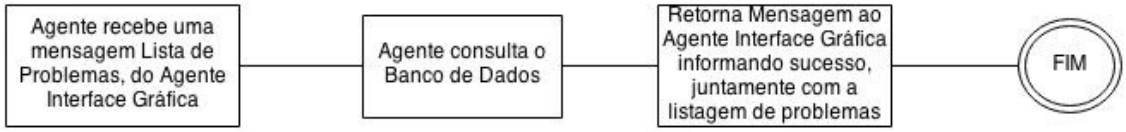
1. Mensagem do Agente Interface Gráfica
2. Categoria do Problema (sequencial, condicional, iterativo, vetor, matriz, função, recursão ou geral)

Saída:

1. Lista contendo as chaves dos problemas contidos na plataforma, na categoria solicitada.

Comportamento:

Figura 5.12: Comportamento: Recuperar Lista de Problemas cadastrados na Plataforma



5.2.3.4 Comportamento 04: Alterar dados de um Problema

Restrições:

1. Ser usuário do tipo administrador ou professor.
2. Ter executado o comportamento "Recuperar informações de um problema" para obter os dados do problema a ser alterado.
3. Não é possível alterar a chave do problema.

Entrada:

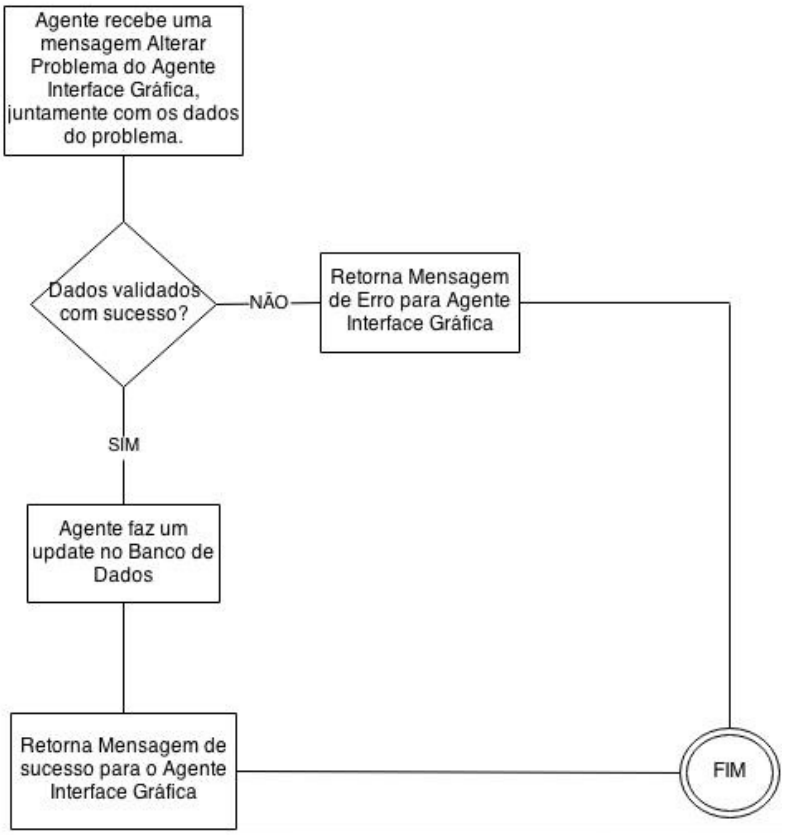
1. Mensagem do Agente Interface Gráfica
2. Dados do Problema a ser alterado

Saída:

1. Problema alterado na base de problemas
2. Erro: Falta de informações do problema

Comportamento:

Figura 5.13: Comportamento: Alterar dados de um Problema



5.2.4 AGENTE 04: Agente Gerenciador de Usuários e Turmas

Esse agente terá como função coordenar o processo de inserção, consulta e alteração de usuários na plataforma (funcionalidades nº 11 a 18 da tabela 4.1). Ele ainda incorporará as novas funcionalidades para trabalhar com turmas que serão incluídas na ferramenta: inserção e encerramento de turmas e matrícula, inclusão e remoção de alunos em uma turma.

5.2.4.1 Comportamento 01: Inserir um novo Usuário

*Restrições:*Nenhuma

Entrada:

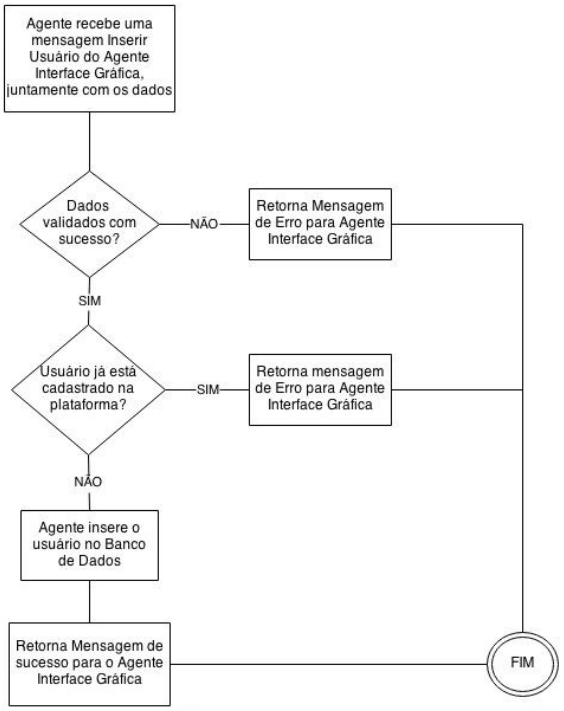
1. Mensagem do Agente Interface Gráfica
2. Informações do novo usuário

Saída:

1. Usuário cadastrado com sucesso na base de dados
2. Erro: falta de dados do usuário, ou dados inválidos
3. Erro: Usuário já cadastrado na plataforma

Comportamento:

Figura 5.14: Comportamento: Inserir um novo Usuário



5.2.4.2 Comportamento 02: Recuperar dados de um Usuário

Restrições: Ser usuário do tipo Professor, Administrador ou Aluno (desde que esteja solicitando suas próprias informações).

Entrada:

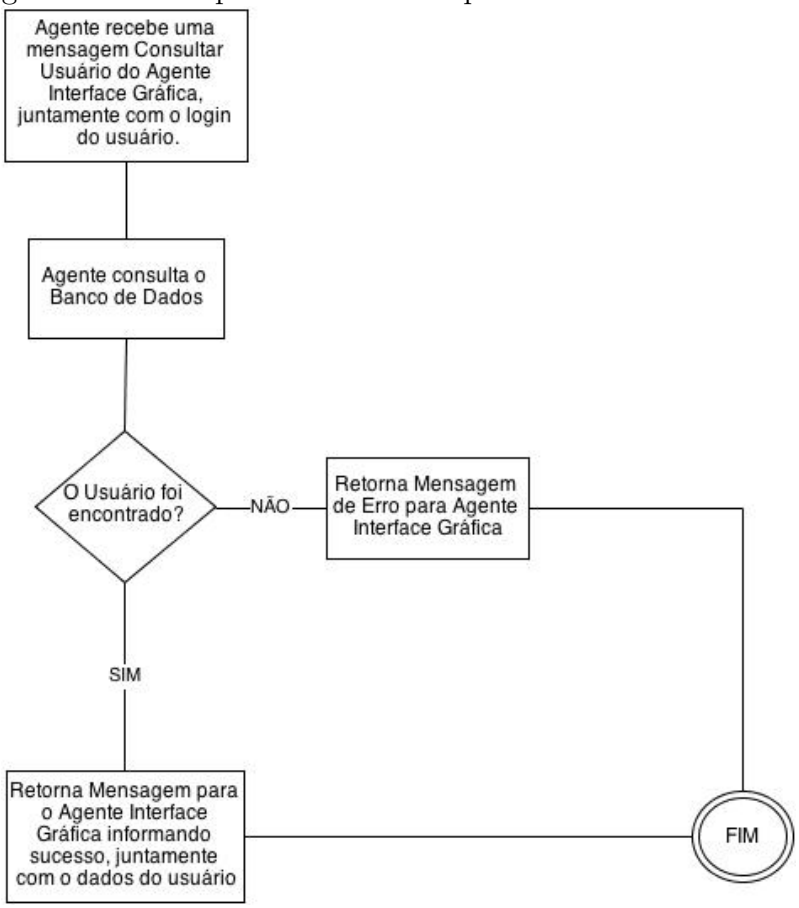
1. Mensagem do Agente Interface Gráfica
2. Login do usuário

Saída:

1. Consulta executada com sucesso, retorno das informações do usuário.
2. Erro: Usuário não encontrado na base de dados

Comportamento:

Figura 5.15: Comportamento: Recuperar dados de um Usuário



5.2.4.3 Comportamento 03: Alterar dados de um Usuário

Restrições:

1. Um usuário só pode alterar as próprias informações

2. Não é permitida a alteração do login do usuário

Entrada:

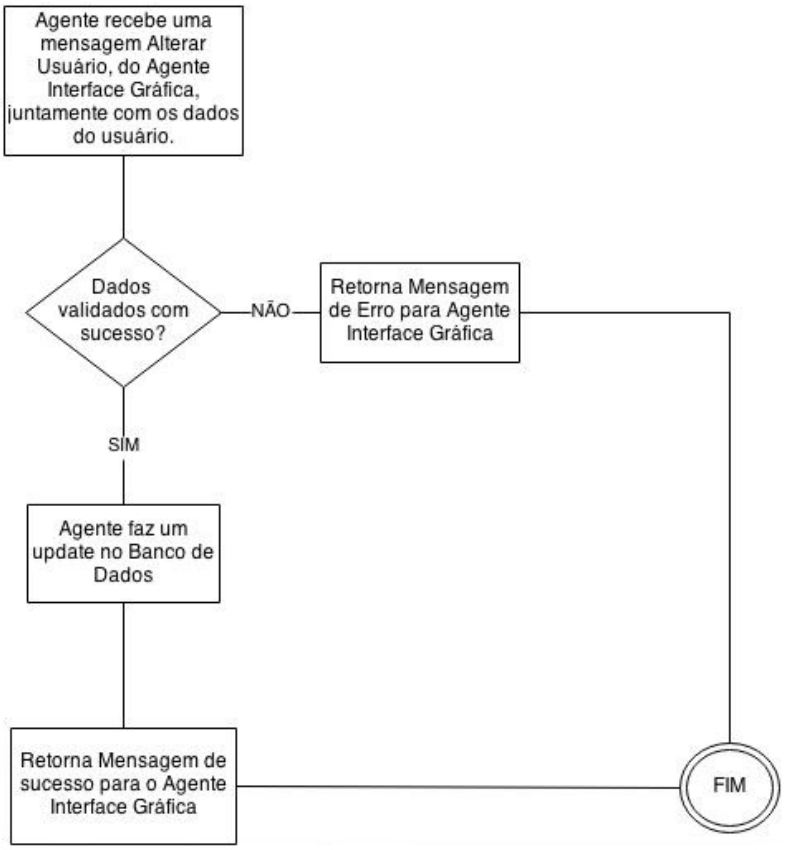
1. Mensagem do Agente Interface Gráfica
2. Login do usuário que se deseja alterar
3. Novas informações do usuário

Saída:

1. Update realizado com sucesso: informações do usuário são alteradas
2. Erro: falta de dados do usuário, ou dados inválidos

Comportamento:

Figura 5.16: Comportamento: Alterar dados de um Usuário



5.2.4.4 Comportamento 04: Recuperar lista de Usuários cadastrados na Plataforma

Restrições: O usuário deve ser do tipo Professor ou Administrador
Entrada:

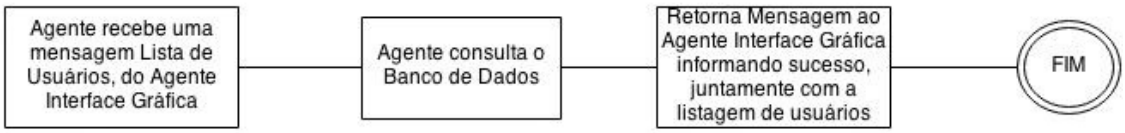
1. Mensagem do Agente Interface Gráfica

Saída:

1. Lista contendo o login de todos os usuários cadastrados na plataforma

Comportamento:

Figura 5.17: Comportamento: Recuperar lista de Usuários cadastrados na Plataforma



5.2.4.5 Comportamento 05: Envio de Senha (Esqueci Minha Senha)

Restrições: Nenhuma
Entrada:

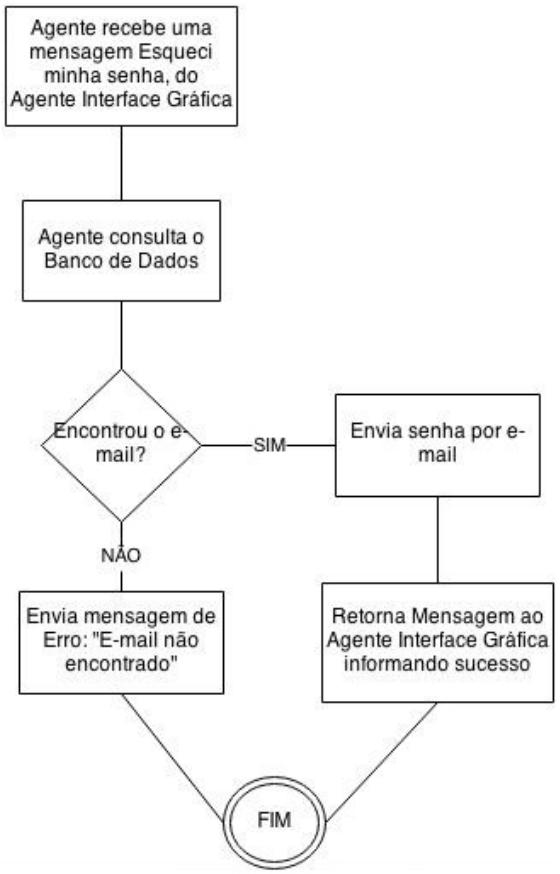
1. Mensagem do Agente Interface Gráfica
2. E-mail para o qual se deseja enviar a senha

Saída:

1. Senha enviada por e-mail com sucesso
2. Erro: E-mail inexistente na base de dados

Comportamento:

Figura 5.18: Comportamento: Envio de Senha (Esqueci Minha Senha)



5.2.4.6 Comportamento 06: Trocar perfil de Usuário

Restrições: Usuário que solicita a troca deve ser do tipo Administrador
Entrada:

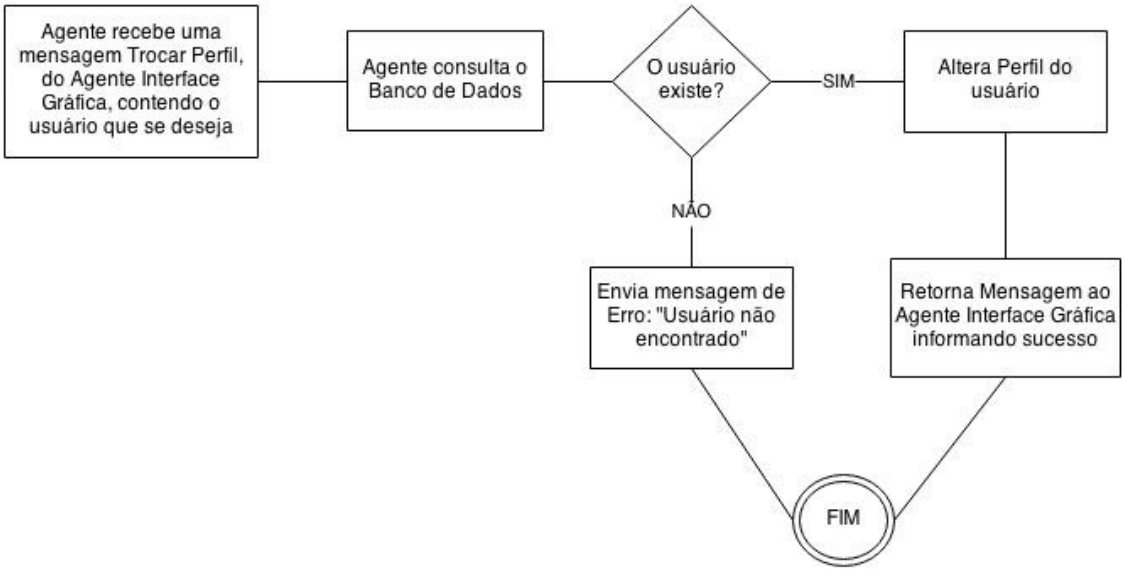
1. Mensagem do Agente Interface Gráfica
2. Chave do usuário que terá seu perfil alterado
3. Novo Perfil de usuário (Aluno, Professor ou Administrador)

Saída:

1. Alteração de Perfil realizada com sucesso
2. Erro: Chave de Usuário inválida

Comportamento:

Figura 5.19: Comportamento: Trocar perfil de Usuário



5.2.4.7 Comportamento 07: Salvar uma Solução

Restrições: Usuário deve estar logado na Plataforma
Entrada:

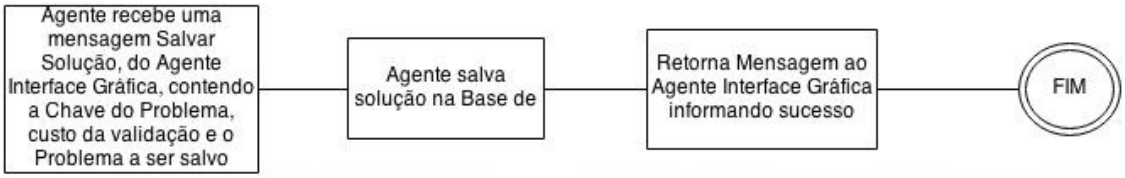
1. Mensagem do Agente Interface Gráfica
2. Chave do problema que será salvo na base de dados
3. Custo da validação do algoritmo (-1 se não tiver sido validado)
4. Problema a ser salvo

Saída:

1. Solução salva com sucesso

Comportamento:

Figura 5.20: Comportamento: Salvar uma Solução



5.2.4.8 Comportamento 08: Recuperar uma Solução salva na Base de Dados

Restrições: Nenhuma

Entrada:

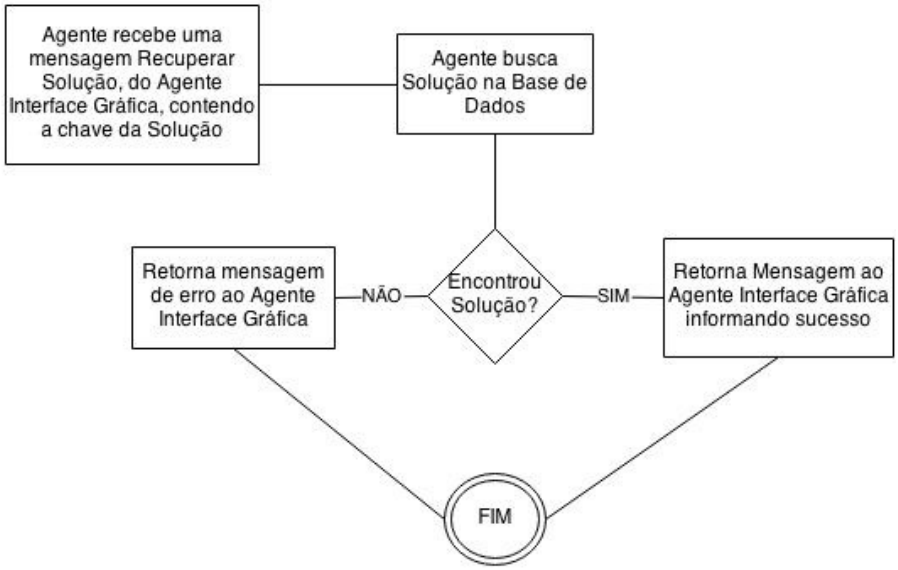
1. Mensagem do Agente Interface Gráfica
2. Chave da solução que será recuperado na base de dados

Saída:

1. Solução recuperada com sucesso
2. Erro: Solução não encontrada na base de dados

Comportamento:

Figura 5.21: Comportamento: Recuperar uma Solução salva na Base de Dados



5.2.4.9 Comportamento 09: Recuperar lista de soluções de um usuário para determinado problema

Restrições: Usuário deve estar logado na plataforma
Entrada:

1. Mensagem do Agente Interface Gráfica
2. Chave do problema cujas soluções deseja-se recuperar

Saída:

1. Lista contendo as chaves das soluções salvas para o problema informado como parâmetro

Comportamento:

Figura 5.22: Comportamento: Recuperar lista de soluções de um usuário para determinado problema

5.2.4.10 Comportamento 10: Excluir uma Solução

Restrições: Usuário deve estar logado na plataforma

Entrada:

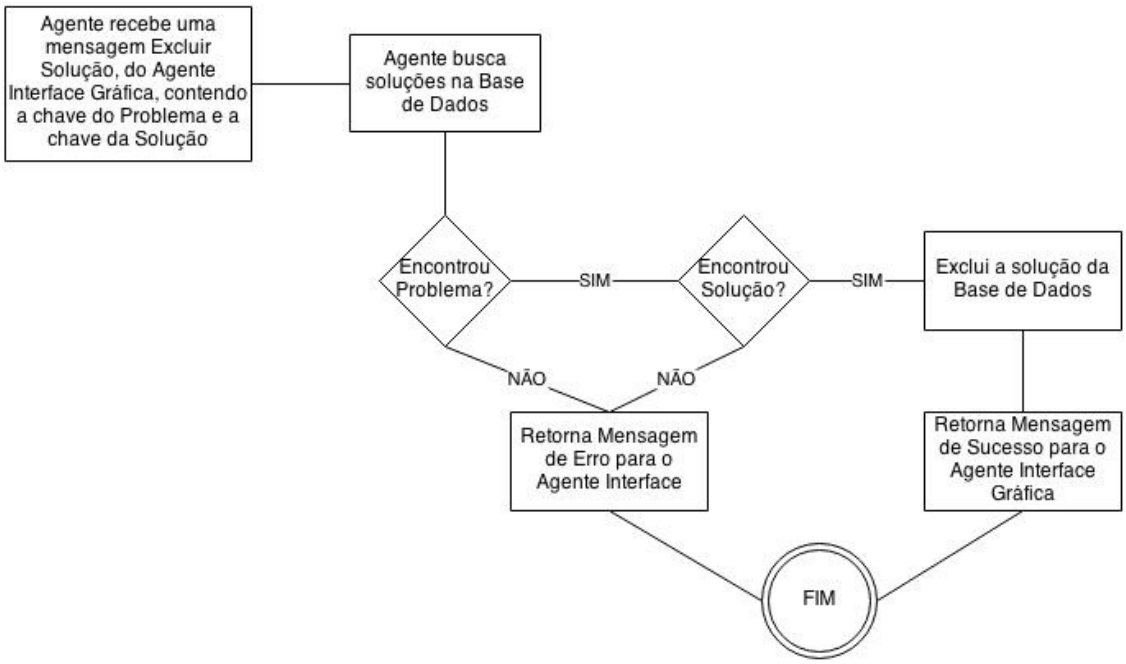
1. Mensagem do Agente Interface Gráfica
2. Chave do problema cuja solução deseja-se excluir
3. Chave da solução que deseja-se excluir

Saída:

1. Solução excluída com sucesso
2. Erro: Chave do Problema é inválida
3. Erro: Chave da Solução é inválida

Comportamento:

Figura 5.23: Comportamento: Excluir uma Solução



5.2.4.11 Comportamento 11: Criar uma Turma

Restrições: Usuário deve ser do tipo administrador
Entrada:

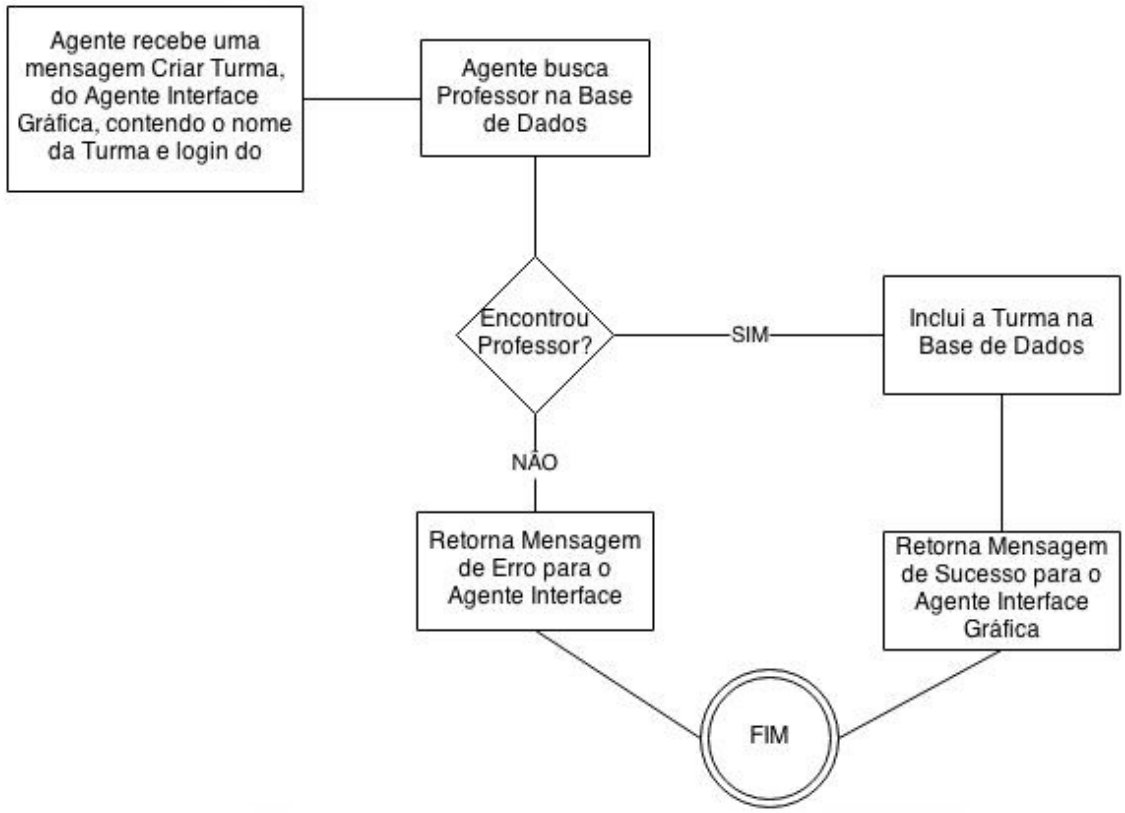
1. Mensagem do Agente Interface Gráfica
2. Nome da Turma
3. Login do professor da turma

Saída:

1. Turma criada com sucesso
2. Erro: Login do Professor é inválido

Comportamento:

Figura 5.24: Comportamento: Criar uma Turma



5.2.4.12 Comportamento 12: Finalizar uma Turma

Restrições: Usuário deve ser do tipo administrador
Entrada:

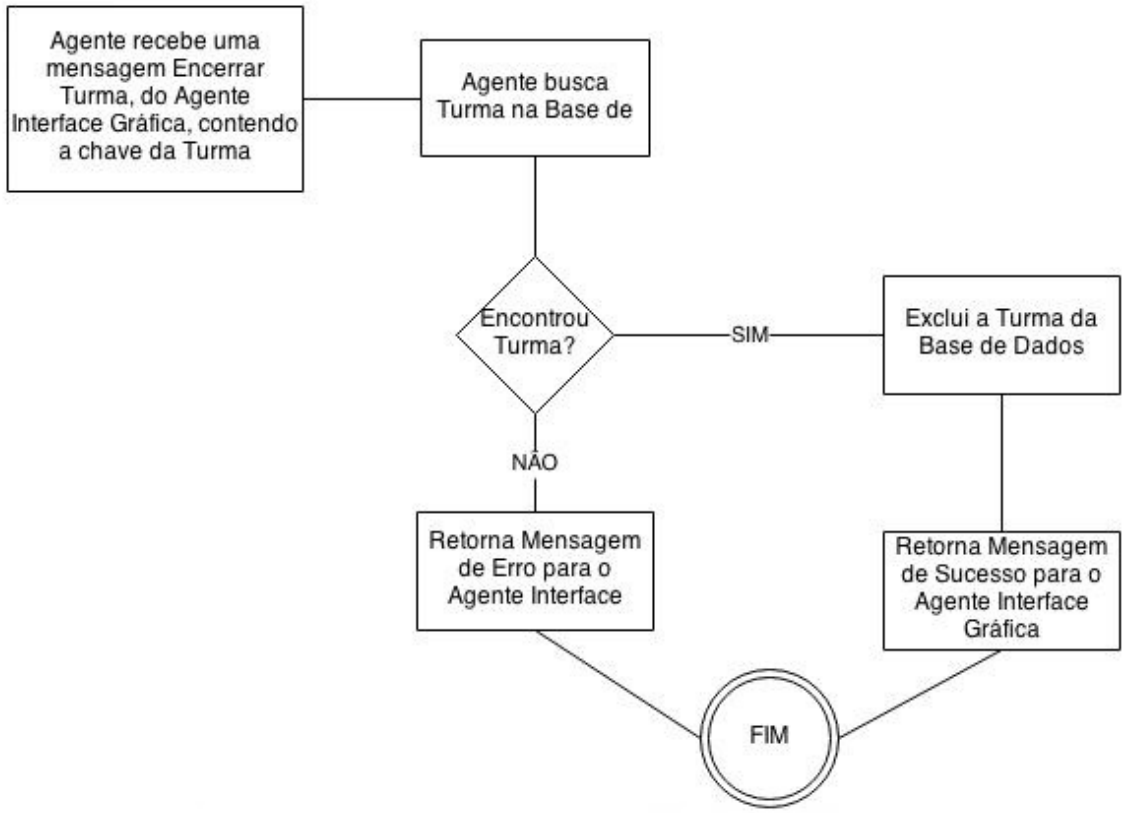
1. Mensagem do Agente Interface Gráfica
2. Chave da Turma que será finalizada

Saída:

1. Turma finalizada com sucesso
2. Erro: Chave da Turma é inválida

Comportamento:

Figura 5.25: Comportamento: Finalizar uma Turma



5.2.4.13 Comportamento 13: Alterar dados de uma Turma

Restrições: Usuário deve ser do tipo administrador ou professor
Entrada:

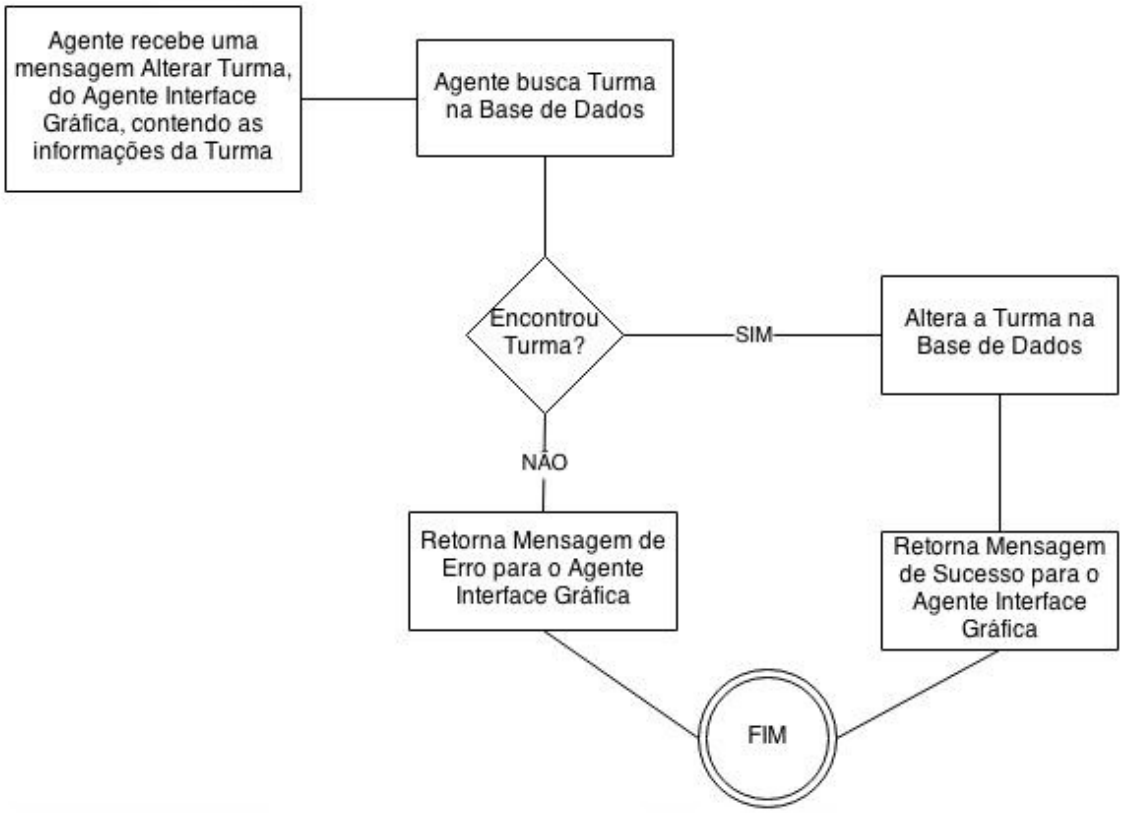
1. Mensagem do Agente Interface Gráfica
2. Dados da turma que serão alterados

Saída:

1. Turma alterada com sucesso
2. Erro: Turma não encontrada na base de dados

Comportamento:

Figura 5.26: Comportamento: Alterar dados de uma Turma



5.2.4.14 Comportamento 14: Inserir alunos em uma Turma

Restrições: O usuário deve ser do tipo professor ou administrador
Entrada:

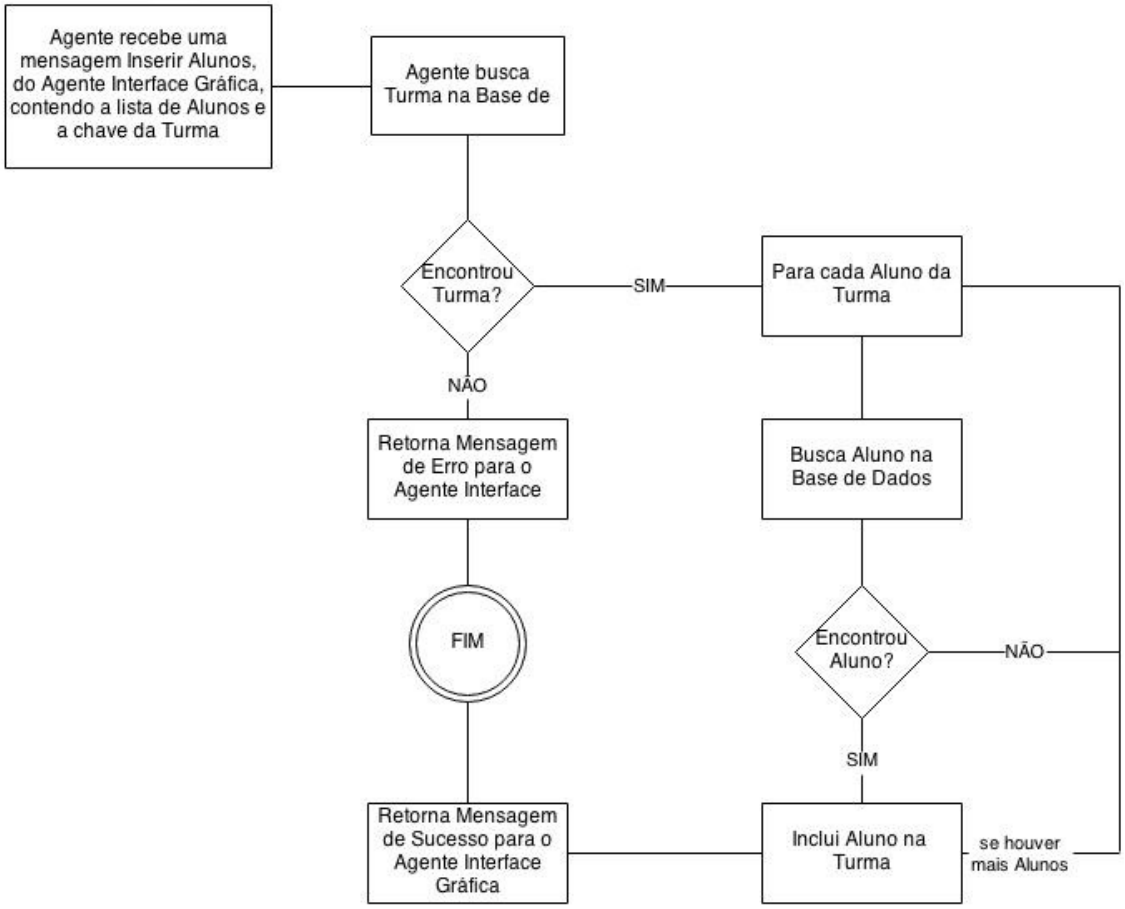
1. Mensagem do Agente Interface Gráfica
2. Lista de logins de alunos que serão inseridos na turma
3. Chave da turma em que os alunos serão matriculados

Saída:

1. Alunos alocados na turma com sucesso
2. Erro: O login de algum aluno não existe na Base de Dados
3. Erro: Turma não existente na Base de Dados

Comportamento:

Figura 5.27: Comportamento: Inserir alunos em uma Turma



5.2.4.15 Comportamento 15: Matrícula de aluno em uma turma

Restrições: O usuário deve ser do tipo aluno
Entrada:

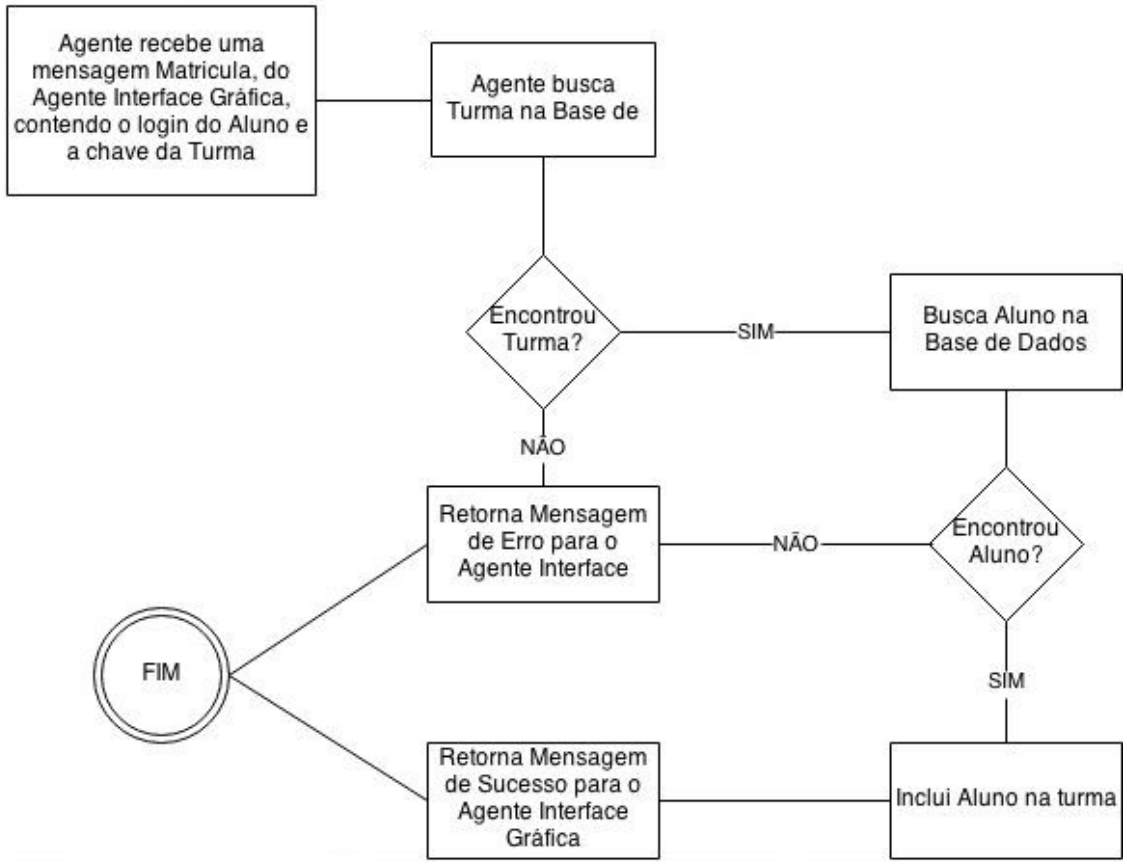
1. Mensagem do Agente Interface Gráfica
2. Login do aluno que será inserido na turma
3. Chave da turma em que o aluno será matriculado

Saída:

1. Aluno alocado na turma com sucesso
2. Erro: O login do aluno não existe na Base de Dados
3. Erro: Turma não existente na base de dados

Comportamento:

Figura 5.28: Comportamento: Matrícula de aluno em uma turma



5.2.4.16 Comportamento 16: Remover alunos de uma turma

Restrições: O usuário deve ser do tipo professor ou administrador
Entrada:

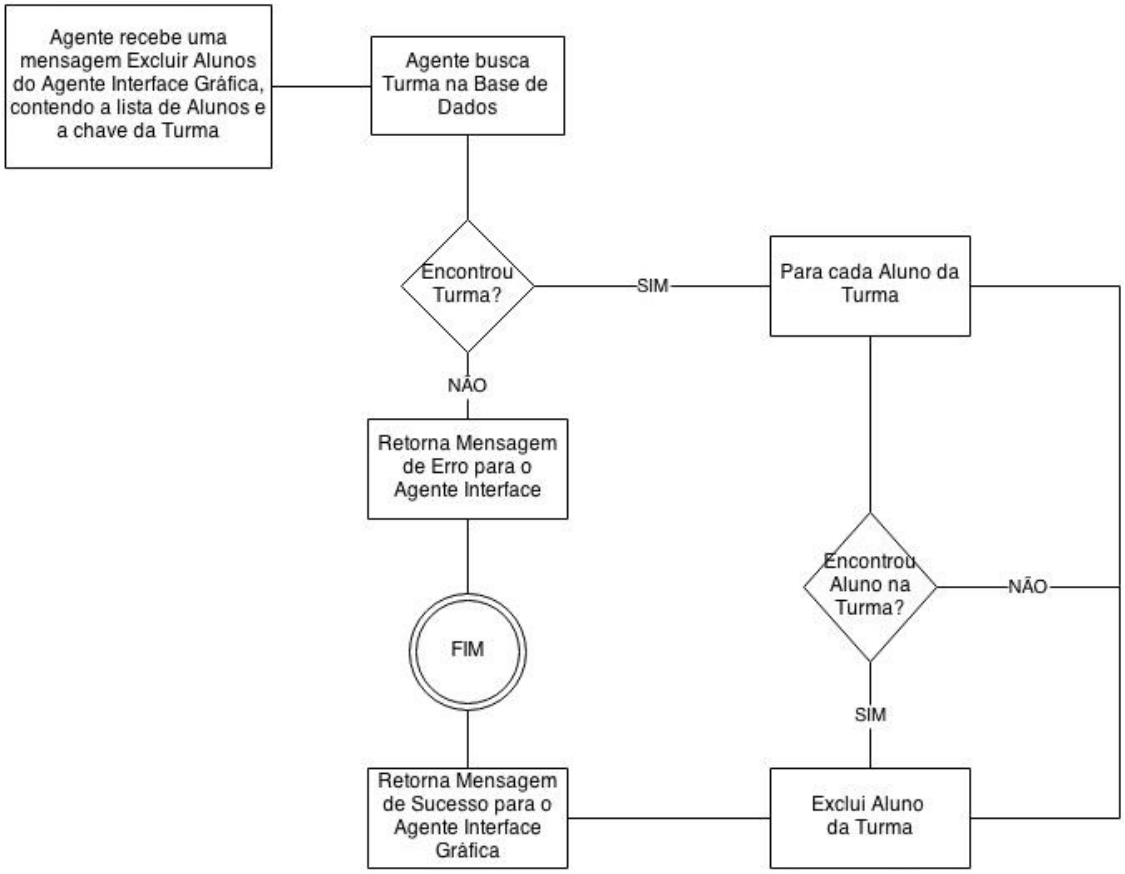
1. Mensagem do Agente Interface Gráfica
2. Lista de logins de alunos que serão removidos da turma
3. Chave da turma em que os alunos serão desmatriculados

Saída:

1. Alunos removidos da turma com sucesso
2. Erro: O login de algum aluno não estava matriculado na turma
3. Erro: Turma não existente na base de dados

Comportamento:

Figura 5.29: Comportamento: Remover alunos de uma turma



5.2.4.17 Comportamento 17: Recuperar as Turmas de um Professor

Restrições: O usuário deve ser do tipo professor ou administrador
Entrada:

1. Mensagem do Agente Interface Gráfica
2. Login do professor

Saída:

1. Lista das chaves das turmas do professor informado como parâmetro
2. Erro: O login do professor é inválido

Comportamento:

Figura 5.30: Comportamento: Recuperar as Turmas de um Professor

5.2.4.18 Comportamento 18: Recuperar os Problemas resolvidos de um aluno

Restrições: O usuário deve ser do tipo professor ou administrador

Entrada:

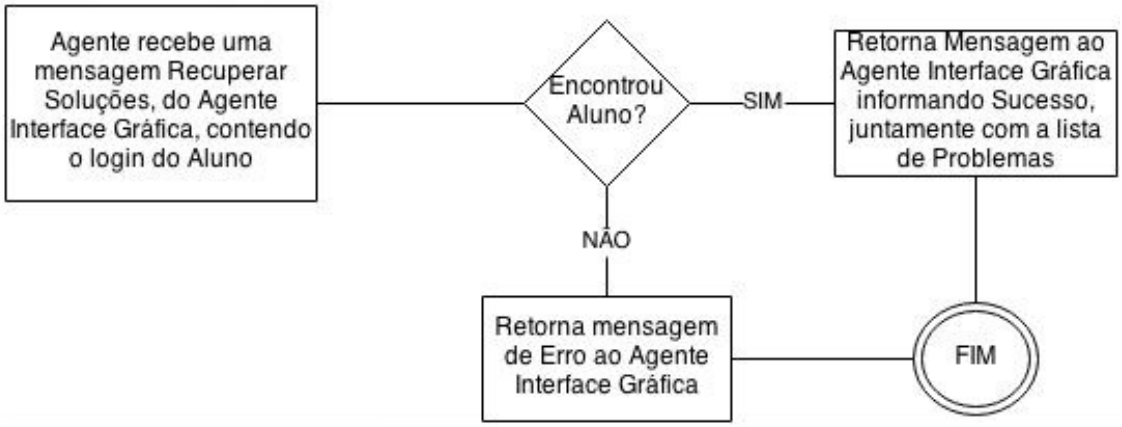
1. Mensagem do Agente Interface Gráfica
2. Login do aluno

Saída:

1. Lista das soluções criadas pelo aluno
2. Erro: O login do aluno é inválido

Comportamento:

Figura 5.31: Comportamento: Recuperar os Problemas resolvidos de um aluno



5.2.4.19 Comportamento 19: Recuperar a lista de Alunos que resolveram determinado Problema

Restrições: O usuário deve ser do tipo professor ou administrador

Entrada:

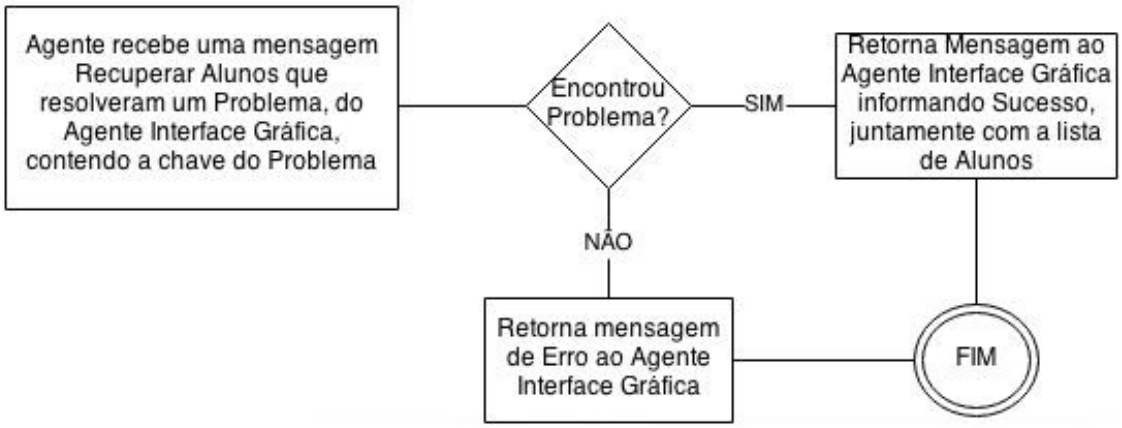
1. Mensagem do Agente Interface Gráfica
2. Chave do problema

Saída:

1. Lista de logins de alunos que resolveram o problema.
2. Erro: A chave do problema é inválida

Comportamento:

Figura 5.32: Comportamento: Recuperar a lista de Alunos que resolveram determinado Problema



5.2.4.20 Comportamento 20: Ranking de Usuários

Restrições: Nenhuma
Entrada:

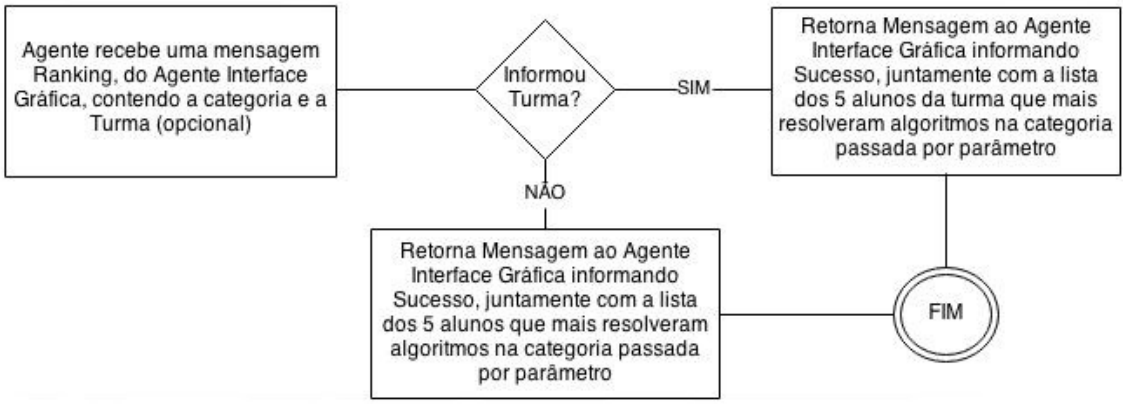
1. Mensagem do Agente Interface Gráfica
2. Categoria de Problema
3. Chave da turma (opcional)

Saída:

1. Lista contendo o login dos usuários Top 5 geral da plataforma, na categoria informada.
2. Se for informado o número da turma, é retornado o TOP 5 daquela turma em específico, na categoria informada.

Comportamento:

Figura 5.33: Comportamento: Ranking de Usuários



5.2.4.21 Comportamento 21: Quantidade de Problemas resolvidos por um usuário em determinada categoria

Restrições: Nenhuma
Entrada:

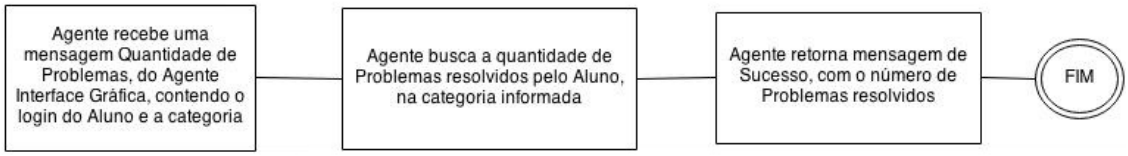
1. Mensagem do Agente Interface Gráfica
2. Login do Usuário
3. Categoria de Problema

Saída:

1. inteiro com a quantidade de problemas resolvidos na categoria informada

Comportamento:

Figura 5.34: Comportamento: Quantidade de Problemas resolvidos por um usuário em determinada categoria



5.2.4.22 Comportamento 22: Melhor solução para um Problema

Restrições: Nenhuma
Entrada:

1. Mensagem do Agente Interface Gráfica
2. Chave do Problema

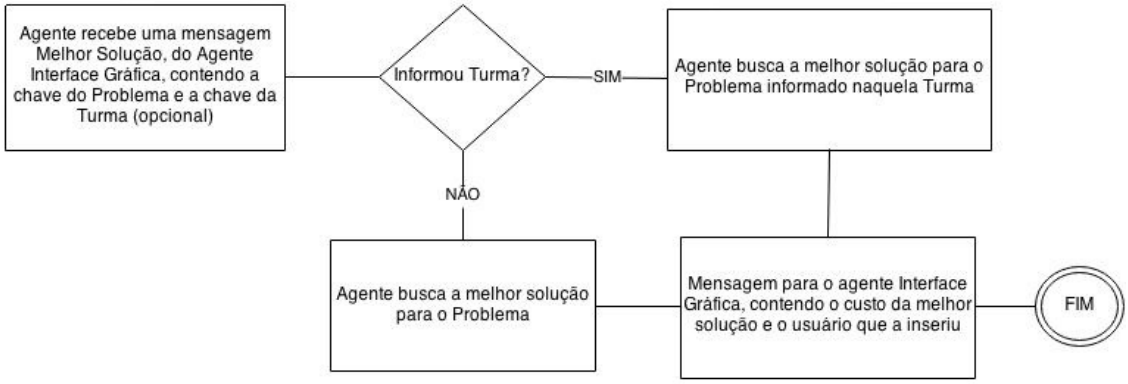
3. Chave da turma (opcional)

Saída:

1. Login do usuário e custo de validação da melhor solução daquele problema
2. Se for informado a chave da turma, é retornado o login do usuário e custo de validação da melhor solução daquele problema para a turma informada

Comportamento:

Figura 5.35: Comportamento: Melhor solução para um Problema



5.3 Decisões de Arquitetura

Esta seção apresenta algumas das decisões tomadas durante o processo de elaboração desta arquitetura, juntamente com as opções possíveis e justificativas que levaram a esta escolha.

5.3.1 Linguagem de Programação para o portal

Alternativa 1: Manter a situação atual com o portal implementado em duas linguagens: Java e Python. E integrar os agentes JADE com o código Python de alguma forma (escrita de wrapper de código ou utilizar algum framework implementado em Python que seja compatível com a tecnologia FIPA).

Alternativa 2: Reescrever o código que atualmente está em Python na linguagem Java.

Solução que será adotada: Alternativa 2, pois o código que se encontra em Python é mais simples de se implementar em Java, do que se desenvolver um método intermediário de comunicação entre o código em Python e as classes Java e *JADE*.

5.3.2 Organização dos agentes na plataforma

Alternativa 1: Os agentes rodarão no servidor como um webservice. Como consequência dessa decisão haverá inicialmente uma sociedade única de agentes que atenderá a todos os usuários do Portal de algoritmos. Essa alternativa possui alguns problemas: o primeiro diz respeito à quantidade de mensagens que precisarão ser trocadas entre os agentes e a interface gráfica através da internet, pois o sistema é altamente dependente da interação com o usuário, logo as trocas constantes de mensagens causarão lentidão no programa local e congestionamento no servidor.

O segundo problema é referente à quantidade de agentes da plataforma, pois nessa organização todos os usuários compartilharão recursos como o compilador e o agente interface gráfica que acabarão por se tornar muito requisitados e criarão um gargalo no sistema. Esse problema pode ser resolvido replicando-se o número de agentes conforme sua importância (por exemplo criar 5 agentes interface gráfica), para atender a todas as requisições. Mas essa solução recai no problema de excesso de trocas de mensagem exposto anteriormente.

Alternativa 2: Cria-se uma sociedade única de agentes para cada usuário da plataforma, que baixará tal sociedade para a sua máquina, juntamente com o applet Java no carregamento da plataforma. Essa solução evita os problemas de excessivas trocas de mensagens, mas pode acarretar num aumento do tamanho da applet afetando o carregamento da aplicação.

Solução que será adotada: Será implementada uma arquitetura híbrida entre as duas propostas. Tal solução visa evitar os problemas descritos anteriormente, pois os agentes que possuem pouco uso, ou pouca troca de mensagens rodarão no servidor, enquanto os demais agentes executarão localmente.

5.3.3 Interface Gráfica do Sistema

Conforme descrito na seção anterior, haverá a unificação das duas partes do sistema: o applet Java e os formulários escritos em Python, logo será necessária uma reestruturação da Interface Gráfica do sistema.

Essa nova interface gráfica fará a diferenciação de grupos de usuários, se mostrando diferente para alunos, professores e administradores, facilitando o uso da plataforma.

5.4 Definição do Framework que será utilizado na implementação do Portal de Algoritmos

Com base na arquitetura desenvolvida para o Portal de Algoritmos definida anteriormente, pode-se então estabelecer qual dos dois frameworks estudados no capítulo 3 é o mais adequado para o problema em questão.

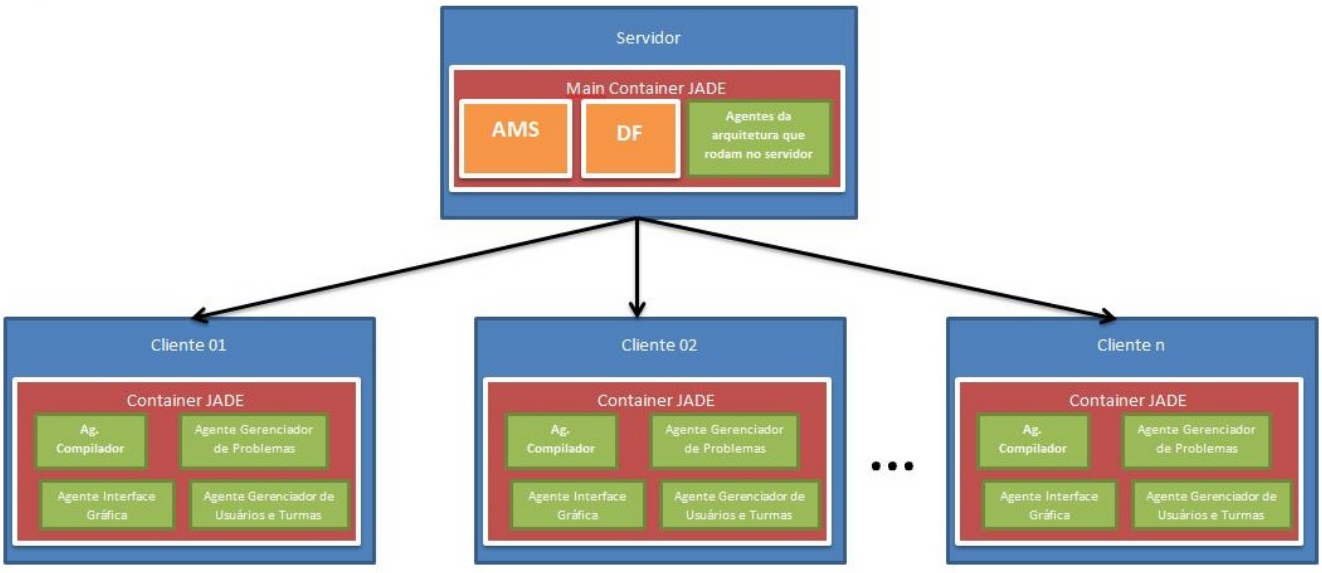
Então, optou-se pelo uso do framework *JADE* pelas seguintes razões:

- O framework *JADE* é totalmente compatível com o padrão *FIPA*, principalmente no que diz respeito à comunicação entre agentes, que é um aspecto importate no desenvolvimento de um sistema multiagente.
- *JADE* não requer o conhecimento de uma linguagem de programação adicional para que implemente agentes, basta se familiarizar com a *API* da plataforma para implementar agentes, diferentemente do *JIAC* que necessita do conhecimento na linguagem JADL para o desenvolvimento.
- *JADE* é um framework mais difundido do que *JIAC*, logo há um número maior de usuários e mais material de estudo disponível, facilitando a resolução de problemas que podem vir a surgir no decorrer da implementação. Além disso, a documentação disponível no site do framework *JIAC* está incompleta e *JADE* possui uma quantidade maior de material de ajuda disponível para os usuários do framework.

5.5 Implicações do uso do Framework *JADE* na arquitetura proposta

Definido o framework no qual a arquitetura será implementada, serão definidas as implicações que esta escolha representa à estrutura da plataforma. Sendo assim, a arquitetura fica estruturada conforme mostra a Figura 5.36.

Figura 5.36: Arquitetura final da Plataforma



A arquitetura é composta pela aplicação servidor, que conterá o container prin-

cipal do framework *JADE*, e de uma ou mais aplicações cliente, que conterão um container *JADE*, contendo os agentes do Portal de Algoritmos, descritos anteriormente.

Nota-se que foram inseridos dois novos agentes à plataforma, que rodarão no servidor, e que são criados automaticamente pelo framework no momento em que ele é inicializado: Sistema de Gerência de Agentes (Agente *AMS*) e o Facilitador de Diretórios (Agente *DF*). O primeiro deles será responsável pelo serviço de páginas brancas da plataforma, enquanto ao segundo caberá o serviço de páginas amarelas.

São criadas em cada nodo de cliente as Tabelas Locais de Descritores de Agente ($LADT$) e no nodo servidor existe a Tabela de Containers (CT) e a Tabela Global de Descritores de Agente ($GADT$). Essas tabelas facilitam a localização dos agentes dentro da plataforma, evitando comunicação excessiva entre as partes integrantes da aplicação.

Também destaca-se a possibilidade de que os agentes do Portal de Algoritmos possam rodar tanto no lado cliente, bem como no lado servidor da aplicação, como forma de distribuir o processamento pelos nodos da aplicação.

6 IMPLEMENTAÇÃO

Uma vez projetada a arquitetura que será utilizada no Portal de Algoritmos, iniciou-se o trabalho de implementação da mesma, que foi dividido nas seguintes etapas: reestruturação da Interface Gráfica do Portal, configuração do ambiente de testes em uma máquina local, implementação dos Agentes Inteligentes e adaptação da versão atual do Portal de Algoritmos. Nas próximas seções essas etapas serão detalhadas.

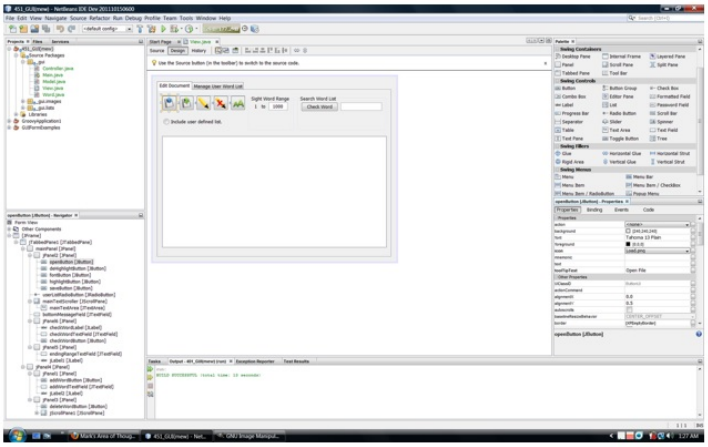
6.1 Reestruturação da Interface Gráfica do Portal de Algoritmos

O primeiro passo para a implementação do Portal de Algoritmos foi a reestruturação da interface gráfica, visto que os formulários *HTML* existentes atualmente estão escritos em Python, e deverão ser reimplementados em Java conforme explicado na seção 5.3.3. A antiga interface pode ser vista na Figura 4.1.

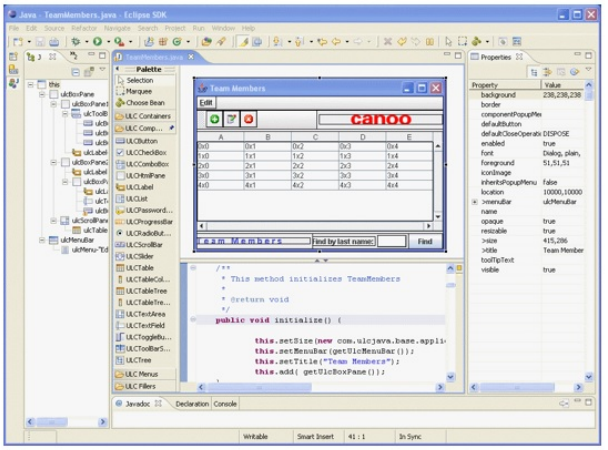
A nova interface foi concebida para ocupar todo o espaço do browser, preenchendo onde antes era ocupado pelos antigos formulários *HTML*, e também para assemelhar o Portal de Algoritmos com outras *IDE*s disponíveis atualmente e principalmente com as utilizadas nas disciplinas posteriores a algoritmos, para que os alunos se acostumem com seu uso.

Como ponto de partida para o desenvolvimento do novo layout da tela do Portal de Algoritmos fez-se um estudo comparativo entre as interfaces gráficas de quatro *IDE*s utilizadas nas disciplinas de programação das disciplinas do Centro de Computação e Tecnologia da Informação da *UCS*: Netbeans e Eclipse, utilizadas no ensino de Java e também Dev-C++ e Code::Blocks, utilizadas nas disciplinas de C. A Figura 6.1 contém uma captura de tela de tais *IDE*s.

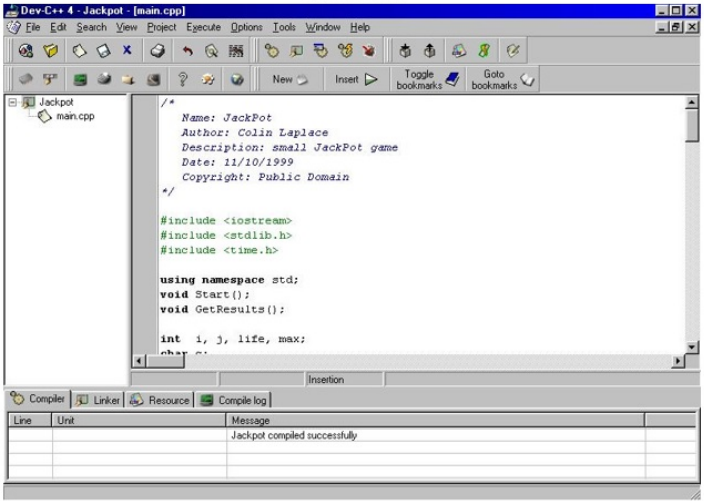
Figura 6.1: Layout Interface Gráfica das *IDE*s utilizadas pelos acadêmicos nas disciplinas de programação



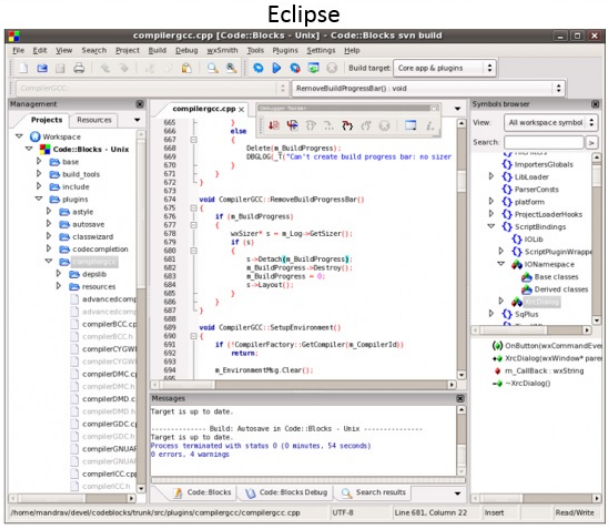
NetBeans



Eclipse



Dev-C++



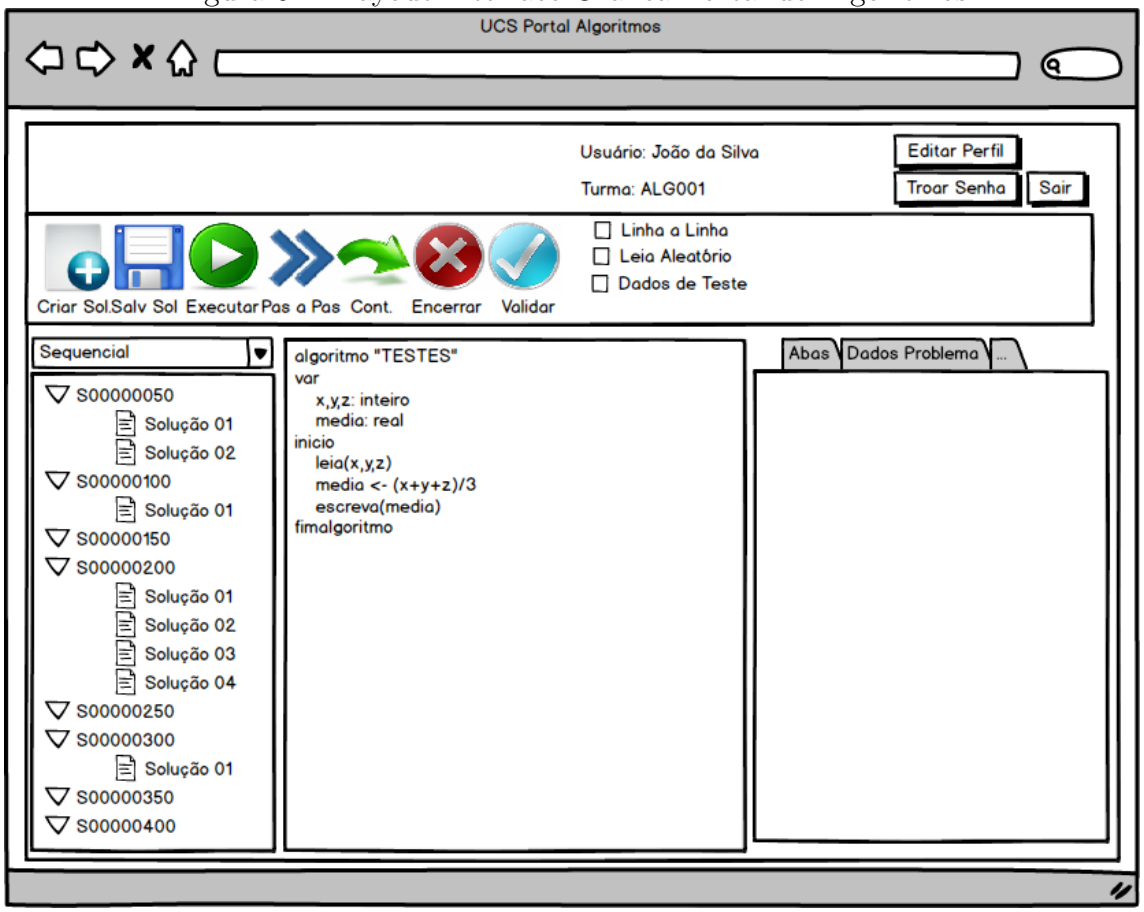
Code::Blocks

Os principais parâmetros observados nas interfaces gráficas das *IDE*s mostradas na Figura 6.1 e utilizados para desenvolver a nova interface do Portal de Algoritmos foram os seguintes:

- Utilização de menus e barras de ferramentas;
- Os projetos são mostrados em forma de árvore, que inclui diversos pacotes e arquivos, sempre ao lado esquerdo do editor de código;
- O editor de código localiza-se sempre no centro da tela;
- A parte de baixo da tela é utilizada para mostrar a execução dos programas
- O lado direito do editor de código expõe propriedades dos arquivos que estão sendo editados.

Com base nessa análise foi proposta a nova Interface Gráfica para o Portal de Algoritmos, que pode ser vista na Figura 6.2.

Figura 6.2: Layout Interface Gráfica Portal de Algoritmos



No topo da tela encontram-se a área de login, e de gerenciamento de perfil de usuário (quando o usuário efetuar login no sistema). Logo abaixo há uma barra de ferramentas, que contém as opções de execução e criação de soluções.

Ao centro da tela encontra-se o editor de código fonte e ao seu lado esquerdo encontram-se os problemas disponíveis na plataforma, subdivididos nas respectivas categorias, bem como as soluções criadas por aquele usuário. Ao lado direito do editor de código estão os dados do problema selecionado e as abas do editor de código e execução.

Apesar de não ter sido implementado, foram previstos três tipos de usuários para o sistema: aluno, professor e administrador. Cada usuário teria uma interface gráfica própria, mas as três interfaces manterão o padrão descrito acima. Por exemplo: a interface de professor conterá uma árvore ao lado esquerdo da tela e esta mostrará uma relação de turmas daquele professor, e os alunos de cada turma. Já na interface de administração, haverá uma listagem de todos os usuários da plataforma, enquanto na interface do aluno há uma listagem de problemas com as respectivas soluções criadas pelo usuário.

6.2 Configuração do ambiente de testes em uma máquina local

Para a instalação do ambiente de testes local, foi disponibilizado pela GTIC da Universidade de Caxias do Sul uma cópia do servidor do portal de algoritmos. Tal cópia consistia da réplica do disco rígido contendo a instalação da biblioteca Python, configurada para rodar o Portal de Algoritmos e uma cópia da base de dados MySQL atualizada. Tal configuração foi feita em ambiente Linux (Ubuntu Server 9.04 32 bits).

Para sua utilização em uma máquina local seria necessário utilizar um software de máquina virtual, para emular o sistema operacional na máquina onde a nova versão foi construida (Windows 8 6.2 64 bits). Para tanto, foram utilizados os softwares: Oracle VirtualBox e VMware Player.

Após a configuração da máquina virtual contendo o servidor da aplicação, buscou-se uma maneira de realizar o acesso ao Portal de Algoritmos através do browser da máquina local, e a troca de arquivos entre as duas máquinas através de um software de FTP.

Foram encontrados problemas na execução dessas atividades, não sendo possível acessar o servidor do Portal de Algoritmos, tampouco trocar arquivos entre os dois sistemas operacionais, apesar das diversas tentativas para sanar tal problema.

Dentre as hipóteses levantadas na tentativa de explicar tal fato, considerou-se a existência de algum problema nas configurações de rede do servidor ou também problemas com o firewall de ambos os sistemas operacionais. Tais indícios foram levantados após uma investigação em fóruns e tutoriais na internet, onde buscou-se usuários que enfrentaram tal problema. Mas mesmo seguindo os passos para solução apresentados, nenhum sanou os problemas de comunicação enfrentados.

Devido ao pouco tempo disponível para solucionar tais problemas, optou-se pela simulação do ambiente do servidor, com o armazenamento de alguns problemas, soluções e usuários em estruturas de dados na memória principal da máquina onde o servidor rodará.

6.3 Implementação dos Agentes Inteligentes

A implementação do Portal de Algoritmos Multiagente foi feita utilizando-se a linguagem de programação Java em sua versão 1.7 (conforme descreve a seção 5.3.1), o framework *JADE* versão 4.3 (veja seção 5.4) e a *IDE* escolhida para o desenvolvimento foi o NetBeans 7.4.

Inicialmente procedeu-se a instalação do framework *JADE* na máquina utilizada para desenvolver a aplicação, descompactando o arquivo obtido em `http://jade.`

tilab.com/ em um diretório. Após isso foi criado um novo projeto na ferramenta NetBeans e foi realizada a importação da biblioteca do *JADE* para o projeto.

Após esse procedimento foram criados quatro pacotes no projeto: `br.ucs.compilador`, `br.ucs.interfaceGrafica`, `br.ucs.usuarios`, `br.ucs.problemas`, cada qual responsável por armazenar as classes de seus respectivos agentes. Ainda, cada pacote possui um subpacote chamado comportamentos, onde residirão as classes que irão conter a implementação dos comportamentos de cada agente.

Depois de configurado o ambiente de desenvolvimento, criaram-se as classes referentes aos quatro agentes da plataforma. Abaixo pode ser vista a implementação de um desses agentes:

Trecho de Código 6.1: Classe AgenteCompilador.java

```
1  package br.ucs.compilador;
2
3  import jade.domain.*;
4  import jade.lang.acl.*;
5  import jade.core.Agent;
6  import br.ucs.compilador.comportamentos.*;
7  import jade.core.behaviours.CyclicBehaviour;
8  import br.ucs.interfaceGrafica.InterfaceIntermediaria;
9
10 public class AgenteCompilador extends Agent {
11
12     public static final int REQUISICAO_DADOS = 0;
13     public static final int ERRO = 1;
14     public static final int MOSTRAR_MENSAGEM = 2;
15     public static final int SUCESSO = 3;
16
17     MessageTemplate MT1;
18
19     protected void setup() {
20         MT1 = MessageTemplate.MatchLanguage("INICIAR COMPORTAMENTO") ;
21         // Criamos uma entrada no DF
22         DFAgentDescription df = new DFAgentDescription();
23         df.setName(getAID());
24         //Criando o serviço
25         ServiceDescription sd = new ServiceDescription();
26         sd.setType("COMPILADOR"); // Tipo do Serviço
27         sd.setName("COMPILADOR"); //Nome do Serviço
28         //adiciona o serviço no na descritor
29         df.addServices(sd);
30         //registrando agente no DF
31         try {
32             //método registrer(agente que oferece, descrição)
33             DFService.register(this, df);
34         } catch (FIPAException e) {
35             e.printStackTrace();
36         }
37         this.addBehaviour(new CyclicBehaviour() {
38             @Override
39             public void action() {
40                 String codigo = "";
41                 InterfaceIntermediaria tc = null;
42                 ACLMessage mensagem = myAgent.receive(MT1);
```

```
43                  if(mensagem != null){
44                      if(mensagem.getLanguage().equals("INICIAR COMPORTAMENTO")){
45                          String comportamento = (String) mensagem.
                                getConversationId();
46                          if(comportamento.equals("PASSO A PASSO")){
47                              try {
48                                  codigo = mensagem.getProtocol();
49                                  tc = (InterfaceIntermediaria) mensagem.
                                        getContentObject();
50                              } catch (UnreadableException ex) {
51                                  System.err.println(ex.getMessage());
52                              }
53                              myAgent.addBehaviour(new CMPPassoAPasso(tc, codigo))
                                    ;
54                              this.block();
55                          }
56                          if(comportamento.equals("EXECUCAO NORMAL")){
57                              addBehaviour(new CMPExecucaoNormal(mensagem));
58                          }
59                      }
60
61                  }
62                  this.block();
63              }
64          });
65      }
66  }
```

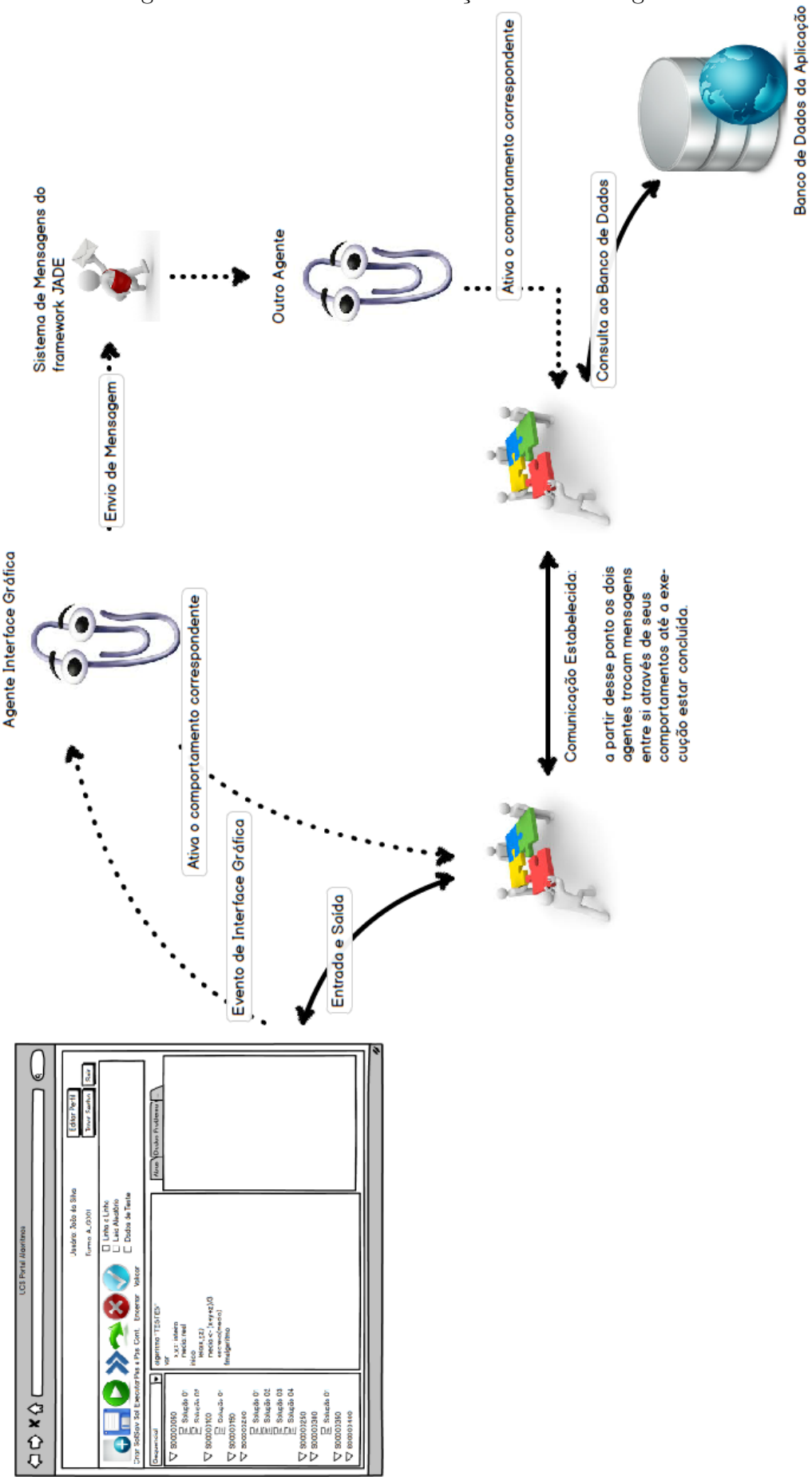
A implementação desse agente se deu estendendo a classe `jade.core.Agent`, que obriga o programador a implementar o método `setup()`, onde as configurações iniciais do agente são feitas e onde a lógica do agente foi implementada.

O modelo exposto acima foi utilizado na implementação dos demais agentes, pois estes funcionam da mesma maneira que o Agente Compilador: eles aguardam por uma mensagem, que irá ativar um dos comportamentos do agente. O único agente com um comportamento um pouco diferente é o Agente Interface Gráfica: ele aguarda por uma entrada do usuário (evento de interface gráfica) ou uma saída (mensagem de outro agente).

A implementação dos comportamentos de cada agente foi feita criando uma classe que estende `jade.core.behaviours` ou uma de suas subclasses. É necessário que se implementem os métodos: `void action()` e `boolean done()`, conforme explicado na seção 3.1.1.1.

A Figura 6.3 representa um esquema da comunicação de dois agentes dentro do programa.

Figura 6.3: Fluxo de Comunicação entre dois agentes



Dessa maneira a comunicação entre dois agentes para que eles possam executar determinada funcionalidade do sistema precisa seguir os seguintes passos:

- Inicialmente, o usuário do sistema, ao clicar em algum elemento da Interface Gráfica dispara um evento. Nesse momento um objeto da classe `jade.gui.GuiEvent` é criado e a Interface Gráfica dispara ao Agente Interface Gráfica um evento.
- Por sua vez, o Agente Interface Gráfica, ao identificar a ocorrência de um evento de Interface Gráfica, avalia qual é o comportamento que deve ser acionado, disparando seu comportamento correspondente e acionando o agente responsável pela tarefa.
- O outro agente, que pode pertencer ao sistema, ou estar alheio a este, ao ser acionado pelo Sistema de Mensagens do framework *JADE* também ativará o comportamento correspondente àquela funcionalidade.
- Quando os comportamentos dos dois agentes estiverem em funcionamento, eles trocarão mensagens entre si para executar determinada funcionalidade.
- Além disso, os agentes consultam informações contidas no banco de dados da aplicação e também realizam operações de entrada e saída com a interface gráfica da aplicação.

6.3.1 Protocolo de Comunicação entre os agentes

Após a definição do comportamento global do sistema, foi pensado em um protocolo para as trocas de mensagens entre os agentes do sistema. O framework *JADE* prevê a utilização de objetos da classe `jade.lang.acl.ACLMessage` para a comunicação entre agentes, logo esta classe serviu como base para a elaboração deste protocolo.

Dos campos presentes nessa classe, os seguintes foram utilizados no protocolo de comunicação do Portal de Algoritmos:

- **int performative**: utilizado para mostrar o objetivo ou o estado da mensagem. Por exemplo, se o resultado da execução de um algoritmo resultou em sucesso ou falha.
- `jade.core.AID` **receiver**: campo que recebe o identificador do agente destinatário da mensagem.
- `jade.core.AID` **sender**: armazena o *AID* do agente que está enviando a mensagem.
- **String conversation_id**: neste campo é armazenada uma string que representa uma operação que deve ser executada pelo agente que recebeu tal mensagem.
- **Object contentObject**: neste campo são armazenadas as informações que

serão trocadas entre os agentes. Quando houver mais de um objeto que deve ser armazenado neste campo, utiliza-se uma instância da classe ArrayList para armazena-los. As classes de todos os objetos que forem enviados entre dois agentes devem implementar a interface `java.io.Serializable`.

- **String content**: quando o conteúdo das mensagens for uma String este campo é utilizado para o armazenamento, pois ele elimina a necessidade de se fazer *casting* de operadores

6.4 Adaptação da versão atual do Portal de Algoritmos

O código fonte do Portal de Algoritmos existente é composto pelas classes escritas na linguagem Java, que contém o compilador e o applet com parte da interface gráfica do portal, e também pelo código escrito em Python para acesso ao banco de dados da aplicação. Conforme descrito anteriormente neste trabalho somente o código escrito em Java foi considerado, devendo o código em Python ser reescrito.

Analisando o código fonte obtido notou-se um forte acoplamento entre as classes do sistema, sendo que o compilador depende fortemente da interface gráfica do sistema, como por exemplo cita-se o método `adicionaAviso(String texto)` pertencente à classe de interface gráfica, que é responsável por escrever o texto que lhe é passado por parâmetro em uma área de texto, promovendo assim a saída da execução de um algoritmo. Esse método é chamado 169 vezes na classe `aSintatico`, que é responsável por realizar a Análise Sintática no algoritmo.

Diante das inevitáveis transformações que tiveram que ser operadas na interface gráfica da aplicação, pode-se perceber uma certa dificuldade em lidar com essa propriedade do sistema, pois uma simples alteração em um componente da interface gráfica podia desencadear diversos erros em diversas partes do código.

Outro agravante a essa situação é em relação ao uso de agentes *JADE* no portal de algoritmos, pois como a interface gráfica e o compilador são tarefas pertencentes a agentes distintos, haveria a necessidade de envio de componentes de interface gráfica via mensagens. Tal ação é impraticável, visto que o envio de objetos via streaming de dados faz necessário a implementação da interface `java.io.Serializable` pelos componentes da interface gráfica, o que não ocorre com os objetos pertencentes ao pacote `javax.swing`.

A solução adotada neste caso foi criar uma classe intermediária, que contém o valor dos campos que seriam utilizados pelo compilador, não sendo necessário, portanto, transitar componentes de interface gráfica entre os agentes da plataforma.

Então, quando ocorrer uma comunicação entre o agente interface gráfica e algum outro agente da plataforma, será necessário que a interface gráfica crie um objeto do tipo `br.ucs.interfaceGrafica.InterfaceIntermediaria` e preencha os campos

dessa classe correspondentes aos componentes que o agente compilador utilizará na execução do algoritmo. No momento da inserção de novos agentes à plataforma, será necessário alterar esta classe (ou criar uma nova classe) para atender aos novos requisitos de troca de informação.

Quando o agente compilador desejar alterar algum componente da interface gráfica ele deve se utilizar de uma mensagem, não sendo necessário que ele utilize de objetos intermediários nesse processo, bastando ele utilizar o protocolo descrito acima.

6.5 Estudo de Caso das funcionalidades implementadas

Nesta seção serão apresentadas as funcionalidades que foram desenvolvidas nesse trabalho, com uma descrição de seu uso e de seu funcionamento. Será feita também uma análise de desempenho de tais comportamentos, pelas trocas de mensagens entre os agentes, através da ferramenta sniffer disponibilizada juntamente com o framework *JADE*.

6.5.1 Comportamento Execução Normal

Para se ativar este comportamento, é necessário que o usuário do sistema (estando logado no portal ou não) digite um algoritmo no editor de código e após isso clique no botão Executar. Então, o agente Interface Gráfica solicitará a execução do algoritmo, cujo resultado pode ser a execução do algoritmo corretamente, a ocorrência de algum erro de compilação (léxico ou sintático) ou de um erro de execução.

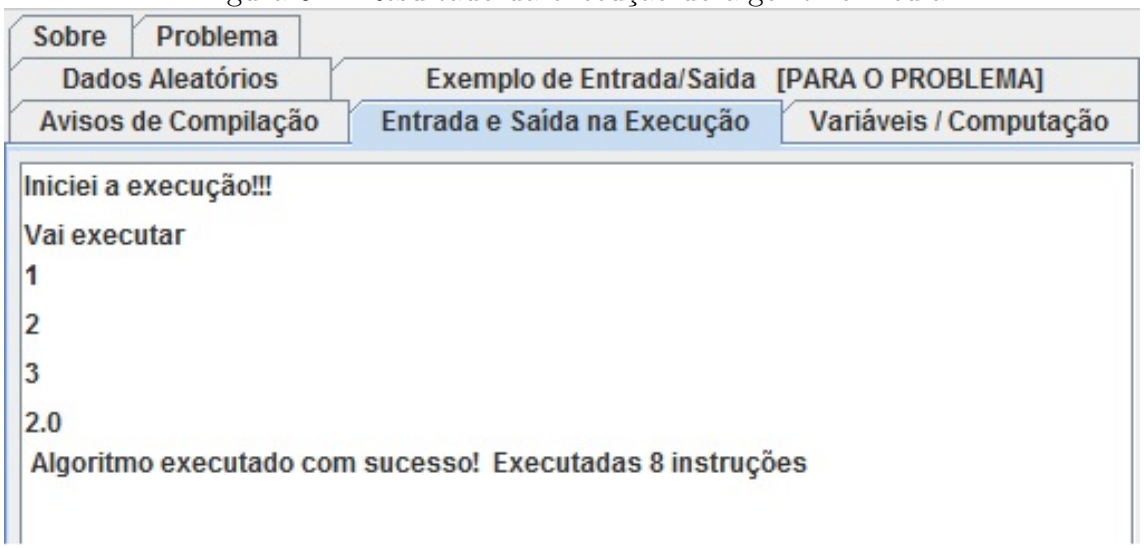
A execução dos testes foi feita utilizando o algoritmo sequencial de código S00000150 presente no Portal de Algoritmos, que calcula a média de três números informados pelo usuário e mostra a sua média aritmética, conforme mostrado abaixo:

Trecho de Código 6.2: Algoritmo que calcula a média aritmética

```
1 algoritmo "MEDIA"
2 var
3     x, y, z: inteiro
4     media: real
5 inicio
6     leia(x, y, z)
7     media <- (x+y+z)/3
8     escreva(media)
9 fimalgoritmo
```

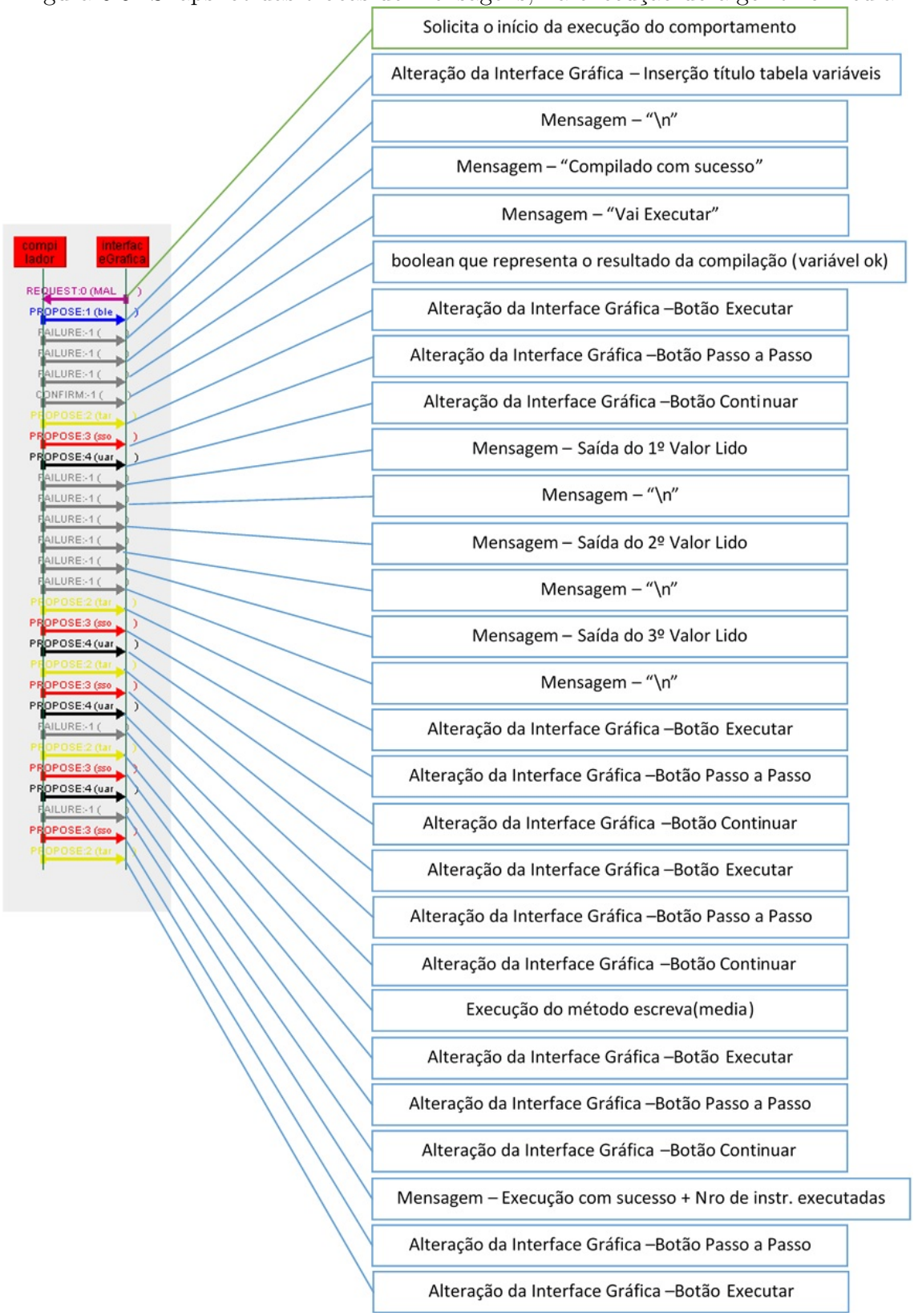
Como resultado de sua execução para as entradas x = 1; y = 2 e z = 3, obteve-se a seguinte resultado na saída padrão do Portal de Algoritmos:

Figura 6.4: Resultado da execução do algoritmo média



A ferramenta Sniffer do framework *JADE* acusou a troca de 28 mensagens entre os agentes (1 mensagem do agente Interface Grafica para o Agente Compilador e 27 mensagens do Agente Compilador para o Agente Interface Gráfica), conforme mostra a Figura 6.5.

Figura 6.5: Snapshot das trocas de mensagens, na execução do algoritmo média



Para avaliar melhor a execução deste algoritmo, considerando como parâmetro as trocas de mensagens efetuadas, foram executadas algumas instâncias de alguns

algoritmos, para que seja possível comparar os resultados obtidos. A tabela 6.1 contém esses valores.

Tabela 6.1: Troca de Mensagens na execução de algoritmos

Problema	**Categoria**	**Particularidade**	**Qtde. Mensagens**	**Qtde. Instruções**
S00000150 - Média Aritmética	Sequencial	Pouca Entrada Pouca Saída Pouco Processamento	28	8
R00000950 - Divisão Inteira Recursiva	Recursivo	Pouca Entrada Pouca Saída Muito Processamento	35	30
I00006700 - Ler 100 nros e escrever 2 últimos primos	Iterativo	Muita Entrada Muita Saída Muito Processamento	9331	8081
I00003800 - Data Juliana	Iterativo	Pouca Entrada Pouca Saída Muito Processamento	91	41
V00007800 - A sequencia "Look & Say", con n = 23	Vetores	Pouca Entrada Saída e Processamento aumentam conforme o valor da entrada aumenta	98.812	63.962
I00001950 - Escrever 50 primeiros primos	Iterativo	Nenhuma Entrada Muita Saída Muito Processamento	64	50

Observando os resultados obtidos, nota-se que o número de mensagens tende a aumentar quando as incidências de entradas e saídas do sistema aumentam, pois estas geram uma troca direta de mensagens entre os agentes Compilador e Interface Gráfica. O mesmo ocorre quando a necessidade de processamento do algoritmo tende a ser maior, pois as mudanças que o agente Compilador provoca no agente

Interface Gráfica tornam-se mais frequentes.

Em relação ao desempenho do sistema, não foi constatada nenhuma mudança no desempenho do novo Portal de Algoritmos em relação à sua versão anterior, considerando-se a execução do sistema com os agentes localizados na máquina do cliente.

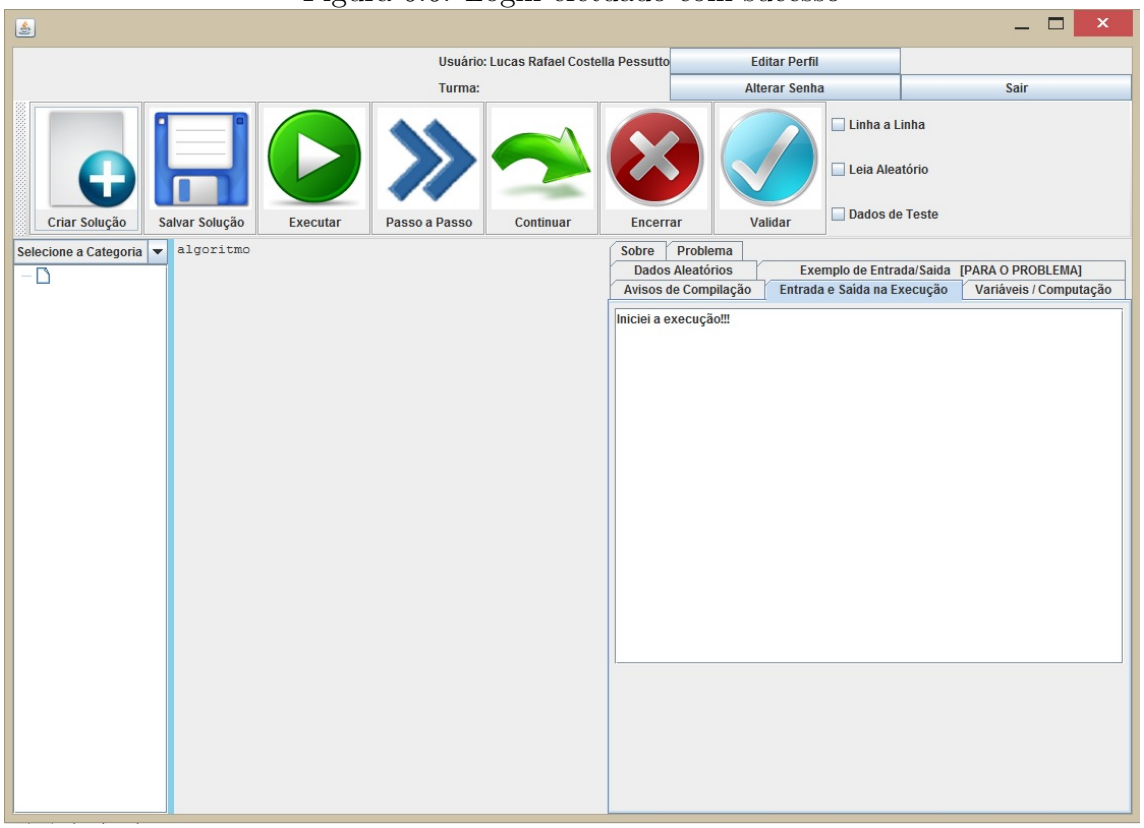
6.5.2 Comportamento Recuperar dados de um usuário

Este comportamento, que é ativado quando o usuário do sistema faz login no Portal de Algoritmos, tem por função validar os dados informados pelo usuário, verificando se ele consta na base de dados da aplicação, e também de retornar informações do usuário para o Agente Interface Gráfica.

A execução de tal comportamento pode ocasionar duas situações: o usuário conseguirá logar no sistema, informando um usuário e senha válidos, ou ele não conseguirá, ou seja, o usuário ou a senha informados estão incorretos.

No caso de um login efetuado com sucesso, são apresentados os dados do usuário e algumas funcionalidades de alteração do perfil, conforme mostra a figura abaixo.

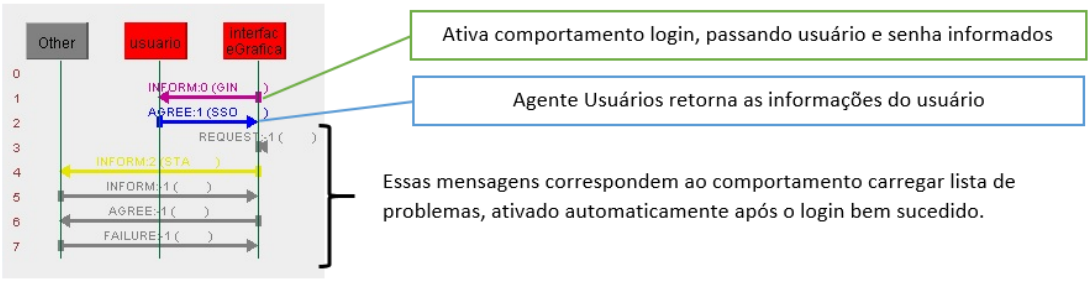
Figura 6.6: Login efetuado com sucesso



Analisando-se as trocas de mensagens entre os agentes Interface Gráfica e Gerenciador de Usuários (Figura 6.7), percebe-se que são trocadas apenas 2 mensagens

entre os agentes, uma que solicita dos dados do usuário e informa o login e a senha, e outra contendo a resposta do agente.

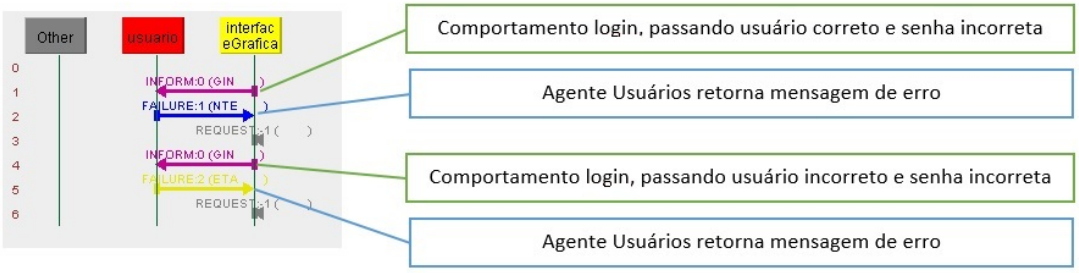
Figura 6.7: Trocas de Mensagens no comportamento Recuperar Dados Usuário



Em caso de falha na tentativa de login, duas mensagens podem ser exibidas: "Login inexistente na base de dados"ou "Senha Incorreta", a primeira é mostrada sempre que o login informado não existir na base de dados da aplicação e a segunda é exibida se o login informado existir na base de dados, mas a senha informada não corresponder a ele.

Testando-se as trocas de mensagens entre os agentes nas situações descritas acima, mantiveram-se em 2 mensagens trocadas em cada situação de erro.

Figura 6.8: Trocas de Mensagens no comportamento Recuperar Dados Usuário em situação de erro



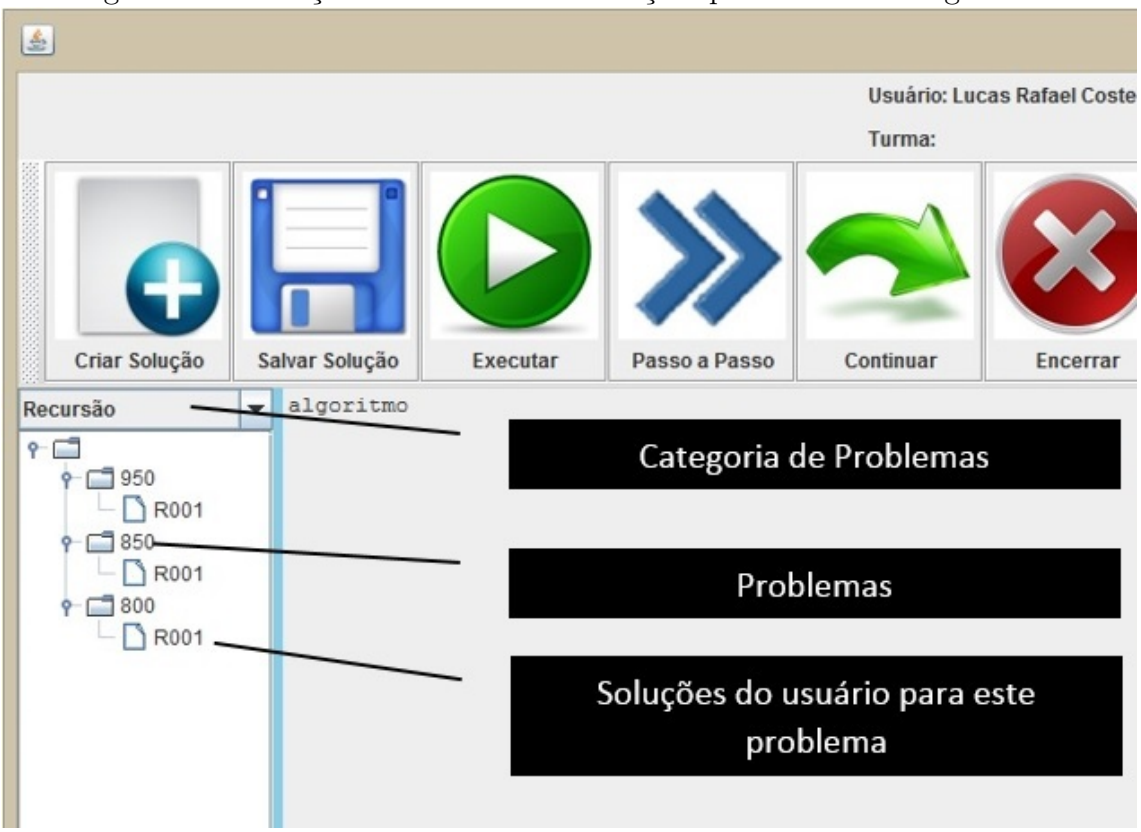
Resumindo, a operação de recuperação das informações do usuário (login), independente dos dados informados ou do resultado (sucesso ou falha), sempre desencadeará a troca de 2 mensagens entre os agentes Interface Gráfica e Gerenciador de Usuários.

6.5.3 Comportamento Recuperar Lista de Problemas do Portal de Algoritmos, e as soluções de determinado usuário

O comportamento Recuperar lista de Problemas e Soluções é ativado automaticamente pelo Agente Interface Gráfica ao final da operação de login do usuário, no caso de sucesso. Logo, não é necessário que o usuário solicite tal operação.

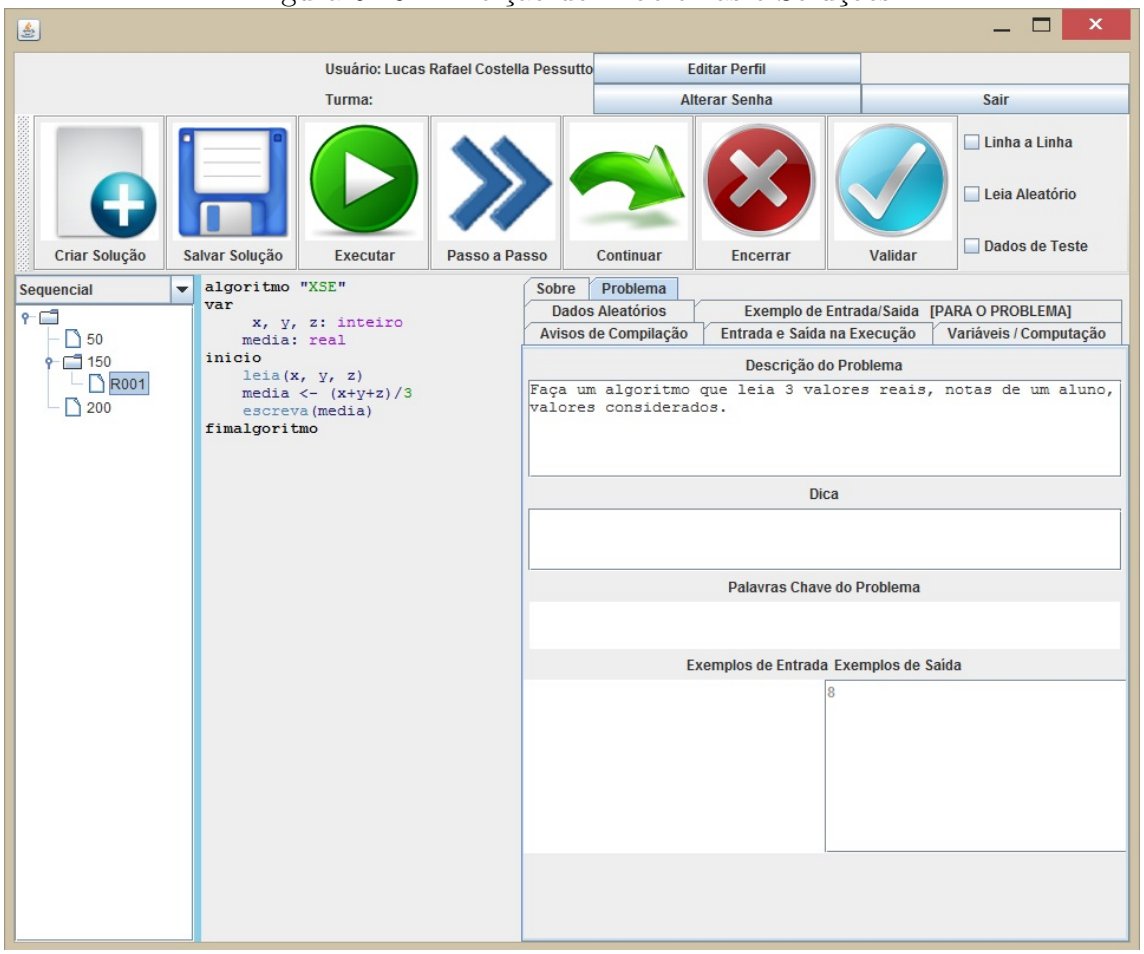
Com os dados obtidos, o agente Interface Gráfica popula um objeto da classe `javax.swing.JTree` que exibe os problemas divididos em suas categorias, com as soluções aparecendo como nodos filhos de seus respectivos problemas, como pode ser visto na Figura 6.9.

Figura 6.9: Exibição de Problemas e Soluções pelo Portal de Algoritmos



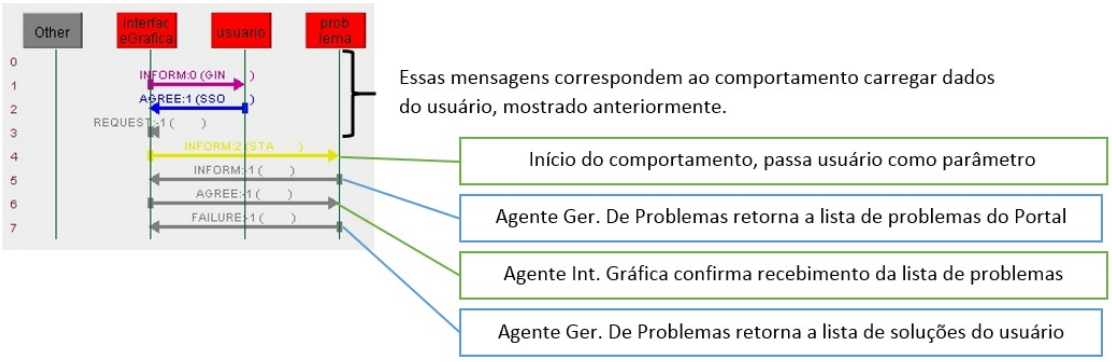
Se um problema for selecionado, suas informações serão exibidas em uma aba do portal e se uma solução for selecionada, além de mostrar os dados do problema, a solução é carregada para o editor de código. A Figura 6.10 demonstra essa funcionalidade em funcionamento.

Figura 6.10: Exibição de Problemas e Soluções



A Figura 6.11 mostra o resultado da execução deste comportamento. Percebe-se que ele possui um número fixo de mensagens trocadas: duas requisitando os dados para o Agente Gerenciador de Problemas e duas com as respostas a esta requisição.

Figura 6.11: Trocas de Mensagens no comportamento Recuperar Lista de Problemas e Soluções



Um problema que pode ser originado deste comportamento diz respeito ao ta-

manho das listas que serão enviadas, pois para a execução dos testes foi considerado um ambiente simulado que continha apenas dez problemas e quatro soluções cadastradas. Ao transpor isso ao portal de algoritmos atual, com mais de 300 problemas cadastrados, as mensagens trocadas poderão ser muito grandes.

Para resolver esse problema, pode-se dividir esse comportamento em dois: O primeiro deles recupera somente os nomes dos algoritmos cadastrados na plataforma, para a exibição na JTree. O outro comportamento é disparado no momento em que se seleciona um problema da lista, onde serão solicitados os dados desse problema. O mesmo aconteceria à lista de soluções.

7 CONCLUSÃO

Neste trabalho foi proposta uma arquitetura multiagente para o Portal de Algoritmos da *UCS*. A principal motivação para essa proposta se deu em virtude do grande potencial que essa plataforma possui para o acoplamento de novas funcionalidades, principalmente na área de Inteligência Artificial.

Quando um programador projetar um novo agente para integrar à plataforma, ele já conhecerá o protocolo de comunicação, bem como as responsabilidades de cada um dos agentes que integram o Portal de Algortimos e possuirá um framework robusto para o desenvolvimento de seu agente. Todas essas facilidades permitirão que o programador se concentre na resolução de seu problema e não se preocupe em criar mecanismos de integração de seu agente ao sistema já existente.

O trabalho de definição de uma arquitetura multiagente foi precedido por um estudo bibliográfico, que permitiu o aprofundamento nos temas relativos a agentes e sistemas multiagentes. Foi realizada também uma análise de dois frameworks para o desenvolvimento de Sistemas Multiagentes, bem como a comparação entre tais frameworks.

Em paralelo a estas atividades foi realizado o estudo do Portal de Algoritmos, identificando a maneira como a aplicação foi constituída e definindo suas principais funcionalidades.

Concluído esse estudo foi desenvolvida a arquitetura descrita no capítulo 5, e após isso ocorreu a sua implementação. Esta atividade iniciou com a reestruturação da interface Gráfica do Portal de algoritmos, a configuração de um ambiente para a realização dos testes, a implementação dos agentes da plataforma e adaptação da versão existente do Portal de Algoritmos.

Ao final desse estudo, pode-se concluir que tal mudança no sistema é viável, mas depende primeiramente de uma reestruturação do mesmo, devido ao fato de haver um forte acoplamento entre as classes do Portal de Algoritmos atual. Mas que a mudança de paradigma é bem-vinda devido ao fato de ela melhorar a estruturação do portal e permitir sua modificação e a inserção de novos agentes de maneira facilitada.

Como trabalhos futuros, sugere-se a reestruturação do compilador e da interface

gráfica do Portal de Algoritmos antigo, de modo a remover o forte acoplamento existente entre essas entidades, a implementação das funcionalidades propostas no capítulo 5 que não foram programadas.

Quando estas atividades forem concluídas, pode-se dar início ao desenvolvimento de novos agentes inteligentes que se beneficiem da arquitetura proposta e que agreguem novas funcionalidades ao Portal de Algoritmos, como exemplo pode-se citar: um sistema de recomendação de problemas aos usuários, uma plataforma de estudos colaborativa, que permita que os alunos que utilizam o portal estudem em grupo e compartilhem seu conhecimento ao mesmo tempo que utilizem o portal de algoritmos e por fim cito o trabalho do colega Israel Vargas de Sá, que propôs a gameficação do Portal de Algoritmos em seu trabalho "O uso da ludificação em um ambiente de desenvolvimento de algoritmos".

REFERÊNCIAS

BELLIFEMINE, F.; CAIRE, G.; GREENWOOD, D. **Developing Multi-Agent Systems with JADE**. London: John Wiley Sons Ltd, 2007.

BITTENCOURT, G. **Inteligência Artificial**: ferramentas e teorias. 3.ed. Florianópolis: Editora da UFSC, 2006.

BORDINI, R. H. et al. **Multi-Agent Programming**: languages, tools and applications. New York: Springer, 2009.

DICKINSON, I. J. Agent Standards. **HP Laboratories Technical Report HPL**, [S.l.], 1997.

DORNELES, R. V.; JR., D. P.; ADAMI, A. G. AlgoWeb: um ambiente baseado na web para aprendizado de algoritmos. **Revista Novas Tecnologias na Educação**, [S.l.], v.9, n.2, 2011.

FERREIRA, S. L. C.; GIRARDI, R. Arquiteturas de Software Baseadas em Agentes: do nível global ao detalhado. **Revista Eletrônica de Iniciação Científica da SBC, Porto Alegre**, [S.l.], 2000.

REZENDE, S. O. **Sistemas Intligentes**: fundamentos e aplicações. Barueri - SP: Manole, 2003.

GENESERETH, M. R.; KETCHPEL, S. P. Software agents. **Commun. ACM**, [S.l.], v.37, n.17, p.48 – 53, 1994.

JIAC - Java-based Intelligent Agent Componentware. <Disponível em: `http://www.jiac.de/`>. Acesso em: 20 de Maio de 2013.

JúNIOR, J. C. R. P. et al. AVEP Um Ambiente de Apoio ao Ensino de Algoritmos e Programação. **Anais do XXVI Congresso da SBC**, [S.l.], 2006.

JúNIOR, J. C. R. P.; RAPKIEWICZ, C. E. Um Ambiente Virtual para apoio a uma Metodologia para Ensino de Algoritmos e Programação. **Revista Novas Tecnologias na Educação - CINTED / UFRGS**, [S.l.], v.3, n.2, Novembro 2005.

JOHNSON, R. E.; FOOTE, B. Designing Reusable Classes. **Journal of object-oriented programming**, [S.l.], v.1, n.2, p.22 – 35, 1988.

LUGER, G. F. **Inteligência Artificial**: fundamentos e aplicações. Porto Alegre: Bookman, 2004.

POOLE, D. L.; MACKWORTH, A. K. **Artificial Intelligence**: foundations of computational agents. Cambridge: Cambridge University Press, 2010.

POSLAD, S.; BUCKLE, P.; HADINGHAM, R. **The FIPAOS agent platform**: open source for open standards. 2000.

PREE, W.; SIKORA, H. Design patterns for object-oriented software development. In: SOFTWARE ENGINEERING, 19., New York, NY, USA. **Proceedings...** ACM, 1997. p.663–664. (ICSE '97).

RUSSELL, S. J.; NORVIG, P. **Inteligência Artificial**. 2.ed. Rio de Janeiro: Elsevier, 2004. Traduzido por: Vandemberg D. Souza.

SCHIEL, D. **Associações Biológicas**. <Disponível em: `http://educar.sc.usp.br/ciencias/ecologia/associa.html`>. Acesso em: 22 de Abril de 2013.

SMITH, R. G. The contract net protocol: high-level communication and control in a distributed problem solver. **IEEE Transactions on Computers**, [S.l.], v.100, n.12, p.1104 – 1113, 1980.

THE Foundation of Intelligent Physical Agents (FIPA). <Disponível em: `http://www.fipa.org/`>. Acesso em: 13 de Maio de 2013.

WEISS, G. (Ed.). **Multiagent Systems**: a modern approach to distributed artificial intelligence. Massachusetts: Massachusetts Institute of Technology, 1999.

WOOLDRIDGE, M. **An Introduction to Multiagent Systems**. London: John Wiley Sons, LTD, 2002.