UNIVERSIDADE FEDERAL FLUMINENSE
CENTRO TECNOLÓGICO
INSTITUTO DE COMPUTAÇÃO

HELIOMAR KANN DA ROCHA SANTOS
ROBERTO WEIDMANN MENEZES

SISTEMA MULTI-AGENTES PARA SIMULAÇÃO DE COMPORTAMENTO DE UMA RESIDÊNCIA

NITERÓI

2009

HELIOMAR KANN DA ROCHA SANTOS
ROBERTO WEIDMANN MENEZES

SISTEMA MULTI-AGENTES PARA SIMULAÇÃO DE COMPORTAMENTO DE UMA RESIDÊNCIA

Trabalho realizado para a disciplina "Introdução a Sistemas Multi-Agentes" no curso "Mestrado em Ciência da Computação" da Universidade Federal Fluminense.

ORIENTADOR: PROFª.DR. Viviane Torres da Silva

NITERÓI

“Seja em você a mudança que quer para o mundo”

Ghandi

ABSTRACT

This work uses the ASF framework to develop a simulation of a residence inhabited by residents agents interacting with a central service to request house cleaning services. The residents may have different roles, but everybody call the cleaning service to their houses when the environment becomes dirty or under other conditions. Calling the employee depends on the state of cleanliness and tidiness of the residence's rooms. In that way we can simulate several houses with residents of varied personalities and monitor their learning through dynamic graphics.

Keywords: ASF, framework, multi-agents, agents, simulation, residence.

RESUMO

O presente trabalho utiliza o framework ASF para desenvolver uma simulação de uma residência habitada por agentes moradores interagindo com uma central de serviço para solicitar serviços de faxina. Os moradores podem possuir papeis distintos, mas todos solicitam o serviço de faxina na residência quando o ambiente torna-se sujo ou de acordo com algumas condições. Chamar a empregada depende da situação de limpeza e arrumação dos cômodos da residência. Dessa forma podemos simular residências com vários moradores de personalidades variadas e acompanhar o aprendizado dos moradores através de gráficos dinâmicos.

Palavras-chaves: ASF, framework, multi-agentes, agentes, simulação, residência.

SUMÁRIO

1. Introdução 9

1.1 Objetivo 9

1.2 Motivação 9

1.3 Estrutura do documento 9

2. Estudo Bibliográfico 10

2.1 Agentes 10

2.2 Sistema Multi-Agentes 10

2.3 ASF 11

3. Aplicação 12

3.1 Escopo 12

3.2 Ambiente 12

3.3 Agentes 12

3.3 Crenças 13

3.4 Planos e Objetivos 14

3.5 Ações 15

3.6 Mensagens 16

4. Aprendizagem 17

5. Conclusão 21

6. Referências 22

Apêndice I 23

Apêncide II 29

LISTA DE FIGURAS

Figura 1: Classe ComportamentoHigienico 14

Figura 2: Classe do papel específico *MoradorHigienico* 14

Figura 3: LeaftGoal TornarResidenciaHabitavel.....15

Figura 4: Tabela com as probabilidades de cada ação para cada um dos papeis permitidos aos moradores 17

Figura 5: Pseudocódigo do calculo das probabilidades 18

Figura 6: Pseudocódigo do algoritmo de aprendizagem 18

Figura 7: Gráfico de exemplo da execução do algoritmo de aprendizagem.....19

Figura 8: Pseudocódigo da função heurística adotada 19

Figura 9: Gráfico de exemplo da execução do algoritmo de aprendizagem com a função heurística.....20

LISTA DE ABREVIATURAS E SIGLAS

ASF: Java Enterprise Edition

API: Application Programming Interface

POO: Programação Orientada a Objetos

URL: Uniform Resource Locator.

IDE: Integrated Development Environment

1. INTRODUÇÃO

1.1 Objetivo

O desenvolvimento desse trabalho foi proposto como forma de aprendizado do paradigma orientado a agentes. ASF é orientado a objetos, mas a idéia desse framework é facilitar a utilização dos conceitos de *Agentes* utilizando OO.

1.2 Motivação

O crescimento do poder computacional faz aumentar agressivamente o desenvolvimento de novas soluções de software. As idéias desses softwares não necessariamente são novas, o conceito de Agentes existe desde a década de 80, no entanto as condições necessárias para o desenvolvimento de tal paradigma eram precárias. Agentes é um ramo evolutivo de uma linguagem de programação, é uma ferramenta complementar para resolver problemas específicos. Um agente nesse paradigma representa a abstração de um ser humano efetuando alguma ação para alcançar determinado objetivo. Todo o contexto real pode ser simulado de alguma forma, como por exemplo, o ambiente onde o agente se encontra, os papéis que ele pode operar, os seus planos e objetivos para atingir suas metas etc.

Então o aprendizado e evolução desse paradigma são cruciais para o desenvolvimento e resolução de problemas de mais alto nível, diminuindo assim preocupações necessárias da OO para os desenvolvedores e facilitando ainda mais a vida do usuário final.

1.3 Estrutura do documento

O restante desse documento é dividido em mais duas seções principais e a conclusão, na primeira seção faremos um estudo bibliográfico dos conceitos de agentes e do framework utilizado, já na segunda seção será apresentada com mais detalhes a explicação da aplicação implementada pelo grupo.

2. ESTUDO BIBLIOGRÁFICO

Nesse capítulo faremos uma breve apresentação do paradigma orientado a Agentes, assim como a apresentação do framework utilizado ASF.

2.1 Agentes

O termo "agente" existe na informática desde a década de 80 porém, devido a sua complexidade, cada autor adota uma definição que se adapta melhor ao seu contexto de trabalho[1]. No caso, algumas das definições mais referenciadas são[2]:

- [3] apresenta que "um agente é qualquer coisa que pode ser vista percebendo um ambiente por meio de sensores e atuando no mesmo através de atuadores".

- [4], [5] e [6] descrevem que "agentes são sistemas computacionais que habitam um ambiente complexo e dinâmico, sensoreiam e atuam autonomamente sobre este ambiente, realizando desta maneira uma série de metas e tarefas as quais foram projetados".

- [7] diz que "agentes inteligentes realizam continuamente três funções: percepção das condições dinâmicas de um ambiente, ação de modo a afetar as condições do ambiente e raciocínio para interpretar percepções, realizar inferências e determinar ações".

É notável a semelhança em comum nas definições apresentadas: pode-se dizer que agentes são entidades autônomas especializadas na execução de uma determinada tarefa, tendo a capacidade de perceber o ambiente em que atuam, tomam decisões sobre informações obtidas deste ambiente e executam alguma tarefa como resultado [3].

2.2 Sistema Multi-Agentes

Conforme apresentado em [8] e [9], um sistema multi-agentes possui a capacidade de se adaptar a novas situações, mudando sua organização e determinando a formação da coletividade dos agentes. Também permite um alto nível de abstração, por se apresentar na forma de uma metáfora natural para a modelagem de sistemas complexos de forma distribuída, aonde o conhecimento, o controle e os recursos podem estar localizados em diferentes plataformas, arquiteturas e sistemas.

As arquiteturas de SMA se baseiam em lógica e utilizam mecanismos de raciocínio de um agente em relação a sua percepção de mundo, com e para os outros agentes e também para o ambiente. Divide-se essas arquiteturas em dois tipo:

- SMA reativos: Os agentes agem e comportam-se baseados em estruturas e definições pré-definidas e agem sob um estímulo-resposta. Da mesma forma selecionam ações com base na percepção atual, ignorando o restante do histórico de percepções [10].

- SMA cognitivos: Os agentes possuem um estado mental, podendo ser crenças, desejos e intenções, e raciocinam para construir um plano de ações que leva a um objetivo pretendido. Com a capacidade de alterar seu funcionamento a fim de adaptar-se às condições e situações do ambiente, além de comunicar-se, sua complexidade cresce, pois eles também raciocinam sobre as ações a serem realizadas [3].

2.3 ASF

Framework ASF é um framework orientado a objetos para implementar sociedades de agentes. Sociedade de agentes são sistemas multi-agentes nos quais é necessário representar agentes desempenhando papéis em organizações [11]. Usando este framework, é possível implementar vários agentes, cada um desempenhando inúmeros papéis em diferentes organizações.

Baseado no framework conceitual conhecido como TAO [12], define um conjunto de abstrações que caracteriza sociedades de agentes. De modo geral, essa implementação permite a existência de agentes com múltiplos papeis, seguindo planos independentemente, através de ambientes e organizações distintas [13]. Além dessas características, utiliza o modelo de BDI [14] e a especificação de comunicação ACL [15].

3. APLICAÇÃO

Nessa seção será apresentada a aplicação que foi desenvolvida no trabalho. Para tal, utilizamos diagramas de atividades, classes e diagramas de estados que auxiliaram o entendimento. Os diagramas completos e partes mais significativas do código estarão contidos nos apêndices, assim como a referência para o projeto completo localizado em um controlador de versões online.

3.1 Escopo

O Sistema simula uma vivência simplificada entre moradores de uma residência que ao detectarem certas características no ambiente fazem solicitações de faxina para uma central de serviços. Na central existe uma secretária que recebe o pedido de faxina e os coloca em um mural para que as faxineiras cadastradas nesse ambiente possam fazer o serviço. Os moradores habitam a residência composta por cômodos e esperam ser felizes na mesma. Naturalmente os agentes moradores possuem personalidades distintas que acarretam em uma combinação de ações do tipo limpar, sujar, arrumar, desarrumar, que são tomadas com grau de probabilidade compatível com o seu comportamento.

3.2 Ambiente

Existem dois tipos de ambientes: (1) *Central de Serviços,* esse ambiente só é criado uma vez, juntamente com uma agente *Secretária*; (2) *Residência,* esse tipo de ambiente pode ser criado sem restrições de número. Os ambientes possuem três cômodos registrados como objetos, podendo possuir vários moradores e faxineiras. Em determinados momentos agentes podem se locomover de um ambiente para o outro.

3.3 Agentes

Foram criados três tipos básicos de papéis para os agentes:

- O papel de “Morador” tem a função de identificar quem de fato habita o ambiente em questão (Residência). Existem seis especificações desse papel: (1) Relaxado; (2) Organizado; (3) Higiênico; (4) Não Higiênico; (5) Equilibrado; (6) Bagunceiro. Um Agente que possui um papel de Morador poderá limpar, sujar, arrumar e desarrumar, de acordo com suas crenças de sua especificação, por exemplo, o papel equilibrado tem chances de limpar, sujar, arrumar e desarrumar iguais (25% cada). Identificando níveis críticos de limpeza e arrumação o morador faz uma solicitação à Central de Serviços para uma faxina em sua residência.

- O papel de “Faxineira” foi criado para vincular um agente ao perfil de faxineiro. Um agente nesse papel tem obrigação de limpar e arrumar a residência e assim esse agente cumpre seu plano de faxina. Os agentes faxineiros são cadastrados em uma Central de Serviços, lá eles ficam verificando um mural de requisições no qual eles extraem a informação de qual residência está solicitando serviço de faxina. Para cada faxina o agente tem um tempo máximo para efetuar seu trabalho e quando o faxineiro termina uma ação e percebe que o seu tempo de faxina naquela residência se esgotou, ele retorna para a central de serviços onde volta a verificar o mural de requisições, simulando uma faxineira real com tempo limitado que trabalha para uma central de serviços.

- O papel de “Secretária” foi criado para atender as solicitações de faxina das residências que são recebidas na Central de Atendimento. O agente nesse papel ao receber uma solicitação de faxina deve atualizar o mural de requisições para que as faxineiras possam se direcionar às devidas residências.

3.3 Crenças

Para descrever o perfil de cada agente, crenças iniciais foram criadas e vinculadas aos agentes. Para facilitar o uso das crenças foram criadas as classes de comportamento, que nada mais são do que um conjunto de crenças. As classes de comportamento são utilizadas para criar os papeis específicos.

As crenças utilizadas foram:

- Limpar
- Sujar
- Arrumar
- Desarrumar
- ChamarEmpregada

Com exceção da crença “ChamarEmpregada”, que tinha sempre o valor “true”, essas crenças possuíam valores inteiros discretos de maneira que a soma alcance 100, assim cada valor representa a probabilidade de um agente decidir o que fazer diante de algumas situações.

A **Figura 1** representa a classe do comportamento Higiênico e a **Figura 2** representa o papel específico “MoradorHigienico” utilizando-se das crenças pré-estabelecias na classe comportamento.

```
…
public class Higienico extends Comportamento{

    public Higienico(){

    }

    @Override
    protected void carregaCrencas() {
        crencas.add(new LeafBelief("int", "dessarruma", 15));
        crencas.add(new LeafBelief("int", "arruma", 15));
        crencas.add(new LeafBelief("int", "limpa", 60));
        crencas.add(new LeafBelief("int", "suja", 10));
        crencas.add(new LeafBelief("boolean", "chamaEmpregada", true));
    }
…
}
```

Figura 1: Classe ComportamentoHigienico

```
…
public class MoradorHigienico extends Morador implements Serializable {

    public MoradorHigienico(String nome, MainOrganization organizacao) {
        super("Higienico" + nome, organizacao);
        beliefs = new ArrayList(new Higienico().getCrencas());

        this.setOwner(organizacao);
    }
…
}
```

Figura 2: Classe do papel específico *MoradorHigienico*

Nesse caso definiu-se que um agente, com as crenças de um Papel Higiênico, ao analisar um cômodo em 60% das vezes ele decidiria em **limpar** o cômodo, 15% arrumar, 15% desarrumar e 10% das vezes ele decidiria sujar.

3.4 Planos e Objetivos

Cada agente tem um plano e cada plano um objetivo. O plano “PlanoFaxina” e o “PlanoSecretaria” tem o objetivo “TornarResidenciaHabitavel” e são atribuídos aos agentes

com papel “Faxineira” e “Secretaria” respectivamente, assim como o plano “PlanoHabitar” tem a meta “ResidirFeliz” e é atribuído ao agente morador.

A **figura 3** mostra a implementação do objetivo simples “TornarResidenciaHabitavel”.

```
...
public class TornarResidenciaHabitavel extends LeafGoal implements
Serializable{

    public TornarResidenciaHabitavel(){
        super("boolean", "Tornar_Residencia_Habitavel",true);
    }

    public TornarResidenciaHabitavel(boolean valor){
        super("boolean", "Tornar_Residencia_Habitavel",valor);
    }
}
...
```

Figura 3: LeaftGoal TornarResidenciaHabitavel

3.5 Ações

As treze ações que foram definidas para os respectivos planos são as seguintes:

- “AcaoArrumar” – “PlanoHabitar” e “PlanoFaxina”
- “AcaoDesarrumar” – “PlanoHabitar”
- “AcaoLimpar” – “PlanoHabitar” e “PlanoFaxina”
- “AcaoSujar” – “PlanoHabitar”
- “AcaoVerificarComodo” – “PlanoHabitar” e “PlanoFaxina”
- “AcaoChamarEmpregada” – “PlanoHabitar” e “PlanoSecretaria”
- “AcaoAtenderRequisicao” – “PlanoSecretaria”
- AcaoAtualizarQuadroTarefas – “PlanoSecretaria”
- AcaoIrParaACentralAtendimento – “PlanoFaxina”
- AcaoVisitarResidencia – “PlanoFaxina”
- AcaoTrocarComodo – “PlanoFaxina” e “PlanoHabitar”

- AcaoFazNada – “PlanoFaxina” e “PlanoSecretaria”
- AcaoPegarFaxina – “PlanoFaxina”

Iniciante adotamos que agentes moradores praticam a ação de verificar cômodo (AcaoVerificarComodo) contida no seu plano (PlanoHabitar). Para isso ao iniciar a simulação enviamos uma mensagem para o agente verificar um determinado cômodo. Inicialmente agentes secretarias também praticam a ação de fazer nada (AcaoFazNada) até que recebam alguma requisição e as faxineiras tentam pegar alguma faxina no quadro de avisos pela ação AcaoPegarFaxina.

3.6 Mensagens

Assim como os seres detectam as coisas ao redor através de alguma informação pelo cérebro “processada”, o “sentido” do agente em nosso sistema foi através das mensagens, ou seja, a mensagem que vai dizer o que o agente vai fazer de acordo com cada situação. Exemplo, ao entrar em um cômodo um agente verifica o estado do mesmo e ao perceber que o cômodo está precisando ser limpo, ele pode enviar uma mensagem para si mesmo fazendo com que sua próxima ação seja limpar o cômodo, ou então ele propaga essa mensagem para a central de serviços solicitando faxina. Exemplo 2, ao sujar um quarto o agente envia para si próprio que ele tem que verificar o cômodo, essas ações são tomadas de acordo com o papel dos agentes, já citados na seção de crenças.

4. APRENDIZAGEM

O papel Modador, de todos os abordados neste trabalho, permite a execução de ações baseadas em probabilidade, ou seja, que são executadas em ordem aleatória. Essas ações são: (1) AcaoLimpar, referente a limpeza de um cômodo; (2) AcaoSujar, referente ao ato de sujar um cômodo; (3) AcaoArrumar, referente a arrumação de um cômodo; e (4) AcaoDesarrumar, referente ao ato de desarrumar um cômodo.

A execução dessas ações representam, de forma genérica, as conseqüências de todas as possíveis tarefas realizadas por um morador, na vida real, que influenciam o trabalho de uma faxineira.

A probabilidade de cada uma das ações depende diretamente do papel do morador, influenciado pelo seu comportamento. Os possíveis papeis referente a um morador são: (1) MoradorBagunceiro, (2) MoradorEquilibrado, (3) MoradorHigienico, (4) MoradorNaoHigienico, (5) MoradorOrganizado e (6) MoradorRelaxado. O tipo de comportamento é dedutível pelo nome. A **Figura 4** apresenta as probabilidades de executar essas ação para cada um dos papeis apresentados.

Papel	Ações			
	AcaoLimpar	AcaoSujar	AcaoArrumar	AcaoDesarrumar
MoradorBagunceiro	15%	10%	15%	60%
MoradorEquilibrado	25%	25%	25%	25%
MoradorHigienico	60%	10%	15%	15%
MoradorNaoHigienico	10%	60%	15%	15%
MoradorOrganizado	15%	15%	60%	10%
MoradorRelaxado	10%	40%	10%	40%

Figura 4: Tabela com as probabilidades de cada ação para cada um dos papeis permitidos aos moradores

Para permitir um maior grau de realismo nos resultados obtidos durante a simulação, foi proposto um algoritmo de aprendizagem possibilitando os moradores alcançarem seu objetivo mais rapidamente: residindo feliz, ou seja, viver em uma casa limpa e arrumada.

Toda vez que uma ação é executada com sucesso, essa informação é armazenada, formando um registro de todas as ações realizadas por cada morador. Essas informações serão utilizadas posteriormente para a aprendizagem.

Quando um morador alcança o seu objetivo, é utilizado o histórico de ações para tentar inferir quais ações o levaram a este fim. Com essas informações é calculado a probabilidade delas acontecerem, através de probabilidade simples. O pseudo-algoritmo a seguir explica em detalhes esse calculo.

```
Pseudocódigo do calculo das probabilidades
Entrada: 'Lista com <nome da ação, número de ocorrências>'
Saída: 'Lista com <nome da ação, probabilidade>'
1:  calcular o somatória das ocorrências das seguintes ações:
      AcaoLimpar, AcaoSujar, AcaoArrumar e AcaoDesarrumar
2:    para cada elemento na lista de entrada
3:      se for uma das ações
            (AcaoLimpar, AcaoSujar, AcaoArrumar, AcaoDesarrumar)
        então
4:        armazenar na lista de saída
         <nome da ação, total de ocorrências dessa ação / somatório de ocorrências>
5:      fim se
6:    fim para
7:    retornar a lista de saída
```

Figura 5: Pseudocódigo do calculo das probabilidades

Logo após o calculo anterior, as probabilidades das ações envolvidas no histórico serão utilizadas para atualizar as chances atuais do morador. Para isso é realizada uma média aritmética entre o novo valor calculado com o utilizado até então pelo agente.

A seguir é apresentado um pseudo-algoritmo explicando o processo como um todo.

```
Pseudocódigo do algoritmo de aprendizagem
Entrada: 'Lista com <nome da ação, número de ocorrências>',
          'probabilidade das ações do morador'
Saída: 'Lista com <nome da ação, novas probabilidades>'
1:    calcular a lista de <nome da ação, probabilidade>
      através do pseudocódigo 1, usando a lista de entrada
2:    para cada elemento na lista de <nome da ação, probabilidade>
3:        adicionar à lista de saída
          <nome da ação, (probabilidade calculada + probabilidade atual) / 2>
4:    fim para
5:    retornar a lista de saída
```

Figura 6: Pseudocódigo do algoritmo de aprendizagem

A Figura 7 apresenta um gráfico utilizando as probabilidades das ações de um agente em execução junto com o algoritmo proposto.

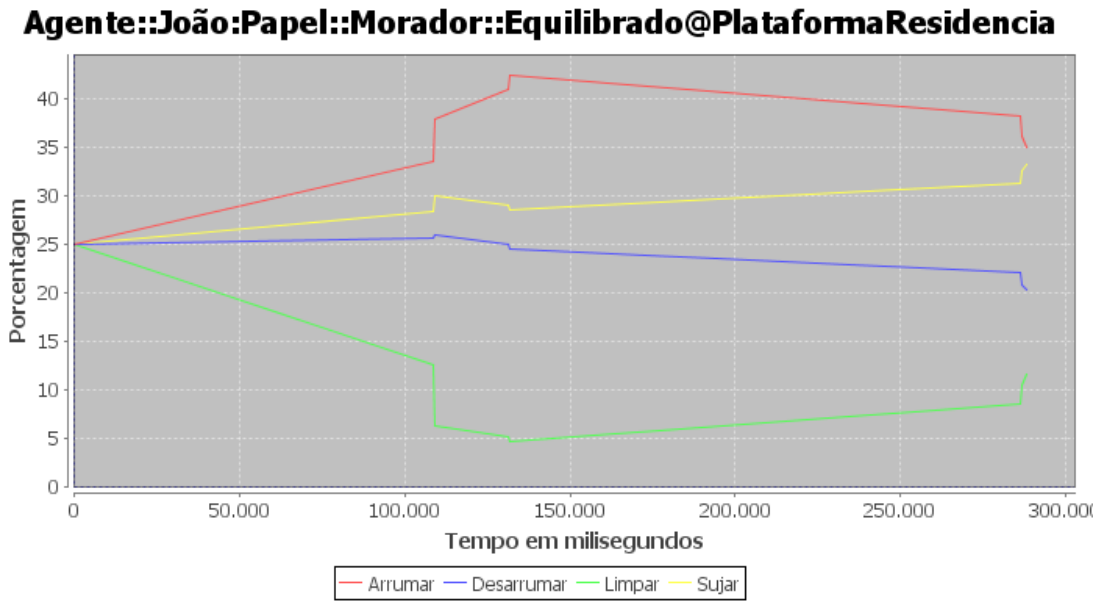


Figura 7: Gráfico de exemplo da execução do algoritmo de aprendizagem

Após vários testes, foi perceptível que, ao longo do tempo, dificilmente um morador conseguiria alterar a probabilidades de suas ações a ponto de apresentar comportamento semelhante ao de outros papeis, ou seja, apesar de tratarmos de eventos aleatórios, com o passar do tempo as probabilidades convergiam para os valores iniciais.

Na tentativa de aprimorar a aprendizagem, foi proposta uma nova alteração utilizando uma função heurística para definir quando seria útil aprender. No caso, o morador somente aprenderia se essa nova função permitisse. A função heurística adotada pode ser resumida na seguinte frase: "Se a probabilidade de limpar e arrumar a casa superar as atuais, então as aprende".

A seguir é apresentado o pseudo-algoritmo para o seu calculo.

```
Pseudo-algoritmo 3 Pseudocódigo da função heurística adotada
Entrada: 'Probabilidades atuais do morador',
         'Probabilidades calculadas baseado no histórico de ações'
Saída: 'Valor booleano se pode ou não aprender'
1:     se
         (probabilidade atual do morador para AcaoLimpar
          + probabilidade atual do morador para AcaoArrumar)
          > (probabilidade calculada para AcaoLimpar
             + probabilidade calculada para AcaoArrumar)
       então
2:        retornar 'Verdadeiro'
3:     senão retornar 'Falso'
```

Figura 8: Pseudocódigo da função heurística adotada

A Figura 9 apresenta um gráfico, semelhante ao da Figura 7, utilizando a função heurística para o aprendizado.

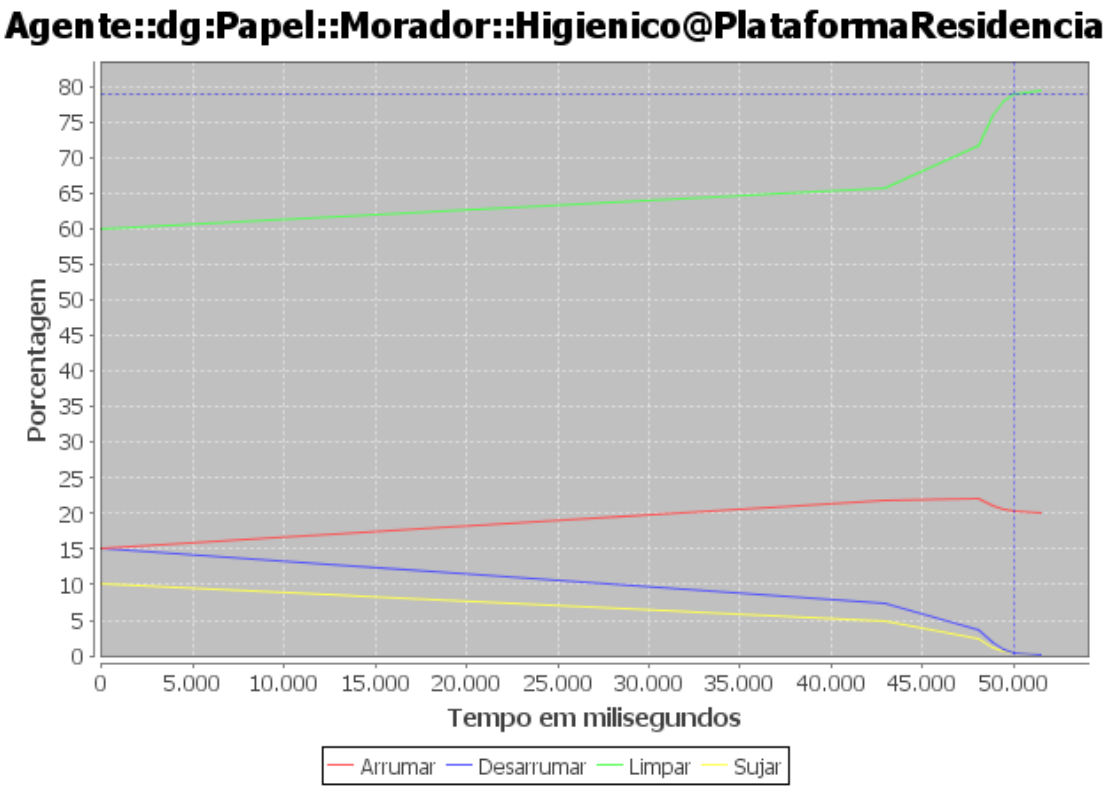


Figura 9: Gráfico de exemplo da execução do algoritmo de aprendizagem com a função heurística

Após uma análise mais detalhada, concluímos que o uso desta função permitiu que os moradores tendessem a limpar e arrumar mais a sua residência e, conseqüentemente, atingindo seu objetivo, porém eles demoraram mais para aprender.

5. CONCLUSÃO

O trabalho conseguiu atingir sua finalidade de fixar os conhecimentos adquiridos ao longo da disciplina. Pensar orientado a agentes não é tarefa simples, e quando se sem um framework que te apóie é sempre proveitoso.

O framework ASF não é orientado a agentes, mas permite que o desenvolvedor não se preocupe com muitos detalhes implementais não relacionado à *agentes*, ele também oferece meios de abstrair um pouco o conceito de objetos e se enquadrar nos conceitos de *agentes*.

O resultado obtido com o sistema foi muito positivo, foi além do esperado. Consolidando que a idéia de se resolver problemas pensando orientado a agentes é muito promissora.

Como foi o primeiro contato tanto com esse paradigma, tanto conceitual quanto implemental, muitos problemas foram descobertos ao longo da implementação, inviabilizando algumas idéias e desprendendo um alto teor de energia do grupo.

Apenas a documentação oferecida no site do framework não foi suficiente para entendermos exatamente como o framework se comportava, dessa forma não sabíamos o que era suficiente para fazer a aplicação funcionar da maneira que estávamos imaginando. Tivemos que solicitar o código fonte do framework e sobrescrever diversas partes, para que o mesmo atendesse nossas necessidades.

Apesar desse desprendimento de energia, consideramos tudo muito proveitoso, pois entendemos o framework de uma maneira muito mais completa, podendo até sugerir melhorias no mesmo, como um controle de concorrência mais eficiente sem espera ocupada, por exemplo.

Como trabalhos futuros, sugerimos a extensão desse projeto para várias finalidades: (1) Estudo e experimento de reputação entre agentes; (2) Estudo e implementação de outras formas de aprendizado; (3) Implementação e melhoria do Framework ASF de forma a torná-lo mais intuitivo, otimizado, manutenível e escalável.

6. REFERÊNCIAS

[1] MONTESCO, CARLOS ALBERTO ESTOMBELO. UCL - Uma Linguagem de Comunicação para Agentes de Software – 2001.

[2] FAGUNDES, FABIANO. OLIVEIRA, ARY HENRIQUE M. DE. Uma Estrutura Baseada em Agentes Inteligentes para Determinação de uma Sequência de Estudos.
[3] RUSSELL, S., NORVIG, P. Artificial Intelligence: A Modern Approach. Prentice-Hall, 1995.

[4] BRAUN, PETER. BRZOSTOWSKI, JAKUB. KERSTEN, GREGORY. KIM, JIN BEAK. KOWALCZYK, RYSZARD. STREEKER, STEFAN. VAHIDOV, RUSTAM . e-Negotiation Systems and Software Agents: Methods, Models, and Applications.

[5] MAES, P. Artificial Life Meets Entertainment: Life-like Autonomous Agents. Communications of the ACM 38(11): 108-114. 1998.

[6] MAES, P. GUTTMAN, R. H. Agents that can buy and sell. Communication of the ACM 42(3): 81-91. 1999

[7] HAYES-ROTH, BARBARA. Guardian: Autonomous Intelligent Agent for Medical Monitoring, IEEE Int. Systems, 1998.

[8] PASQUALOTTI, PAULO ROBERTO. Programação da Comunicação entre Agentes BDI em um Ambiente SMA.

[9] HUBNER, J.F. SICHMAN, J.S. Organização de Sistemas Multiagentes.

[10] BORDINI, R.H., VIEIRA, R. Linguagens de programação orientadas a agentes: uma introdução baseada em AgentSpeak(L).

[11] PUC. Framework ASF. Disponível em: http://www.les.inf.puc-rio.br/frameworkasf/. Acesso em: 20/05/2009.

[12] J. LIND. MASSIVE: Software Engineering for Multi-agent Systems. 2000.

[13] SILVA, VIVIANE TORRES DA. CORTÉS, MARIELA INÉS. LUCENA, CARLOS JOSÉ PEREIRA DE. An Object-Oriented Framework for Implementing Agent Societies. 2004.

[14] WIKIPEDIA. Belief-Desire-Information software model. Disponível em: http://en.wikipedia.org/wiki/BDI_software_agent. Acesso em: 20/05/2009.

[15] FIPA. FIPA Agent Communication Language Specifications. Disponível em: http://www.fipa.org/repository/aclspecs.html. Acesso em: 20/05/2009.

APÊNDICE I

Diagrama de Atividade do papel Empregada

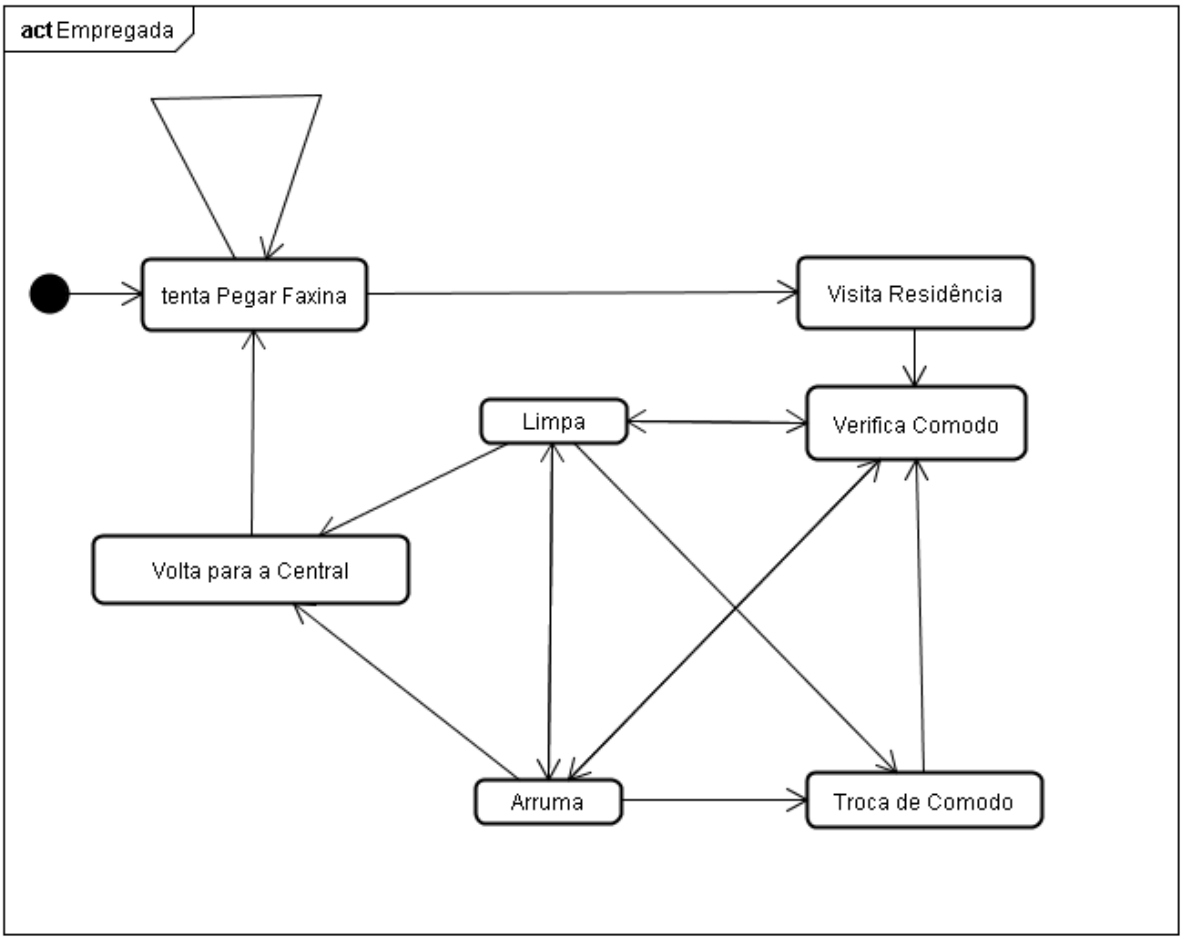


Diagrama de Atividade do papel Morador

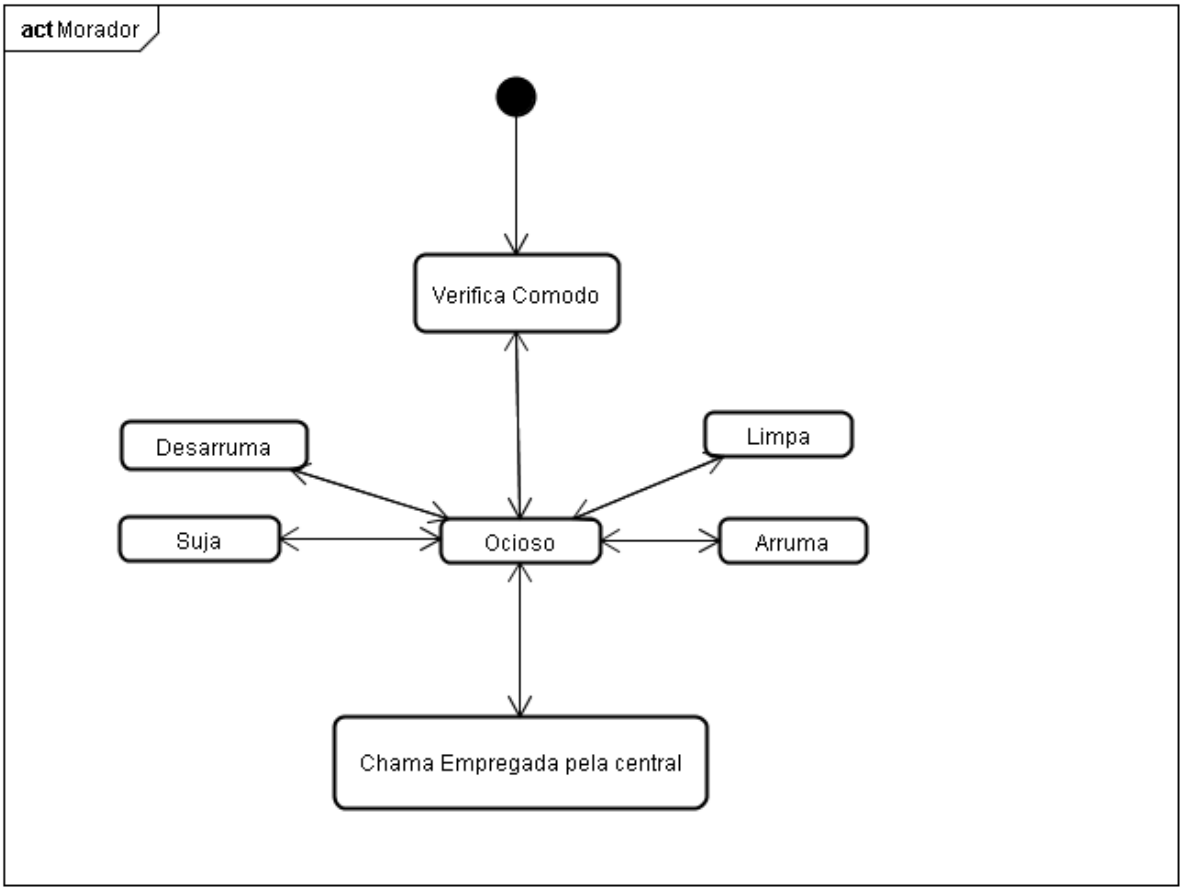


Diagrama de Atividade do papel Secretária

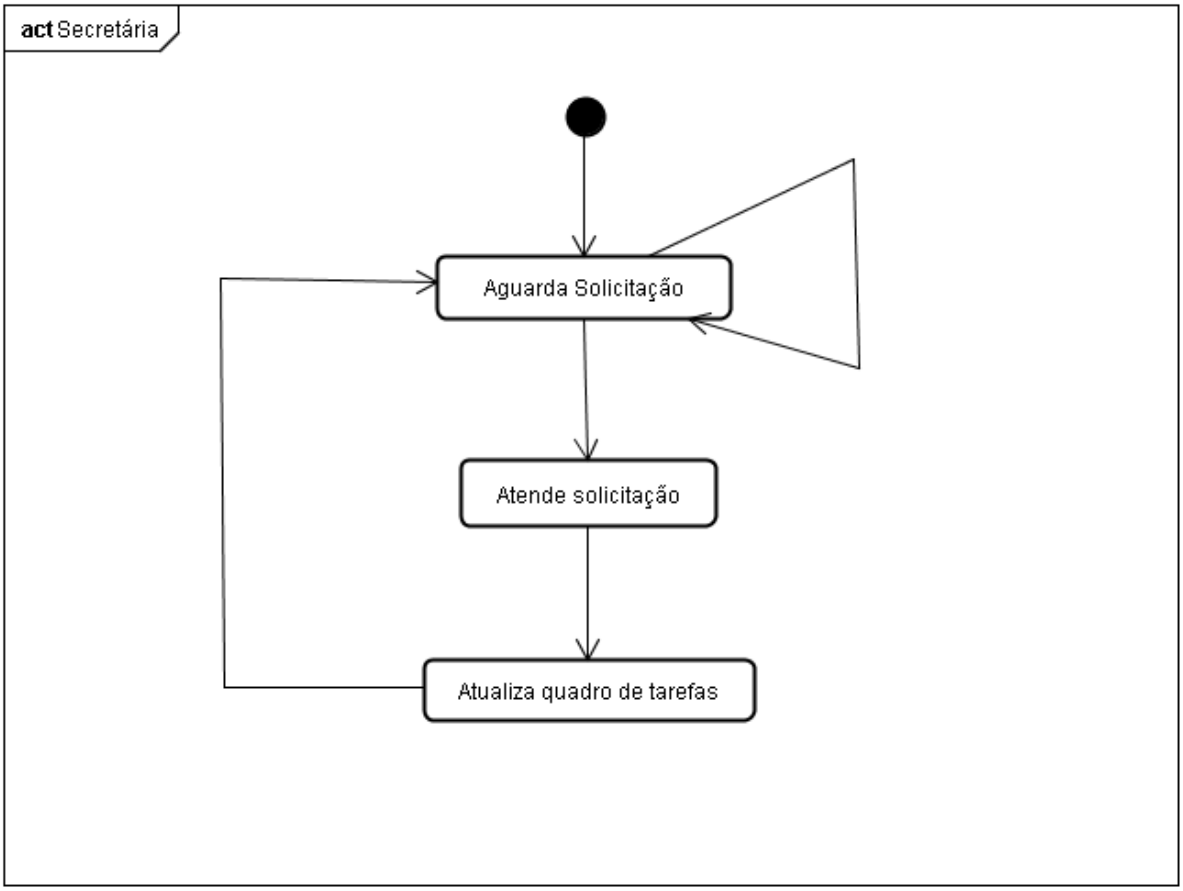
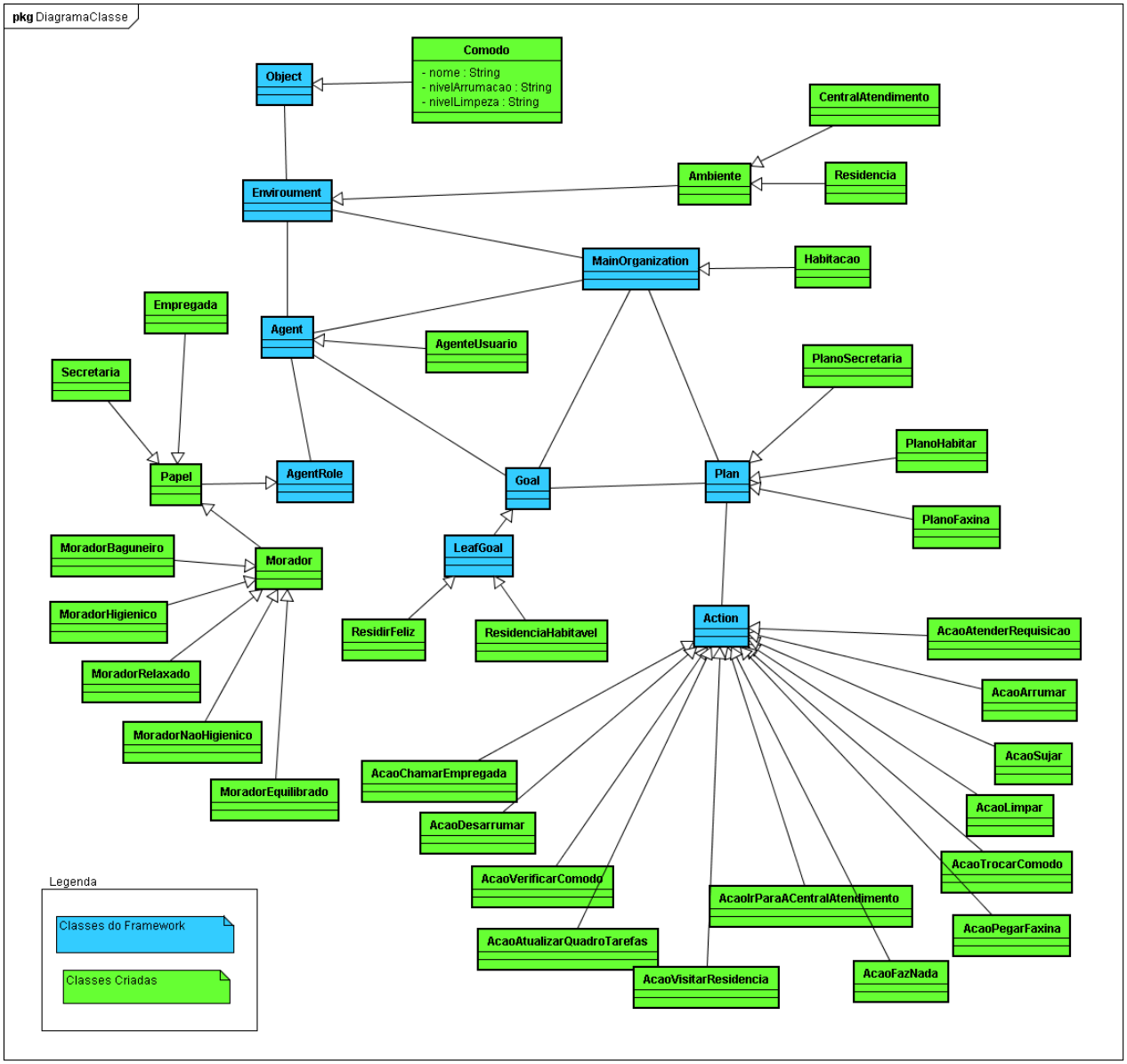
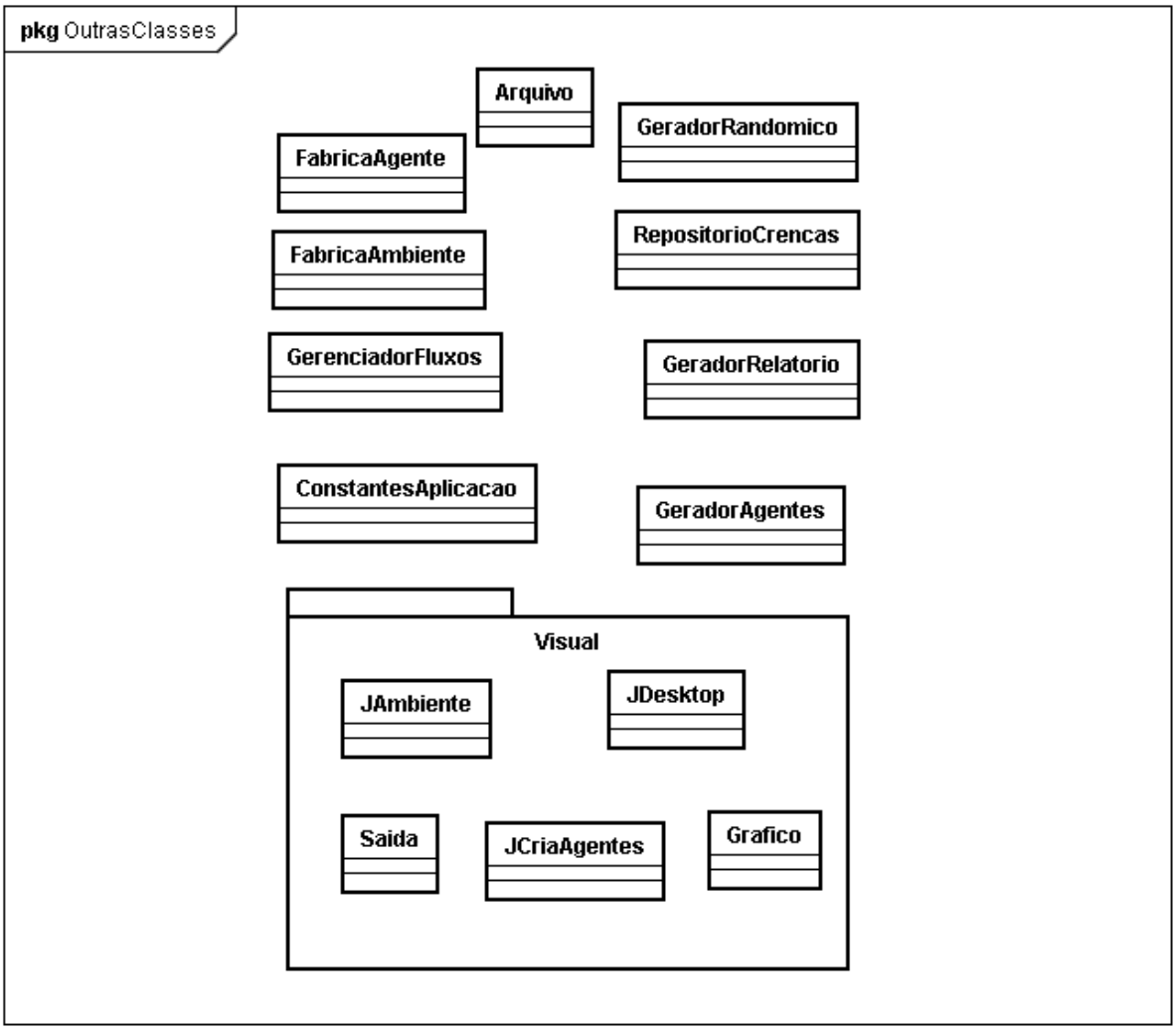


Diagrama de Classes do sistema



Outras Classes utilizadas



APÊNCIDE II

Ferramentas para desenvolvimento

- A IDE utilizada foi NetBeans 6.1 e 6.5 pela familiarização da que os criadores do projeto tinham com essas IDEs.
- Máquina virtual Java 5 e Java 6
- Máquinas: Intel Pentium 4 HT LGA 775 3.4 Ghz, 2 Gb RAM DDR2, Sistema operacional XP
- Máquinas: Intel Centrino Core2Duo 2.4 Ghz, 2 Gb RAM DDR3, Sistema operacional MAC OS X
- Máquinas: Intel Centrino Core2Duo 2.0 Ghz, 2 Gb RAM DDR2, Sistema operacional Windows Vista

Código fonte

- O código fonte encontra-se sob controle de versão no link http://code.google.com/p/asf-ap-simulation/source/checkout .