# 5
# Estudo de Caso

## Resumo

*Neste capítulo será apresentado o resultado do uso do ambiente proposto nesta dissertação no desenvolvimento de um sistema multi-agente, o LearnAgents, que participou do TAC em 2004. Este exemplo tem por objetivo servir como prova de conceito do uso do Albatroz.*

A seguir, será mostrado um exemplo, com uma explicação detalhada, do desenvolvimento de um sistema multi-agente com o Albatroz. Este sistema foi desenvolvido anteriormente e sem auxílio computacional. A modelagem do sistema foi criada e atualizada realizando sua consistência e corretude de forma manual. As classes do código foram criadas sem nenhuma regra ou padronização, cada pessoa da equipe de desenvolvimento poderia visualizar os diagramas e elementos da linguagem de modelagem do ANote de uma maneira diferente na hora de refletir o código.

Várias modificações e aperfeiçoamentos foram necessários para acertar o sistema. Não havia um ambiente de apoio, quer fosse visual para a modelagem ou de geração de código para a implementação. Ou seja, a fase de especificação foi realizada em papel e/ou em arquivos do Word (.doc); e o código sendo criado sem o apoio de uma ferramenta que verificasse sua aderência a modelagem.

## 5.1.
## O Learn Agents

O Learn Agents [12] é um sistema existente desenvolvido para participar do Trading Agent Competition (TAC) [25]. O TAC é um fórum internacional projetado para promover e incentivar pesquisa de qualidade no problema de negociação ente os agentes. O sistema multi-agente TAC 2004 opera em um cenário de turismo onde se comercializa bens para viagem. Os agentes de software são os agentes de viagem que compram e vendem bilhetes de avião, quartos de

hotel, e bilhetes de entretenimento para os clientes. As pontuações do TAC são baseadas nas preferências dos clientes por viagens, e nas despesas dos leilões de viagens.

No TAC, cada agente tem um objetivo de montar pacotes de viagem. Cada pacote é de TACtown a Tampa, por um período de 5 dias. Cada agente está agindo em nome de oito clientes, que expressam suas preferências por vários aspectos da viagem. O objetivo do agente de viagem é maximizar a satisfação total dos seus clientes, com uma despesa mínima nos leilões de viagens.

Uma execução do jogo é chamada de instância. Diversas instâncias são executadas durante cada rodada da competição a fim de avaliar o desempenho médio dos agentes e as variações aleatórias das preferências dos clientes. Cada instância dura nove minutos.

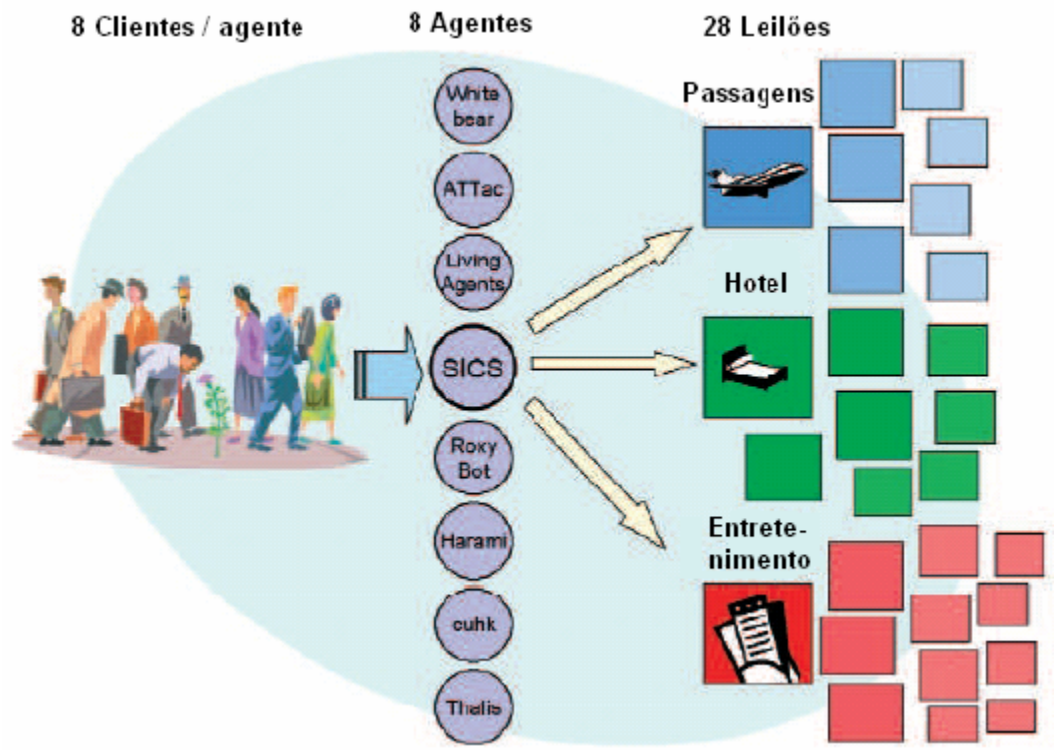


Figura 38 - TAC

Na competição de 2004, o LearnAgents terminou em terceiro lugar deixando pra trás grandes nomes. Este sistema foi desenvolvido com o uso da linguagem de modelagem ANote e arquitetura ASYNC. A agência de viagem é modelada como um sistema multi-agente que negocia nos leilões relacionados de uma instância. Os problemas de negociação foram identificados na visão de objetivos da linguagem de modelagem do ANote, como: calcular as melhores alocações, predizer os preços no leilão e calcular a segmentação da demanda.

Cada agente no Learn Agents é responsável por um ou mais destes subproblemas de negociação.

### 5.1.1. Diagrama de Objetivos

O LearnAgents tem como objetivo principal adquirir pacotes de viagens que minimizem a despesa líquida dos clientes. Adquirir pacotes de viagens envolve construir uma base de conhecimento para manipular os dados do mercado e negociar os pacotes de viagens. Os objetivos funcionais são identificados pelas informações necessárias para monitorar o mercado, classificar as preferências do cliente, predizer o próximo preço dos leilões e calcular melhores alocações que serão incluídas na base de conhecimento; e ao classificar as melhores alocações, criar ordens de ofertas e enviar ofertas nos leilões para a negociação dos pacotes de viagens.



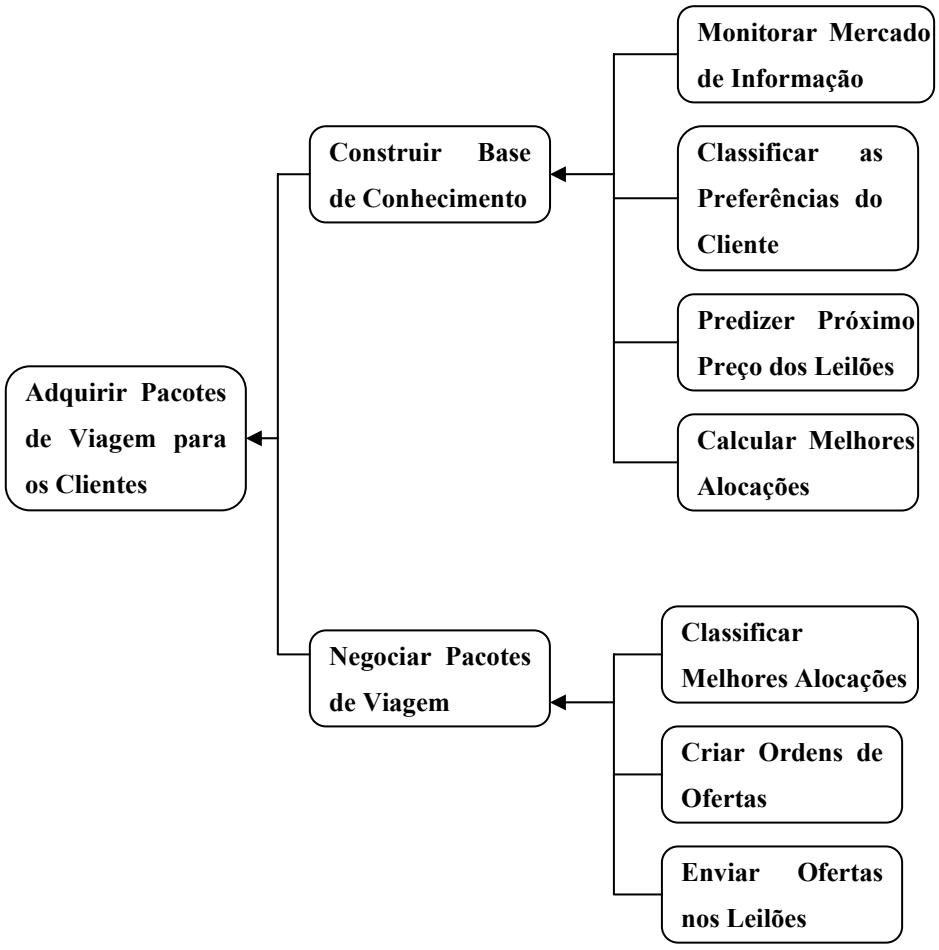


Figura 39 – Diagrama de Objetivos do LearnAgents desenvolvido sem Albatroz

### 5.1.2.
### Diagrama de Classes de Agentes

Cada objetivo funcional modelado no diagrama de Objetivos é conduzido por um ou mais agente no LearnAgents. Estes objetivos afetam também o ambiente modelado no Diagrama de Ontologias pelos agentes. Na figura abaixo, apresentamos o Diagrama de Agentes do LearnAgents.

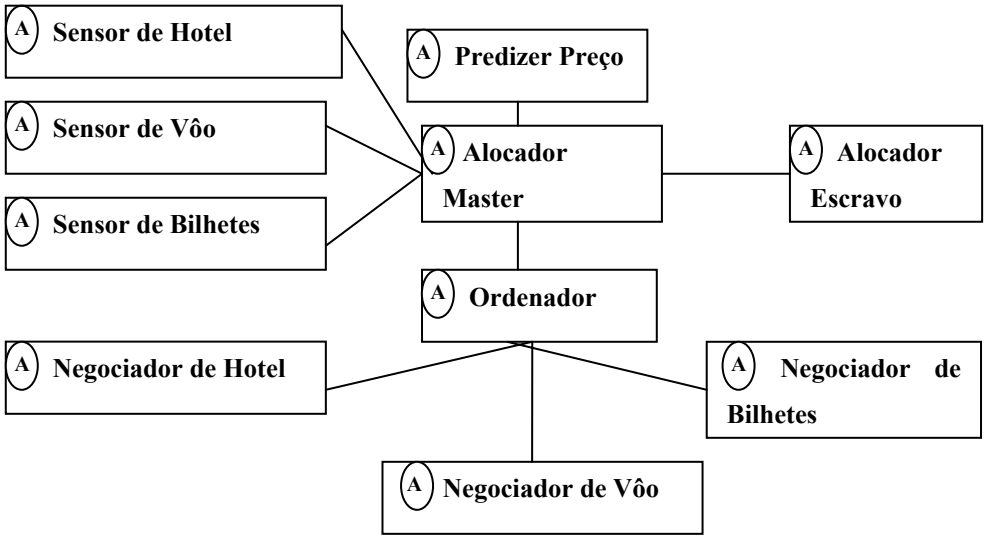


Figura 40 – Diagrama de Classe dos Agentes do LearnAgents desenvolvido sem Albatroz

### 5.1.3.
### Diagrama de Ontologias

O ambiente do TAC é composto por uma instância do jogo e por diversos eventos que são emitidos à agência através de uma infra-estrutura de comunicação. Os leilões dos vôos, dos quartos de hotel e dos bilhetes de entretenimento são parte desta instância. Quando a instância do jogo começa, as preferências do cliente estão emitidas para a agência e armazenadas na tabela de demanda. Assim, a agência pode construir uma estratégia para obter o melhor pacote de viagem com a despesa líquida mínima. Na figura abaixo, é apresentado o diagrama de classe das entidades do diagrama de Ontologias.

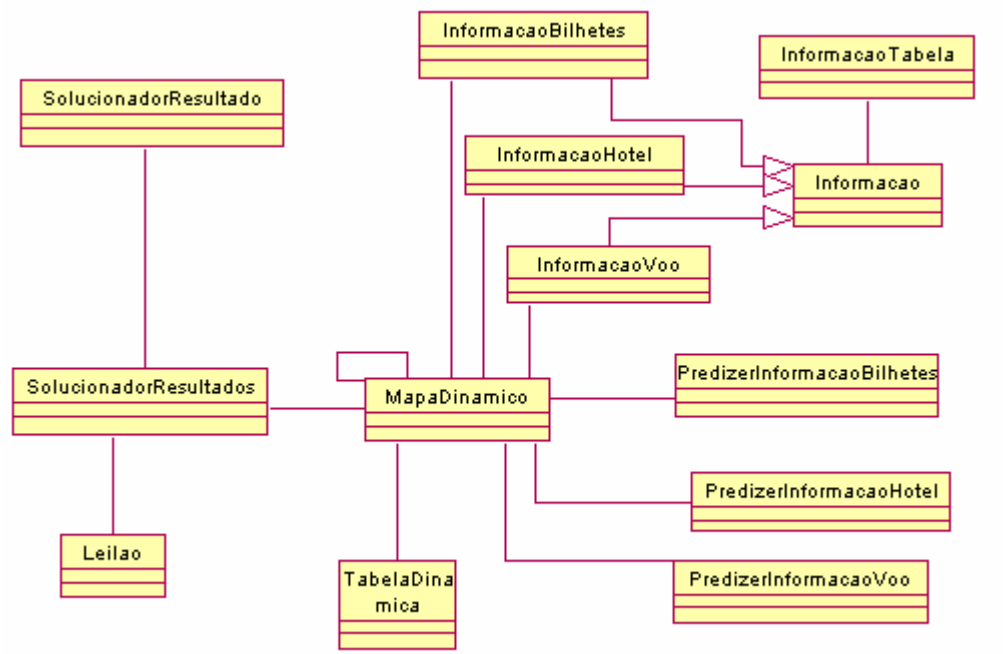


Figura 41 – Diagrama de Ontologias do LearnAgents desenvolvido sem Albatroz



### 5.1.4. Diagrama de Cenários

O Diagrama de Cenários especifica as descrições comportamentais de um ou mais agentes. Por simplificação, neste documento só será mostrado o exemplo do cenário *Predizer Preços* (Figura 42).

| Predizer Preços | |
|---|---|
| Main Agent | Agente Predizer Preço |
| Preconditions | Mensagem para os agentes Sensores |
| Main Action Plan | ENQUANTO (true)<br>Colotar o preço corrente em CKB<br>Predizer preço do leilão<br>Atualizar CKB<br>Enviar mensagem para o agente Alocador Master<br>FIM ENQUANTO |
| Interaction | Agente Alocador Master |
| Variants | |

Figura 42 – Cenário do LearnAgents desenvolvido sem Albatroz

Neste diagrama, especificam-se os preços de vôos e hotéis através de uma série histórica, os agentes *Predizer Preço* são capazes de prever para que valor o preço do leilão deve caminhar.

Cada agente no sistema participa de ao menos um diagrama de Cenários, e todos estes cenários foram mapeados em métodos das subclasses de *Agent* na arquitetura ASYNC.

### 5.1.5.
### Diagrama de Planejamento

De acordo com o cenário *Predizer Preços* exemplificado no item 5.1.4, o plano principal das ações é demonstrado no diagrama de Planejamento (Figura 43).

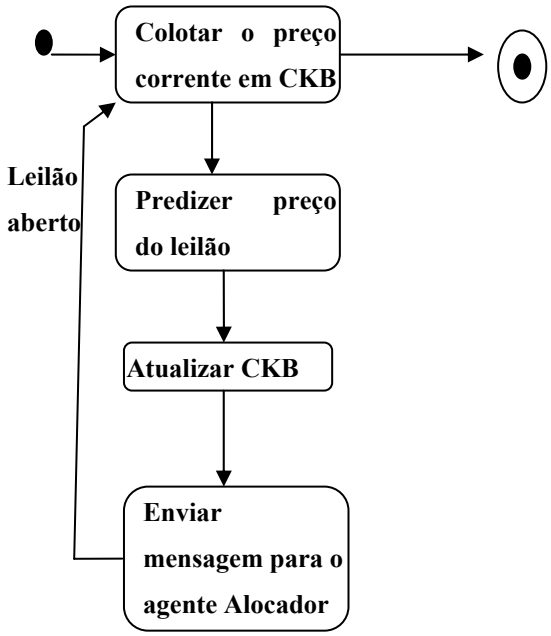


Figura 43 – Diagrama de Planejamento do LearnAgents desenvolvido sem Albatroz

### 5.1.6.
### Diagrama de Interação

A interação no sistema gera protocolos de mensagem que descrevem a dinâmica da comunicação dos agentes. Embora esta interação seja especificada no diagrama de Cenários, o diagrama de Interação claramente guia a especificação das interações na fase de implementação.

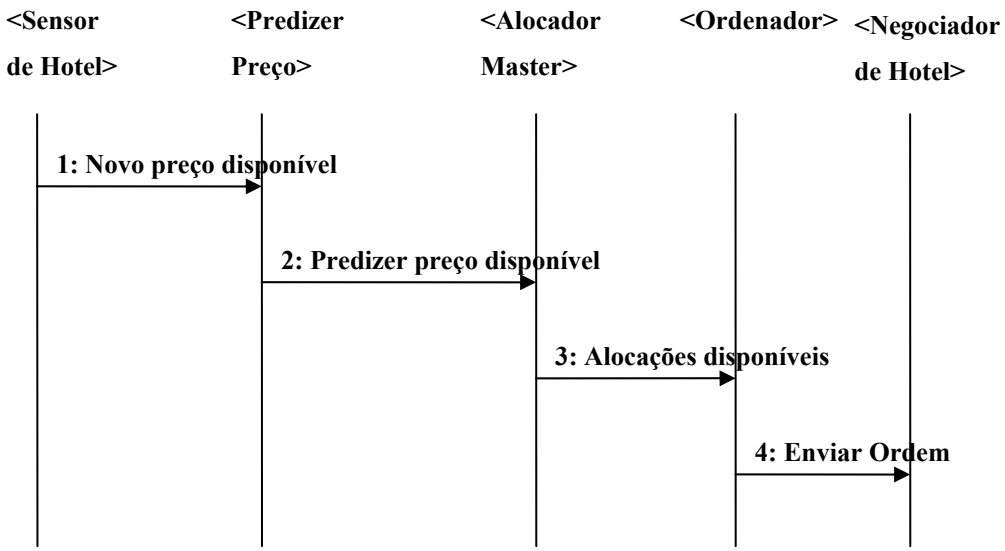


Figura 44 – Diagrama de Interação do LearnAgents desenvolvido sem Albatroz

Na figura acima, o *Sensor de Hotel* recebe um evento do ambiente com novas cotações dos leilões de hotéis. O *Sensor de Hotel* atualiza a base de conhecimento e envia uma mensagem para o *Predizer Preço* informando que os preços foram atualizados. O *Predizer Preço* calcula as novas previsões de preços dos leilões, e armazenas essas predições na base de conhecimento.

Os Alocadores calculam os diferentes cenários baseados nos preços correntes e previstos. O *Alocador Mestre* envia uma mensagem para o *Ordenador* após esse cálculo dos cenários.

O *Ordenador* recebe a mensagem e decide a quantidade necessária de bens para atender a demanda dos clientes. Além disso, o *Ordenador* calcula o valor máximo de um lance para cada bem a ser comprado. Os Negociadores recebem a mensagem do *Ordendador* com os pedidos de compra e o valor máximo do lance. Os Negociadores possuem o objetivo de comprar esses bens pelo menor preço possível, sem exceder um lance maior do que o máximo definido pelo Ordenador.

### 5.1.7. Diagrama de Organização

Neste estudo de caso, optou–se por se desenvolver apenas uma organização, uma vez que a arquitetura do ambiente de simulação do TAC é simples. A figura abaixo mostra o diagrama de Organização.

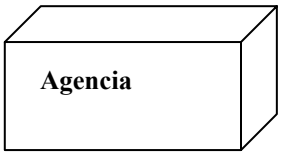


Figura 45 - Organização do LearnAgents desenvolvido sem Albatroz

### 5.1.8.
### A implementação do Sistema

O mapeamento do código dos diagramas do ANote para a arquitetura ASYNC é muito simples. Na figura abaixo, apresentamos as classes que implementam o código do agente *Predizer Preço (PredizerPreco* e *PredizerPrecoIP)*.

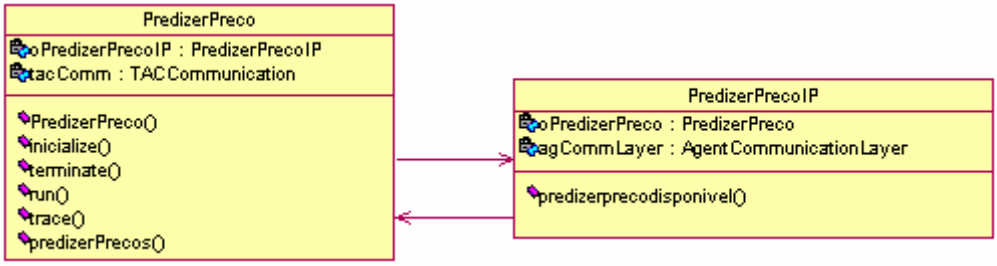


Figura 46 - Classes geradas a partir do agente PredizerPreco

O método *predizerprecodisponivel()* da classe *PredizerPrecoIP* tem o código que envia a mensagem 2: Predizer preço disponível da Figura 44. Neste método, o agente *Predizer Preço* informa que os preços foram atualizados. O *Predizer Preço* calcula as novas previsões de preços dos leilões, e armazenas essas predições na base de conhecimento.

O método *predizerPrecos0* tem o código responsável pelo processo do plano de ação do cenário *Predizer Preços.* Neste método, encontra implementado o fluxo de planejamento do cenário (Figura 43). E finalmente, cada entidade do diagrama de ontologias representa uma classe com seus respectivos atributos e métodos.

## 5.2.
## Usando o Albatroz

Com o uso do ambiente do Albatroz, utilizou-se as ferramentas criadas para ajudar nas fases de especificação e implementação do sistema Learn Agents. Apenas uma casca inicial do código é gerada, sendo responsabilidade do

desenvolvedor a implementação dos métodos (que estarão marcados) da infra-estrutura de comunicação e das regras de negócio.

### 5.2.1. Apoio Visual

Na fase de especificação do sistema utilizamos o plug-in ANote para modelagem e consistência dos elementos.

#### 5.2.1.1. Diagrama de Objetivo

A figura abaixo mostra o diagrama de Objetivos.

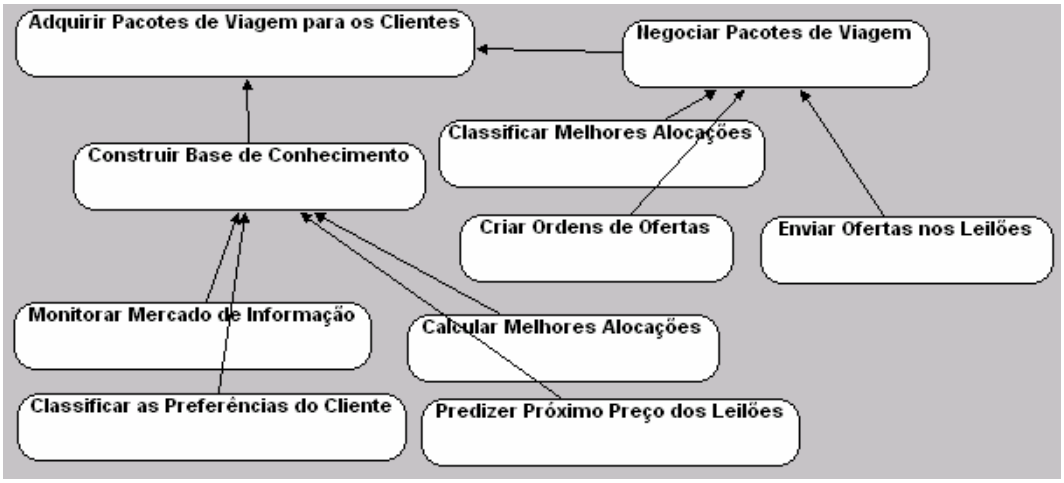


Figura 47 – Diagrama de Objetivos do LearnAgents desenvolvido com Albatroz

#### 5.2.1.2. Diagrama de Agentes

Cada agente existe dentro de uma organização, assim para criar um diagrama de Agentes é necessário criar uma organização (Figura 48)

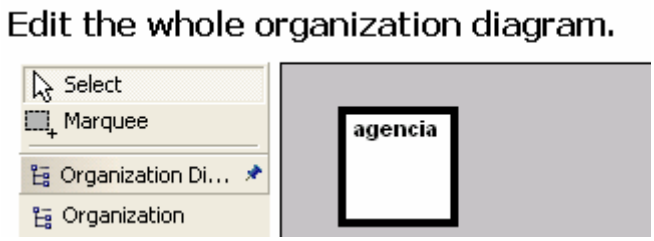


Figura 48 – Diagrama de Organizações do LearnAgents desenvolvido com Albatroz

O diagrama de Agentes para esta organização é mostrado na figura abaixo:

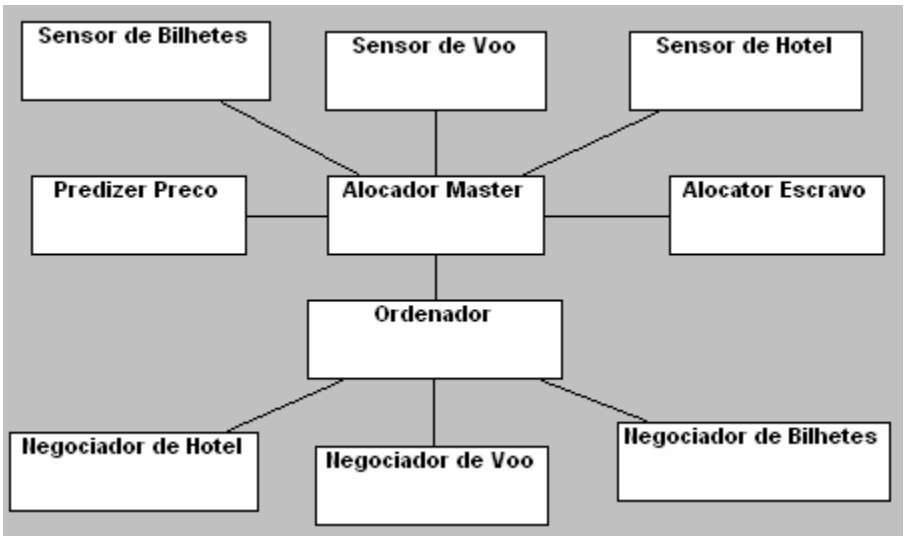


Figura 49 – Diagrama de Agentes do LearnAgents desenvolvido com Albatroz

### 5.2.1.3. Diagrama de Ontologias

A figura abaixo mostra o diagrama de Ontologias.

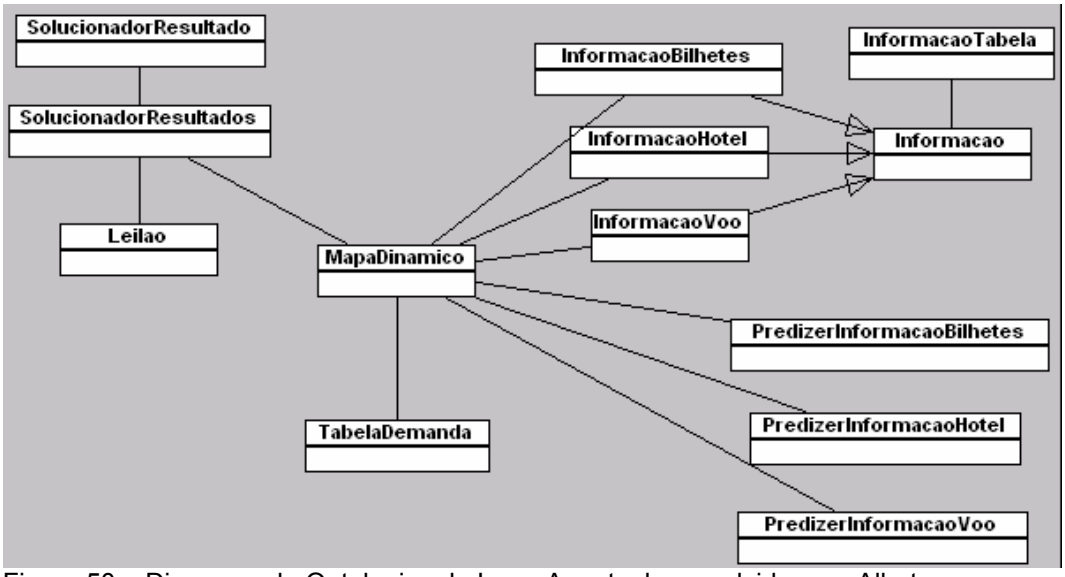


Figura 50 – Diagrama de Ontologias do LearnAgents desenvolvido com Albatroz

### 5.2.1.4. Diagrama de Cenários

A figura abaixo mostra o cenário *Predizer Preços* no diagrama de Cenários.

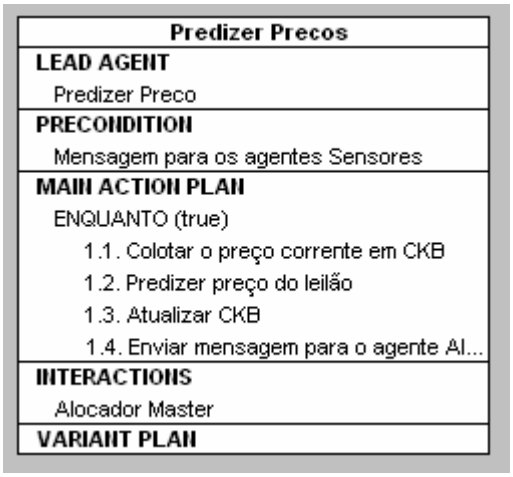


Figura 51 – Cenário do LearnAgents desenvolvido com Albatroz

Para fazer o plano de ação principal, utilizou-se o diagrama de Planejamento mostrado abaixo:

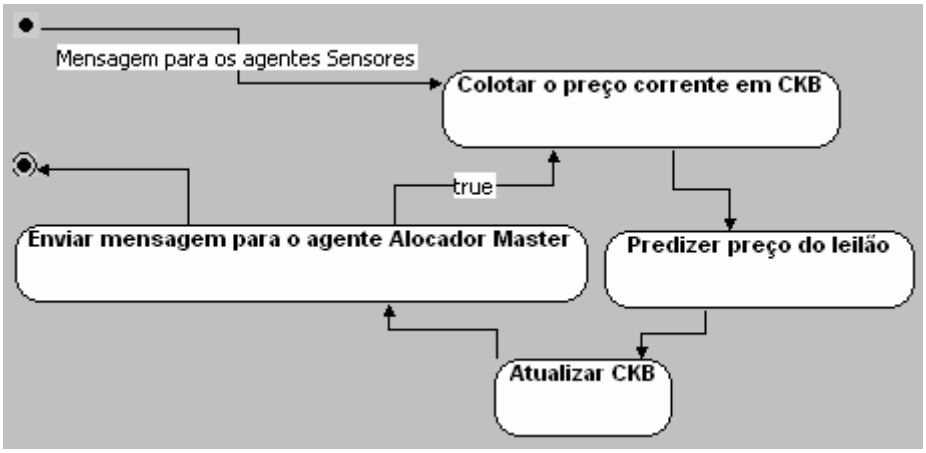


Figura 52 – Diagrama de Planejamento do LearnAgents desenvolvido com Albatroz

### 5.2.1.5. Diagrama de Interação

Sem a utilização do ambiente Albatroz, foi feito apenas um diagrama de Interação. Este diagrama representa de uma forma global e superficial um exemplo das comunicações entre os agentes. Pois nem todos os agentes estão presentes e se comunicando. Ao utilizar o Albatroz, cada diagrama de Interação pertence a um cenário. Ou seja, no cenário *Predizer Preços*, o agente *Predizer Precos* corresponde ao agente principal que interage com o agente *Alocador Master*. Assim, apenas a comunicação entre esses dois agentes será mostrada no diagrama de Interação deste cenário (Figura 53).

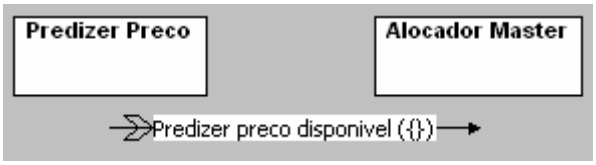


Figura 53 – Diagrama de Interação do LearnAgents desenvolvido com Albatroz

As outras comunicações encontram-se nos diagramas de Interação dos outros cenários. Desta forma, foi constata uma inconsistência entre os diagramas de Agentes e Iteração sem o Albatroz. Pelo diagrama de Interação (Figura 40), existe a necessidade de comunicação entre os agentes *Sensor de Hotel* e *Predizer Preço*, mas que só poderá ocorrer através do agente *Alocador Master* como mostrado no diagrama de Agentes (Figura 44).

### 5.2.1.6. Estrutura intermediária

Como resultado da especificação, a ferramenta gera um arquivo XMI baseado no meta-modelo da linguagem de modelagem ANote (ver Anexo B). Parte do modelo gerado referente aos objetivos é mostrado na figura abaixo:

```
<?xml version="1.0" encoding="UTF-8" ?>
- <anote.plugin.model:ANotePluginModel xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI"
    xmlns:anote.plugin.model="http:///anote/plugin/model.ecore">
  - <goalDiagram id="a200502160810200" name="GoalDiagram">
      <goals id="a200502160811034" name="Adquirir Pacotes de Viagem para os Clientes" x="8"
        y="6" width="264" height="40" targetGeneralizations="a2005021608153714
        a2005021608154818" />
      <goals id="a200502160811155" name="Construir Base de Conhecimento" x="41" y="82"
        width="203" height="40" targetGeneralizations="a2005021608154919 a2005021608155120
        a2005021608155221 a2005021608155522" sourceGeneralization="a2005021608154818" />
      <goals id="a200502160811356" name="Negociar Pacotes de Viagem" x="355" y="12"
        width="177" height="40" targetGeneralizations="a2005021608154015 a2005021608154216
        a2005021608154317" sourceGeneralization="a2005021608153714" />
      <goals id="a200502160811527" name="Monitorar Mercado de Informação" x="7" y="173"
        width="206" height="40" sourceGeneralization="a2005021608154919" />
      <goals id="a200502160812008" name="Classificar as Preferências do Cliente" x="9" y="225"
        width="224" height="40" sourceGeneralization="a2005021608155120" />
      <goals id="a200502160812149" name="Predizer Próximo Preço dos Leilões" x="240" y="228"
        width="218" height="40" sourceGeneralization="a2005021608155221" />
      <goals id="a2005021608123110" name="Calcular Melhores Alocações" x="232" y="180"
        width="180" height="40" sourceGeneralization="a2005021608155522" />
      <goals id="a2005021608125211" name="Classificar Melhores Alocações" x="250" y="69"
        width="188" height="40" sourceGeneralization="a2005021608154015" />
      <goals id="a2005021608130312" name="Criar Ordens de Ofertas" x="262" y="123"
```

Figura 54 – Estrutura intermediária do ANote para o Estudo de Caso

### 5.2.2. Transformação para ASYNC

A partir da estrutura intermediária gerada pela ferramenta ANote, é gerada uma outra estrutura intermediária baseada nas regras pré-definidas presentes no arquivo de configuração. De acordo com as regras do ASYNC descritas no

capítulo 4.3, ocorre a transformação da especificação do sistema baseado na linguagem de modelagem ANote para a arquitetura ASYNC.

A chamada para este ferramenta ocorre diretamente pela representação da instância do modelo criado do ANote, como mostrado na figura abaixo.

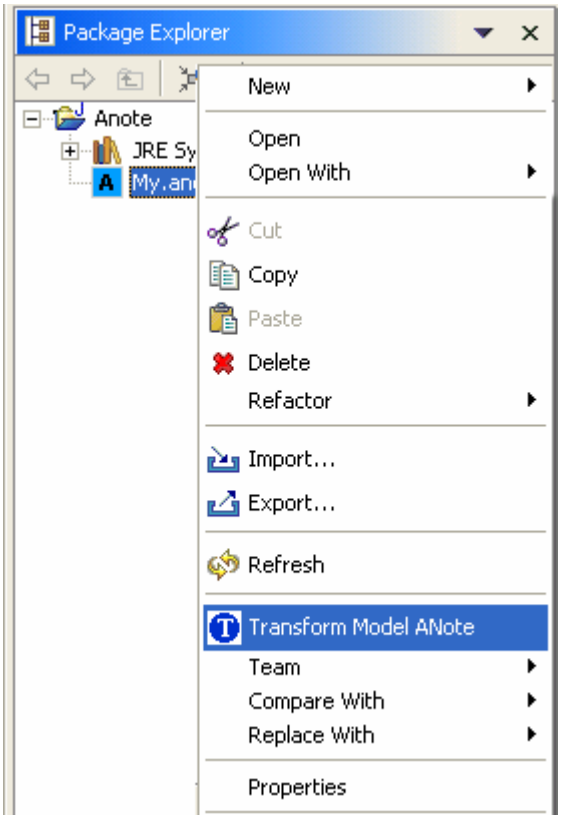


Figura 55 – Chamada para o ANote parte 1

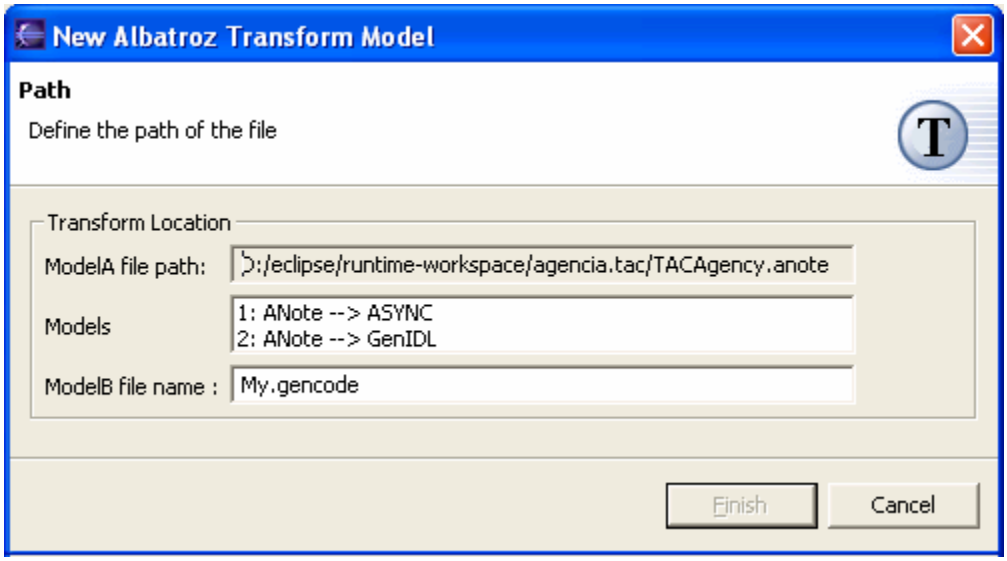


Figura 56 – Chamada para o ANote parte 2

### 5.2.2.1. Regra 1

Para cada agente (elemento do tipo *organizationDiagram.organizations. agentDiagram.agents)* encontrado na modelagem do ANote, são criadas duas classes pertencentes ao framework ASYNC de acordo com o atributo *elements*. Uma, com o mesmo nome do agente estendendo a classe *AgentI* e implementando a interface *AgentInterface,* e a outra, com o mesmo nome adicionado da constante "IP", implementando a interface *InteractionProtocol*. As duas classes novas são colocadas num pacote que terá o nome da organização do agente adicionado do nome do agente.

Atributos e métodos específicos das classes *Agent, AgentInterface* e *InteractionProtocol*, para cada classe nova são mapeados também. Como mostrado na figura abaixo:

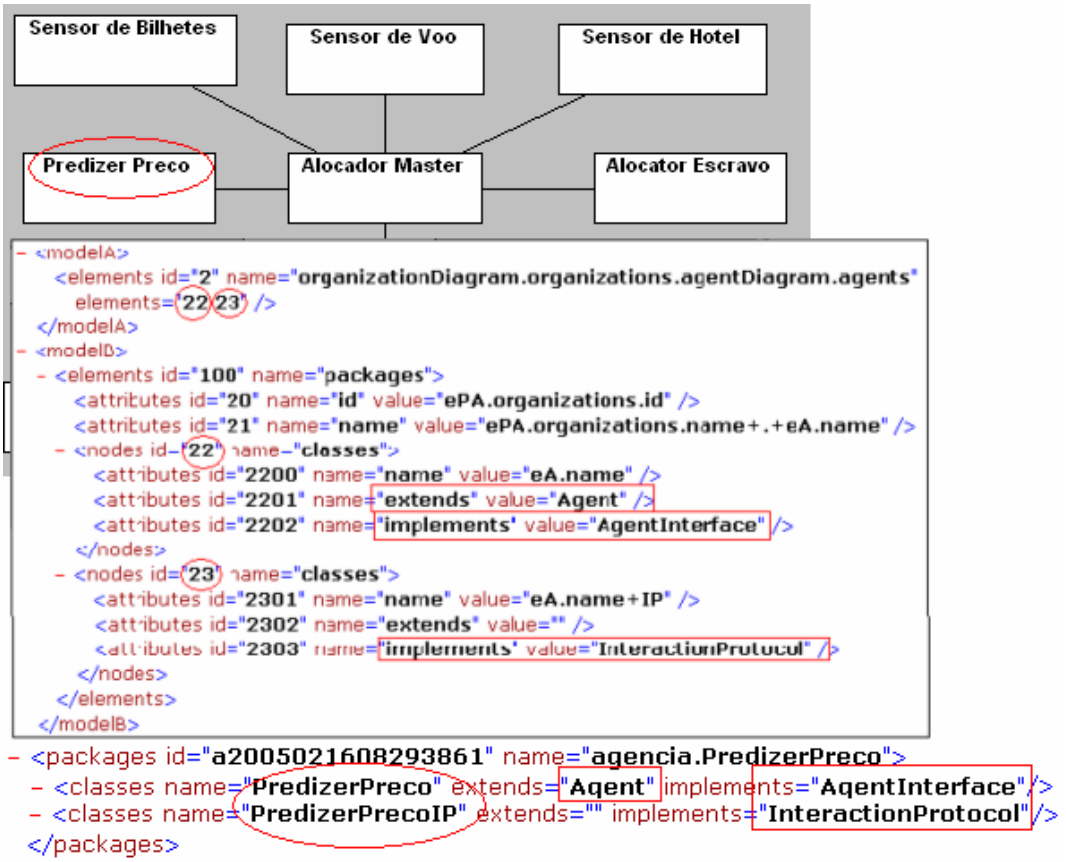


Figura 57 – Regra 1

O *Predizer Preco* encontrado na modelagem do ANote, é do mesmo elemento do tipo *organizationDiagram.organizations.agentDiagramagents.* Assim, serão geradas duas classes que pertencem ao mesmo pacote. Uma classe com o atributo *name* e com valor *eA.name* (significando o nome do agente em

questão); e a outra classes com o atributo name e com valor *eA.name adicionado da cosntante* "IP". O pacote terá o atributo name com o valor *ePA.organization.name* (significando o nome da organização do agente) adicionado do *eA.name*. Na figura acima, não aparecem todos os atributos e métodos que deveriam ser criados para refletir a arquitetura ASYNC por completo.

### 5.2.2.2. Regra 2

A partir de cada cenário (elemento do tipo *scenarioDiagram.scenarios)* encontrados na modelagem do ANote, é criado um método para o leadAgent relacionado com este cenário. Este método é criado dentro da classe criada pela regra anterior que estendem a classe *Agent* pertencente ao framework ASYNC, de acordo com o atributo *elements*. O nome do método será igual ao nome do cenário, o tipo de retorno do método será *void* e possuirá parâmetros nulos. Os agentes relacionados estão definidos através do atributo *references*. Como mostrado na figura abaixo:

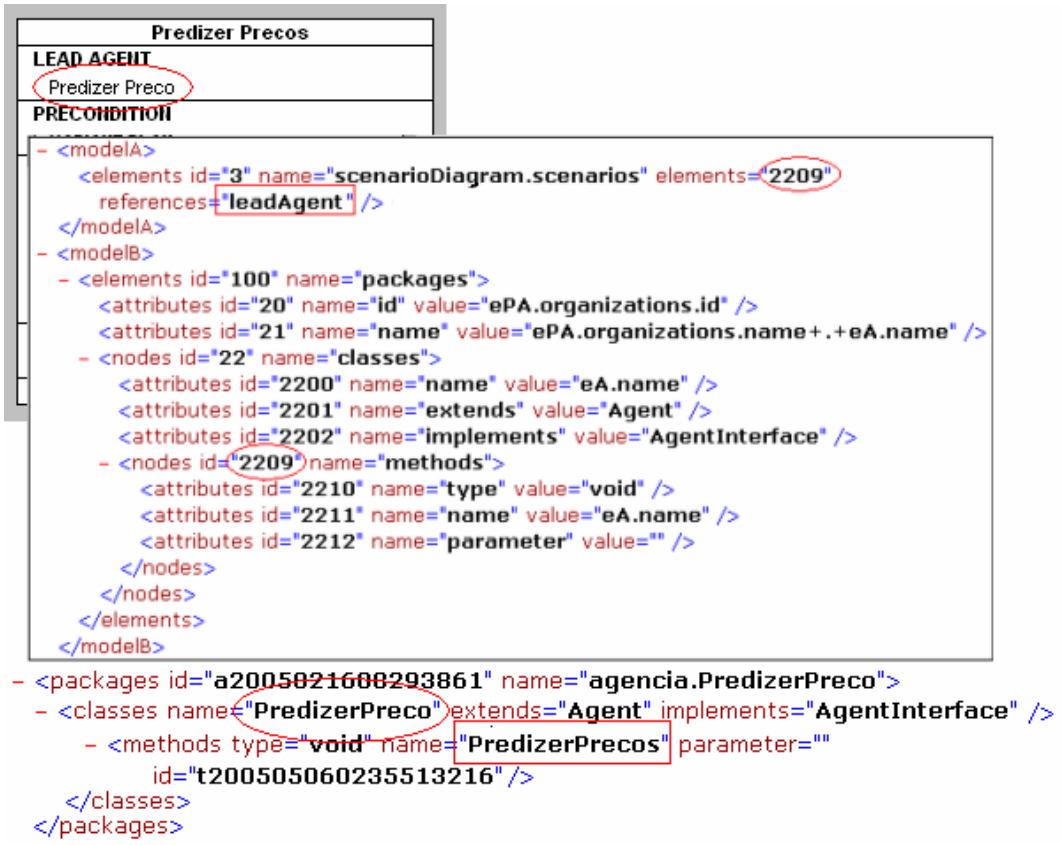


Figura 58 – Regra 2

O agente *Predizer Preço* (leadAgent) encontrado na modelagem do ANote é do mesmo elemento do tipo scenario*Diagram.scenarios.* Assim, será gerado um método que terá o atributo *type* com valor *void*, o atributo *name* com valor *eA.name* (significa do nome do cenário em questão) e o atributo *parameter* com valor vazio. Na figura acima, isto é exemplificado apenas para o agente principal deste cenário.

#### 5.2.2.3. Regra 3

A partir dos planos de ação de cada cenário (elemento do tipo *scenarioDiagram.scenarios.planningDiagram.transitions e scenarioDiagram. scenarios.planningDiagram.states)* encontrados na modelagem do ANote, são criados comandos dentro do método criado pela regra anterior com o nome do estado e um outro comando dentro do método *run()* desta classe. Este método é criado dentro da classe criada pela regra anterior que estendem a classe *Agent* pertencente ao framework ASYNC, de acordo com o atributo *elements*. O nome do método será igual ao nome do cenário. Os agentes relacionados estão definidos através do atributo *references*. Como mostrado na figura abaixo:

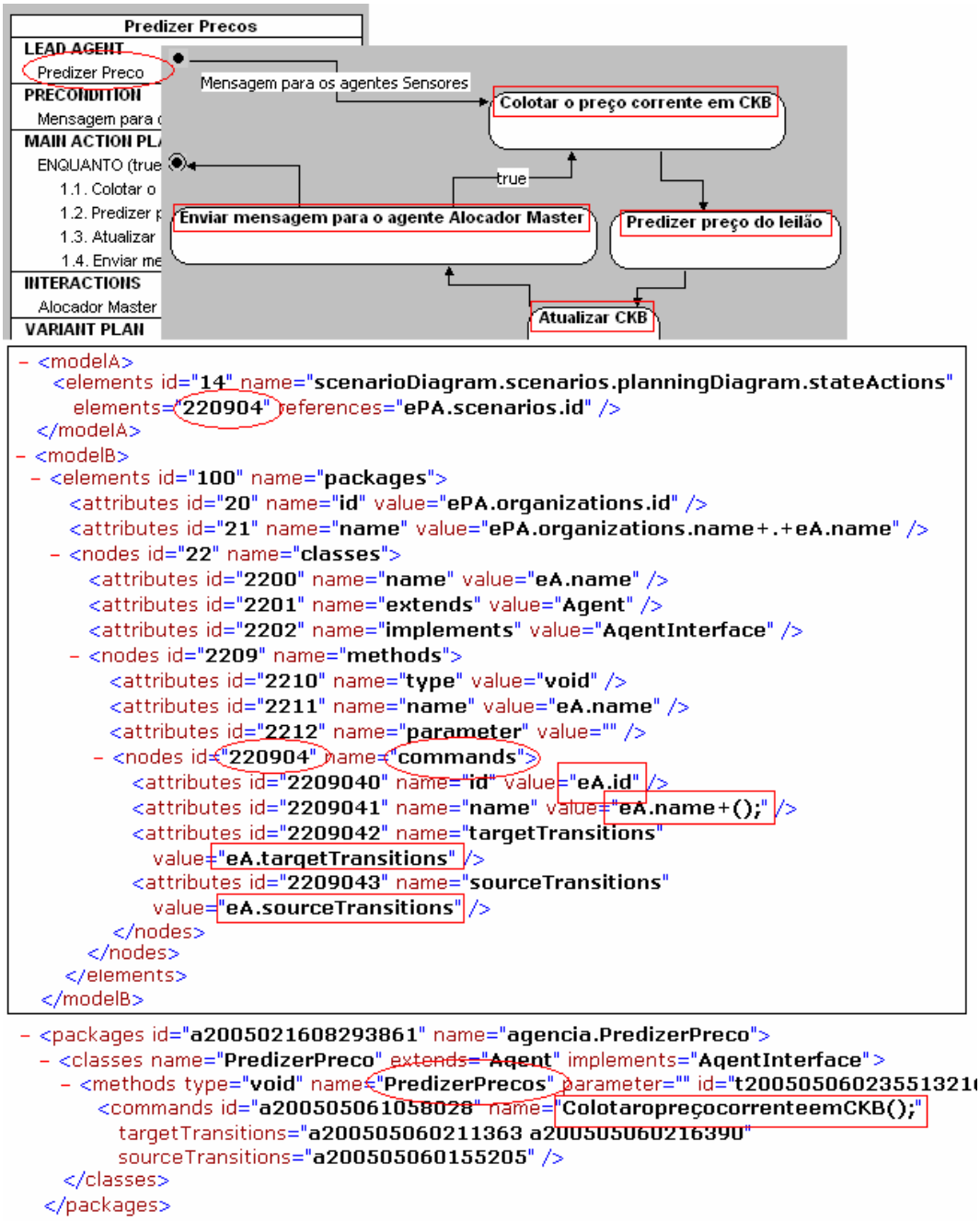


Figura 59 – Regra 3

O agente *Predizer Preço* (leadAgent) encontrado na modelagem do ANote é do mesmo elemento do tipo scenario *Diagram.scenarios.planningDiagram.states* Assim, são criados comandos dentro do método criado pela regra anterior com o nome do estado e um outro comando dentro do método *run()* desta classe. Na figura acima, exemplifica a criação do comando *ColetaropreçocorrenteemCKB* que será colocado dentro do método (com o nome do cenário) seguindo a orcdem do fluxo de ações mostrado no diagrama de planejamento deste cenário.

### 5.2.2.4. Regra 4

Para cada entidade (elemento do tipo *ontologyDiagram.entities)* encontrada na modelagem do ANote, é criada uma classe correspondente à entidade modelada com seus atributos e métodos respectivos, de acordo com o atributo *elements*. Esta classe será inserida no pacote *ontologies* pertencente ao framework ASYNC. Como mostrado na figura abaixo:

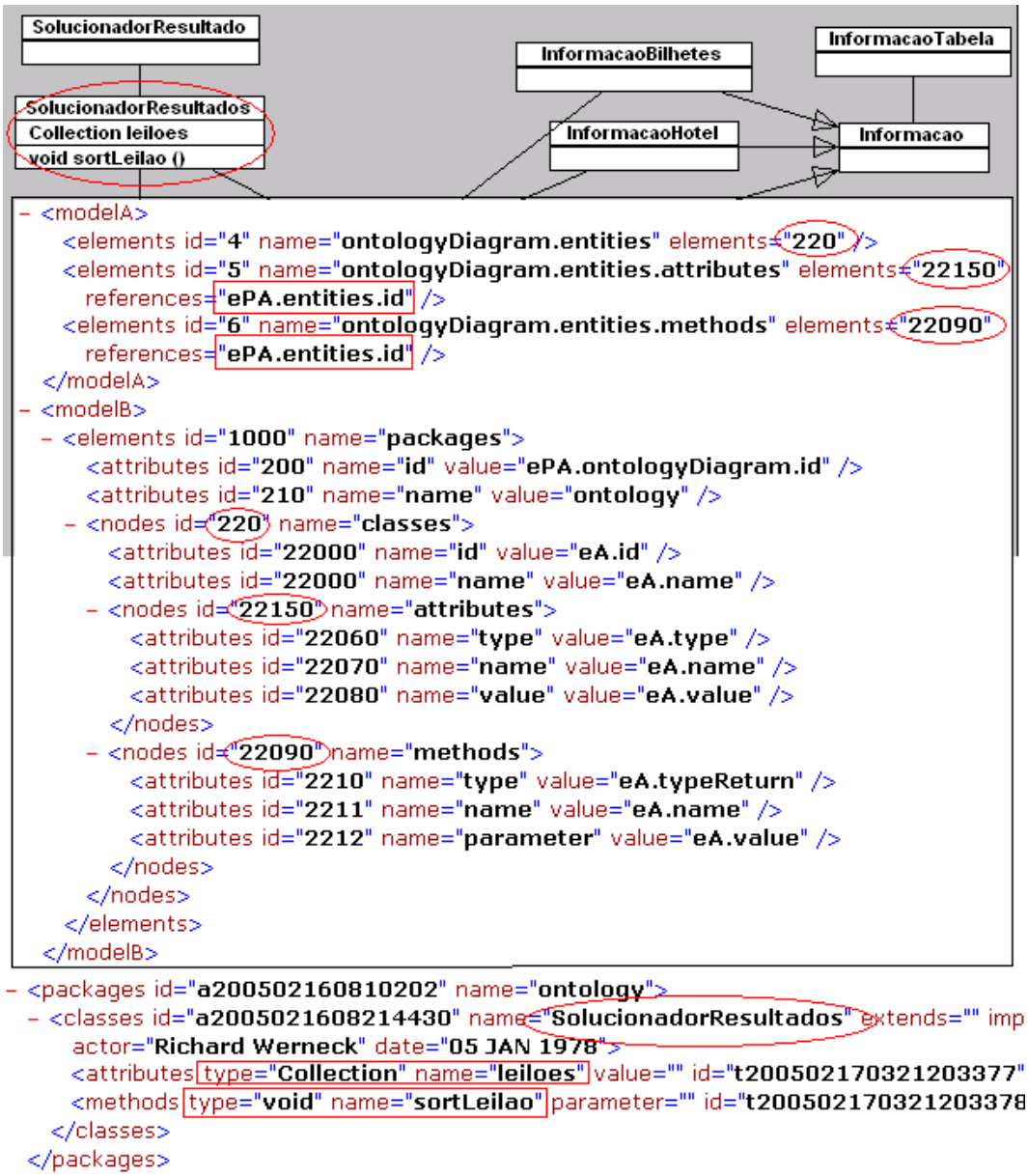


Figura 60 – Regra 4

A entidade *Solucionador Resultados* encontrada na modelagem do ANote, é do mesmo elemento do tipo *ontologyDiagram.entities*. Assim, será gerada uma classe no pacote ontologias e seus respectivos atributos e métodos. Esta classe terá

o atributo *id* com valor *eA.id* (significando o id da entidade em questão) e o atributo *name* com valor *eA.name* (significando nome da entidade em questão). Na figura acima, é exemplificado apenas para uma entidade do Diagrama de Ontologias

### 5.2.2.5. Regra 5

Para cada mensagem (elemento do tipo *scenarioDiagram.scenarios. interactionDiagram.messages)* encontrada na modelagem do ANote, é criado um método para o agente que enviou esta mensagem. Este método é criado dentro das classes criadas anteriormente que implementam a classe *InteractionProtocol* pertencente ao framework ASYNC. O nome do método será igual ao nome da mensagem, o tipo de retorno do método será *void* e possuirá os mesmos parâmetros da mensagem. Como mostrado na figura abaixo:


Figura 61 – Regra 5

A mensagem *Predizer preços disponível* encontrada na modelagem do ANote, é do mesmo elemento do tipo *scenarioDiagram.scenarios.interactionDiagram.messages*. Assim, será gerado um método na classe que implementa a interface *InteractionProtocol* do agente que envia esta mensagem. Este método terá o atributo *type* com valor *void*, o atributo *name* com valor *eA.name* (significando o nome da mensagem em questão) e o atributo *parameter* com valor eA.entities (significando as entidades relacionadas com a mensagem em questão). Na figura acima, isto é exemplificado apenas para uma mensagem de um determinado cenário.

### 5.2.2.6. Estrutura intermediária

Como resultado da transformação, a ferramenta gera um arquivo XML baseado no meta-modelo da linguagem de programação JAVA (ver Anexo B) como descrito no capítulo 4.4. Este arquivo XML gerado servirá de entrada para a ferramenta de geração de código (Generator). Parte do modelo gerado é mostrado na figura abaixo:

```
<?xml version="1.0" encoding="UTF-8" ?>
- <generator.plugin.model:GeneratorPluginModel xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI
    xmlns:generator.plugin.model="http:///generator/plugin/model.ecore">
  - <packages id="a2005021608293861" name="agencia.AgenteSensordeHotel">
    - <classes name="AgenteSensordeHotel" extends="Agent" implements="AgentInterface"
        actor="Richard Werneck" date="05 JAN 1978" id="t200502170321203189">
        <attributes type="boolean" name="stopped" value="false" id="t200502170321203190" />
        <attributes type="boolean" name="waitMsg" value="false" id="t200502170321203191" />
        <attributes type="boolean" name="broadcastRegistered" value="false"
          id="t200502170321203192" />
        <attributes type="AgenteSensordeHotelIP" name="oAgenteSensordeHotelIP" value="null"
          id="t200502170321203193" />
        <methods type="void" name="inicialize" parameter="" id="t200502170321203194" />
        <methods type="void" name="run" parameter="" id="t200502170321203195" />
        <methods type="void" name="terminate" parameter="" id="t200502170321203196" />
        <methods type="void" name="trace" parameter="String msg" id="t200502170321203197" />
      </classes>
    - <classes name="AgenteSensordeHotelIP" extends="" implements="InteractionProtocol"
        actor="Richard Werneck" date="05 JAN 1978" id="t200502170321203198">
        <attributes type="AgenteSensordeHotel" name="oAgenteSensordeHotel" value="null"
          id="t200502170321203199" />
        <attributes type="AgentCommunication" name="agCommLayer" value="null"
          id="t200502170321203200" />
        <methods type="void" name="requestRegistration" parameter=""
          id="t200502170321203201" />
```

Figura 62 – Estrutura intermediária gerada pelo Transform para Estudo de Caso

### 5.2.3. Geração Parcial

A chamada para esta ferramenta ocorre diretamente pela representação do modelo gerado pelo Transform Plug-in, como mostrado na figura abaixo.

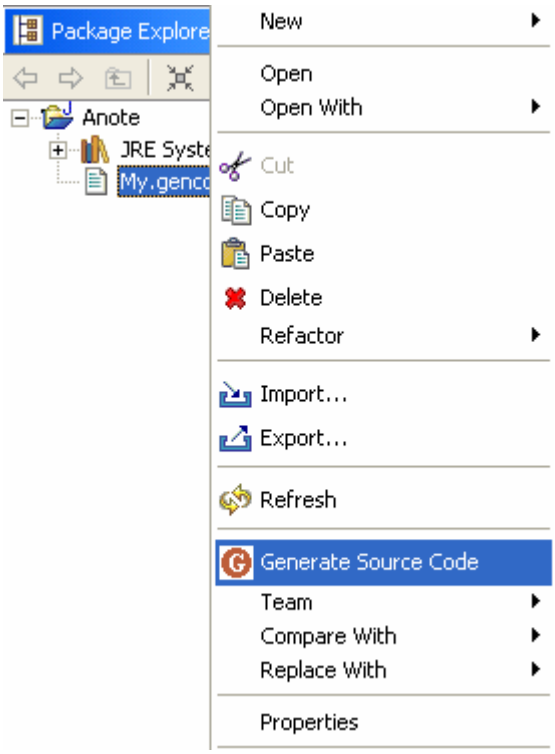


Figura 63 – Chamada para o Generator parte 1

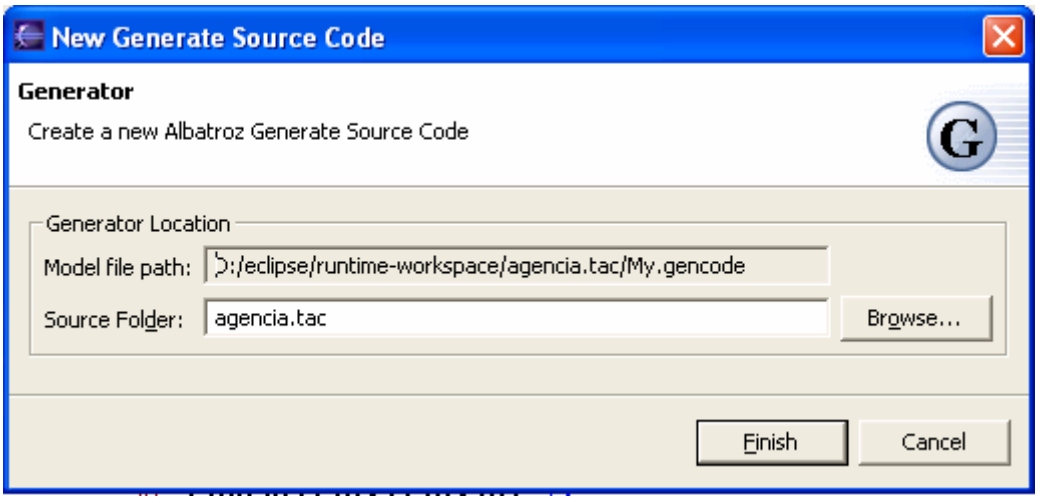


Figura 64 – Chamada para o Generator parte 2

A partir da estrutura intermediária gerada pelo Transform, é possível realizar a geração do código para a arquitetura ASYNC. Neste momento, não importa se a transformação foi feita para o ASYNC ou outra arquitetura qualquer. O plug-in Generator irá interpretar o modelo de entrada e gerar o código desejado (ver Anexo C), como descrito no capítulo 4.4. A figura abaixo mostra os pacotes e algumas classes geradas:

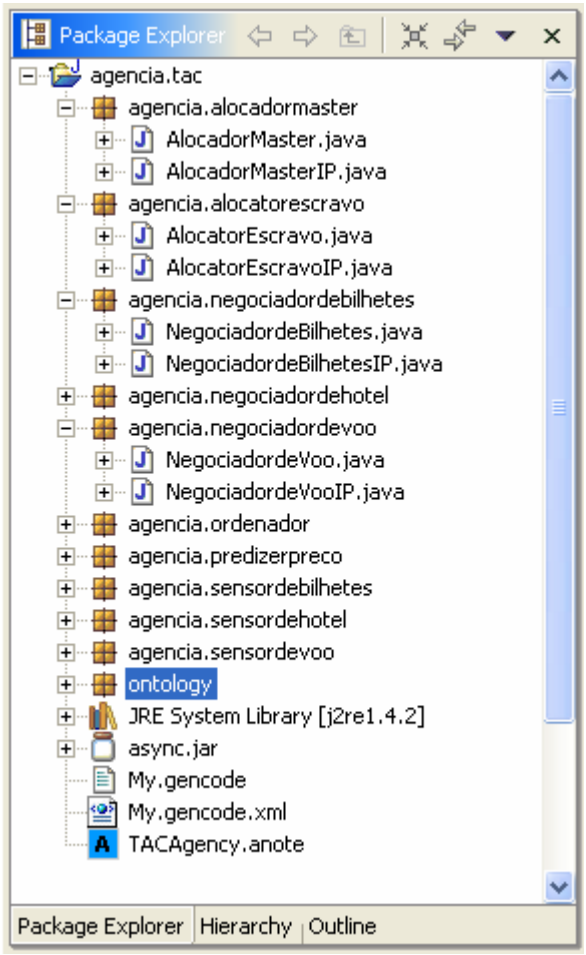


Figura 65 – Estrutura de pacotes e suas classes geradas pelo Generator

```
- <packages id="a2005021608293861" name="agencia.AgenteSensordeHotel">
  - <classes name="AgenteSensordeHotel" extends="Agent" implements="AgentInterface"
      actor="Richard Werneck" date="05 JAN 1978" id="t200502170321203189">
      <attributes type="boolean" name="stopped" value="false" id="t200502170321203190"
      <attributes type="boolean" name="waitMsg" value="false" id="t200502170321203191"
      <attributes type="boolean" name="broadcastRegistered" value="false"
        id="t200502170321203192" />
      <attributes type="AgenteSensordeHotelIP" name="oAgenteSensordeHotelIP" value="
        id="t200502170321203193" />
      <methods type="void" name="inicialize" parameter="" id="t200502170321203194" />
      <methods type="void" name="run" parameter="" id="t200502170321203195" />
      <methods type="void" name="terminate" parameter="" id="t200502170321203196" />
      <methods type="void" name="trace" parameter="String msg" id="t20050217032120319
    </classes>
```

```
package agencia.agentesensordehotel;

public class AgenteSensordeHotel
extends Agent implements AgentInterface {
    //Attributes
    private boolean stopped  = false;
    private boolean waitmsg  = false;
    private boolean broadcastregistered  = false;
    private AgenteSensordeHotelIP oagentesensordehotelip  = null;

    // Construtor
    public AgenteSensordeHotel ()
    {
        // TODO :
    }

    // Methods
    public void inicialize (){
        // TODO :
    }
    public void run (){
        // TODO :
    }
    public void terminate (){
        // TODO :
    }
```

Figura 66 - Geração de Código

Como pode ser observado pela figura acima, a classe *SensordeHotel* foi gerada de acordo com as informações disponibilizadas no arquivo de entrada. Após a construção do sistema utilizando o Albatroz, constatou-se que o código gerado não representa o código completo para execução do sistema, representando apenas o passo inicial para o restante da implementação. Comparando as estruturas de diretórios geradas podemos verificar algumas diferenças:

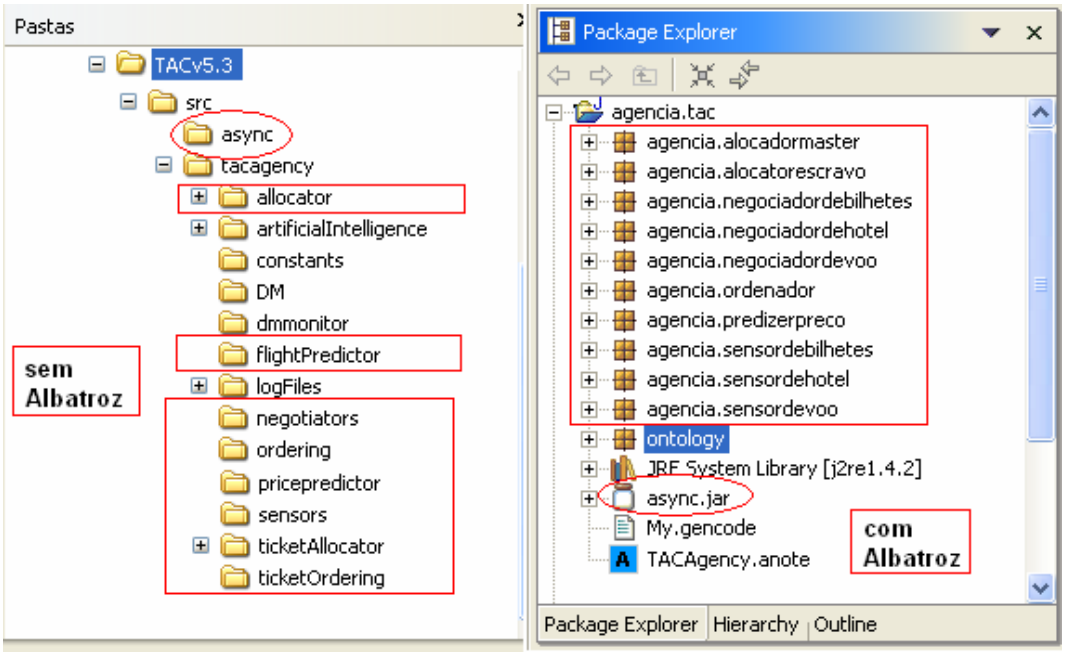


Figura 67 – Comparação de pacotes

| Sem Albatroz | Com Albatroz |
|---|---|
| as classes do framework ASYNC estão representadas como estrutura de diretórios. | as classes do framework ASYNC são adicionadas ao projeto na forma de arquivo .JAR, não são criadas pelo plug-in. |
| os pacotes dos agentes possuem nomes aleatórios | Padronização dos nomes dos pacotes dos agentes, organizando e facilitando futuras manutenções. |
| as classes que não são consideradas agentes ficam distribuídas pelos pacotes aleatoriamente | as classes que não são consideradas agentes são armazenadas no pacote *ontology* |

Tabela 1 – Comparação do Sistema

Os pacotes restantes que não são encontrados na geração do código utilizando o Albatroz (como artificialIntelligence, constants, logFiles,...), correspondem as classes que foram necessárias durante o processo da fase de implementação. Tais classes, não possuem relação direta com a especificação do sistema.

Outra comparação importante esta relacionada com a Regra 5 do item 5.2.2.5. Observando-se o código sem o Albatroz, podemos constatar a presença de vários métodos que foram criados, durante a fase de implementação, de acordo com a necessidade. Por exemplo, código da classe PredizerPreco do LearnAgents desenvolvido sem Albatroz:

```
1.  package tacagency.pricepredictor;
2.  public class PredizerPreco extends Agent implements AgentInterface {
3.  private TACCommunication tacComm;
4.  private float lastAskPrice;
5.  private double initialFlightAskPRice;
6.  public ExponencialSuavizada predicaoPrimeiroMinutoHotel;
7.  public MediaMovelDeslocada predicaoPrimeiroMinutoMediaMovelDeslocada;
8.  public LeastMeanSquares predicaoTerceiroMinutoHotel;
9.  public MediaMovelComCorte mediaMovelComCorteTerceiroMinuto;
10. public boolean firstTime;
11. private static final Logger log = Logger.getLogger( Constants.PRICE_PREDICTOR);

12. public PricePredictorAgent( String _name,
        i.  PricePredictorAgentIP _pricePredictorAgentIP,
        ii. TACCommunication _agTACComm){
        iii. super( _name, _pricePredictorAgentIP );
        iv. ...
13. }
14. public void initialize(){
        i.  ...
15. }
16. public void terminate() {
        i.  ...
17. }
18. public void run() {
        i.  ...
19. }
20. public void trace( String msg ) {
        i.  ...
21. }
22. public void initializePredictor(){
        i.  ...
23. }
24. public void updatePriceHistory(){
        i.  ...
25. }
26. public int getGame(){
        i.  return tacComm.getAgent().getGameID();
27. }
28. public long getTime(){
        i.  return tacComm.getAgent().getGameTime();
29. }
30. public float getLastAskPRice(){
        i.  return lastAskPrice;
31. }
32. public void sendDataVector(String data){
        i.  (( PricePredictorAgentIP ) getInteractionProtocols()).sendDataVector(dataVector);
33. }
34. /***INICIALIZACAO DOS PREDITORES*******/
35. private void initLastAskPrice() {
```

```
            i. ...
}
/***PREDITOR EXPONENCIAL DO TERCEIRO MINUTO DE HOTEL****/
private void initTerceiroMinutoHotel(){
            i. ...
}
/***PREDITOR EXPONENCIAL DO PRIMEIRO MINUTO DE HOTEL****/
/***************Exponencial*****************/
private void initExponencialPrimeiroMinutoHotel(){
            i. ...
}
/***Media Movel Deslocada*****/
private void initMediaMovelDeslocadaPrimeiroMinuto(){
            i. ...
}
/***UPDATE PREDICTORS****/
private void updateExponencialSuavizadaTerceiroMinuto(){
            i. ...
}
private void updatePredicaoPrimeiroMinuto(){
            i. ...
}
private void updatePredicaoPrimeiroMinutoMediaMovelDeslocada(){
            i. ...
}
private void insertLastAskPrice(Auction auction) {
            i. ...
}
public int getMinute(){
            i. ...
}
}
```

Código da classe PredizerPreco do LearnAgents desenvolvido com Albatroz:

```
1.  package agencia.predizerpreco;
2.  import async.Agent;
3.  import async.AgentInterface;
4.  /**
5.    * @author Richard Werneck
6.   * @date  05 JAN 1978
7.   **/
8.  public class PredizerPreco
9.   extends Agent
10.  implements AgentInterface
11. {
12.
13.       //Attributes
14.       private boolean stopped  = false;
15.       private boolean waitmsg  = false;
16.       private boolean broadcastregistered  = false;
17.       private PredizerPrecoIP opredizerprecoip  = null;
18.
19.       // Construtor
20.       public PredizerPreco ()
21.       {
22.                 //TODO
23.       }
24.
```

```
25.     // Methods
26.     public void inicialize (){
27.             // TODO : inicialize
28.     }
29.     public void run (){
30.             // TODO : run
31.                     PredizerPrecos();
32.     }
33.     public void terminate (){
34.             // TODO : terminate
35.     }
36.     public void trace (String msg){
37.             // TODO : trace
38.     }
39.     public void PredizerPrecos (){
40.             // TODO : PredizerPrecos
41.             if (true){//Mensagemparaosagentes Sensores){
42.             while (true) {
43.                 ColotaropreçocorrenteemCKB();
44.                 Predizerpreçodoleilão();
45.                 AtualizarCKB();
46.                 EnviarmensagemparaoagenteAlocadorMaster();
47.             }
48.             }
49.     }
50.     public void ColotaropreçocorrenteemCKB (){
51.             // TODO : ColotaropreçocorrenteemCKB
52.     }
53.     public void Predizerpreçodoleilão (){
54.             // TODO : Predizerpreçodoleilão
55.     }
56.     public void EnviarmensagemparaoagenteAlocadorMaster (){
57.             // TODO : EnviarmensagemparaoagenteAlocadorMaster
58.     }
59.     public void AtualizarCKB (){
60.             // TODO : AtualizarCKB
61.     }
62.     // Get and Set Attributes
63.     public boolean getStopped (){
64.             return stopped;
65.     }
66.     public void setStopped (boolean stopped){
67.             this.stopped = stopped;
68.     }
69.     public boolean getWaitMsg (){
70.             return waitmsg;
71.     }
72.     public void setWaitMsg (boolean waitmsg){
73.             this.waitmsg = waitmsg;
74.     }
75.     public boolean getBroadcastRegistered (){
76.             return broadcastregistered;
77.     }
78.     public void setBroadcastRegistered (boolean broadcastregistered){
79.             this.broadcastregistered = broadcastregistered;
80.     }
81.     public PredizerPrecoIP getOPredizerPrecoIP (){
82.             return opredizerprecoip;
83.     }
84.     public void setOPredizerPrecoIP (PredizerPrecoIP opredizerprecoip){
```

```
85.                 this.opredizerprecoip = opredizerprecoip;
86.         }
87. }
```

Comparando os dois código (com e sem o auxílio do Albatroz) são identificados (em negrito) alguns pontos em comum. Estes pontos em comum correspondem a prória estrutura do framework ASYNC, que é garantida pelas ferramentas Transform e Generator. Como o código foi gerado sem uma padronização definida e por desenvolvedores diferentes, fica muito difícil identificar o mapeamento com a modelagem pois até os nomes dos métodos foram alterados. Os pontos distintos identificados são métodos auxiliares que aparecem ao longo da implementação (possivelmente pelo uso de refactoring) ou são métodos que denotam planos, os quais deveriam aparecer na modelagem mas sem o uso de ferramentas esta consistência fica prejudicada.

Sem o auxílio de uma ferramenta que realize a funcionalidade de engenharia reversa, é extremamente difícil manter uma especificação do sistema refletindo exatamente com o código, ainda mais se tratando de um grupo grande de desenvolvedores. O ambiente Albatroz se preocupa apenas em gerar um código inicial legível e harmônico com a especificação do sistema para as implementações futuras.

## 5.3. Análise Crítica

Após a realização do estudo de caso, é possível constatar que o ambiente padroniza e acelera consideravelmente a construção da aplicação. Algumas vantagens foram verificadas, como:

• interligação das ferramentas, as estruturas intermediárias utilizadas e geradas pelas ferramentas em formato XMI tornam o ambiente extensível e interoperável.

• ajuda visual para modelagem do sistema facilitando o entendimento do problema e garantindo a consistência das informações.

• padronização através das regras estabelecidas no arquivo de configuração do Transform. Cada regra garante que a transformação de modelo para modelo ocorra natural e automaticamente, evitando erros cometidos manualmente.

• automatização do processo de geração parcial de código, o ambiente de desenvolvimento agiliza a construção da infra-estrutura inicial para o restante da implementação.

Enfim, tudo muito mais completo, rápido e com informações seguras.