# 4
# ALBATROZ : Um ambiente para desenvolvimento de SMA

## Resumo

*Neste capítulo será apresentado o processo de desenvolvimento do ambiente Albatroz. Cada ferramenta é detalhada indicando suas funcionalidades.*

O ambiente proposto neste trabalho é dividido em três ferramentas (Figura 9), distribuídas em dois níveis fundamentais: nível superior, com uma ferramenta de apoio visual responsável pela interface gráfica e nível inferior, com ferramentas de transformação de modelos XML e geração de código. Esta divisão conceitual permitiu o desenvolvimento paralelo do sistema, ou seja, cada ferramenta foi desenvolvida independentemente.

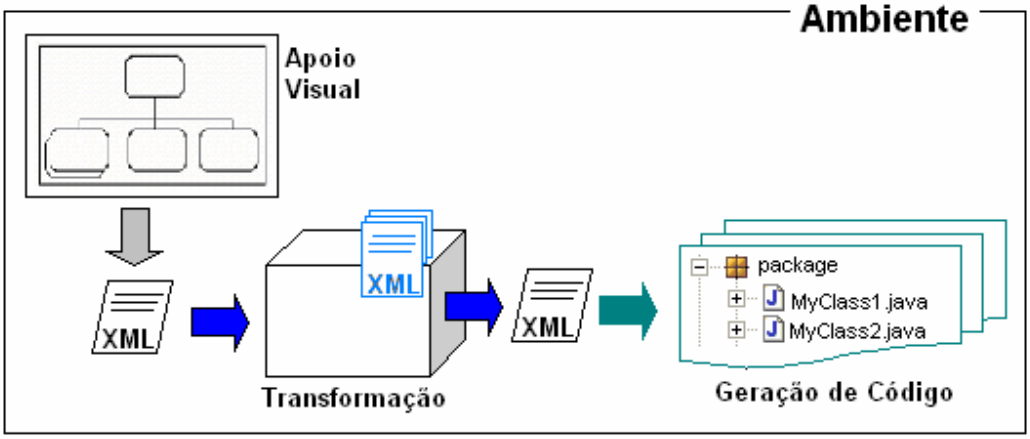


Figura 9 – Ambiente Albatroz

O projeto foi elaborado com o intuito de tornar o ambiente extensível e interoperável, permitindo que novas configurações possam ser feitas no plug-in de transformação para uma outra arquitetura de agentes, ou, até mesmo, para uma outra determinada linguagem de modelagem.

## 4.1. Albatroz

O ambiente proposto foi desenvolvido em Java, na forma de plug-in do Eclipse. Tanto a plataforma de desenvolvimento quanto a linguagem de

programação foram escolhidas por terem simples interação com a linguagem XML, e por permitirem a portabilidade do sistema.

A popularidade do XML provocou o aparecimento de várias ferramentas de análise (parsing) e manipulação de documentos XML, por parte das linguagens de programação (e.g., Java). Esta interação parece ter aparecido principalmente devido ao fato do XML permitir "dados portáveis" que podem ser combinados com o "código portável" que a linguagem de programação Java proporciona.

Este ambiente foi dividido em 3 plugins que funcionam de forma independente, mas que juntos formam o ambiente esperado.

• O primeiro plug-in corresponde ao plug-in de manipulação de diagramas do ANote, ferramenta de apoio à modelagem visual e de persistência dos diagramas do ANote.

• O segundo plug-in corresponde ao Transform, ferramenta que dará o apoio à transformação da estrutura intermediária dos diagramas do ANote para a arquitetura de desenvolvimento de sistemas multi-agentes ASYNC (transformação modelo-para-modelo).

• E, para finalizar, o terceiro plug-in corresponde ao Generator, ferramenta responsável pela geração do código parcial, acelerando assim consideravelmente o processo de implementação (transformação modelo-para-código).

Desta forma, as ferramentas são construídas na forma de plug-ins que são adicionados à plataforma do Eclipse estendendo o sistema, como mostrado na figura abaixo:

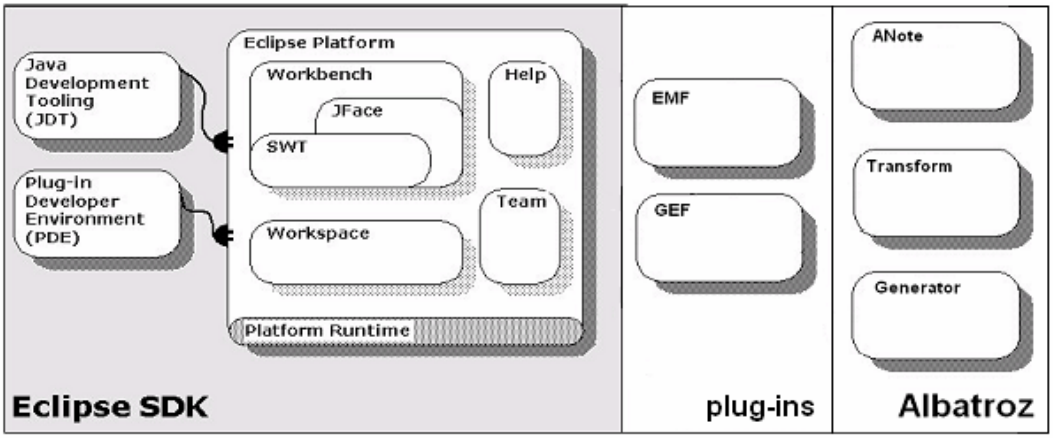


Figura 10 – Plataforma Eclipse com o Albatroz

## 4.2.
## Plug-in de Manipulação de Diagramas do ANote

O plug-in de manipulação de diagramas do ANote (Fig. 11) corresponde a uma ferramenta visual para o apoio à modelagem com o ANote. Este plug-in é responsável por representar graficamente todos os elementos relacionados à linguagem de modelagem ANote, desde as entidades (como agentes, objetivos, etc) e seus relacionamentos, quanto aos diagramas (de organização, agentes, etc).

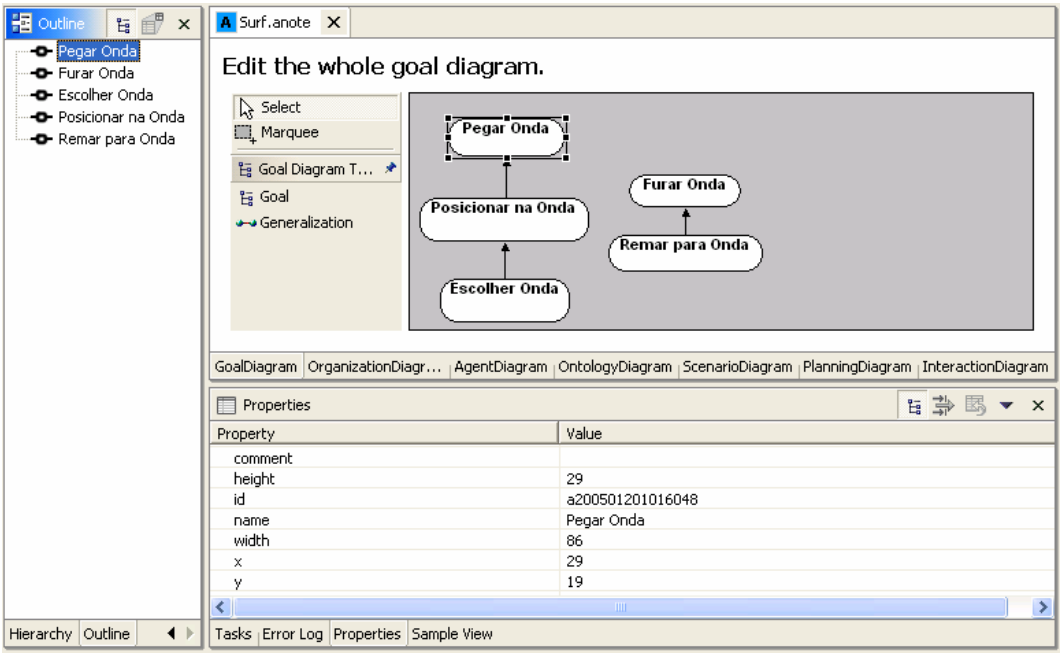


Figura 11 – Ferramenta visual do ANote

Além do apoio à diagramação visual, esta ferramenta faz a consistência entre os diagramas durante a modelagem. Desta forma, o ambiente ajuda o desenvolvedor da aplicação durante a modelagem. Toda vez que uma mudança em um diagrama gerar a necessidade de atualização de um ou mais diagramas de qualquer outra visão do ANote, o desenvolvedor será avisado. O conjunto de regras de consistência entre os elementos de notação do ANote pode ser visto em [23].

Para o desenvolvimento deste plug-in, foi criado um meta-modelo do ANote (Figura 12 e Figura 13) para representação e armazenamento dos diagramas em formato XMI. Assim, este meta-modelo serve de DTD (ver Anexo A) para o arquivo gerado pela ferramenta de apoio visual. Este arquivo permite a interoperabilidade (exportação) de modelos ANote e é o ponto de partida para as ferramentas de transformação e geração de código.

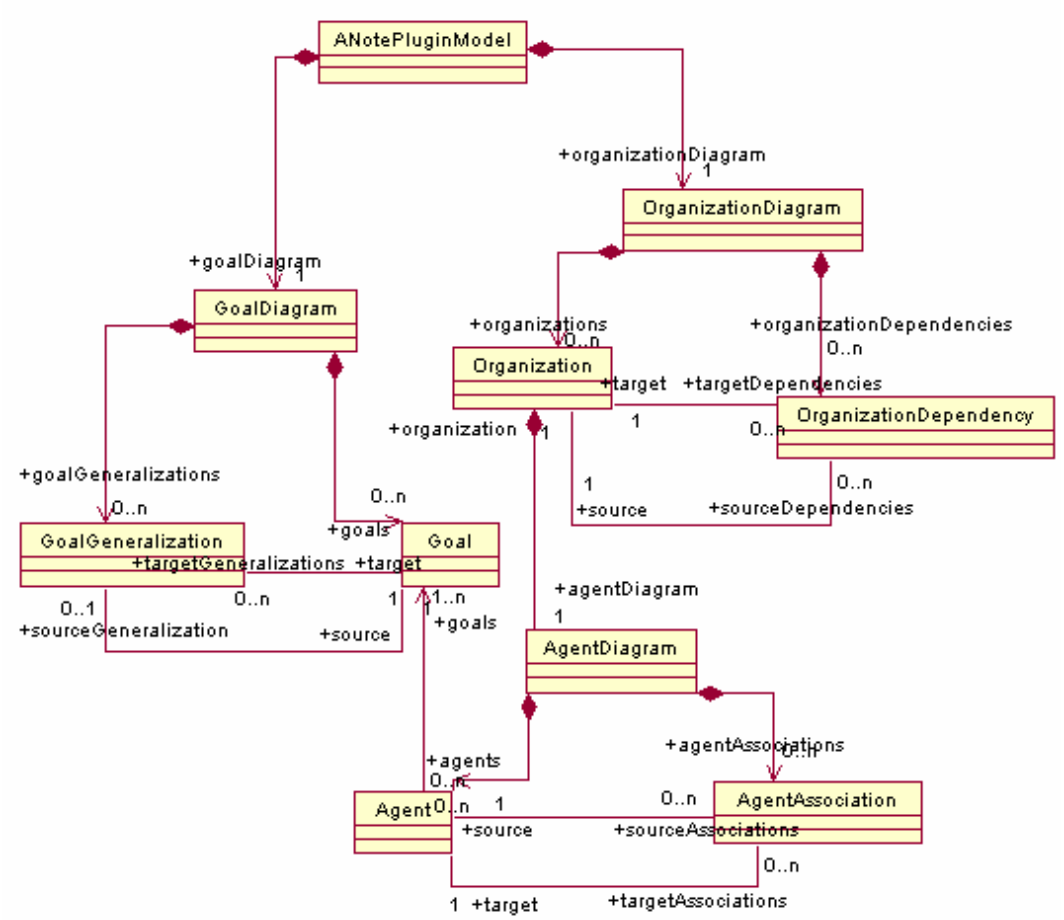


Figura 12 – Meta-modelo do ANote parte 1

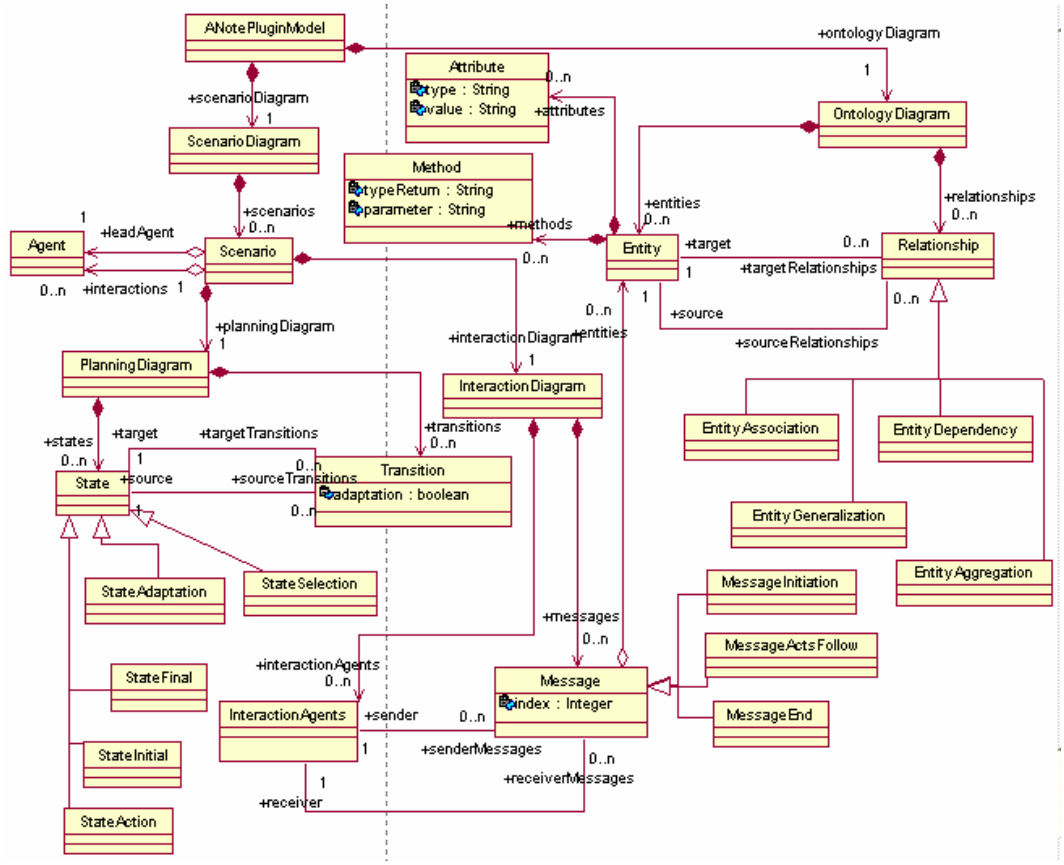


Figura 13 – Meta-modelo do ANote parte 2

```
<?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="a200412150449460" name="GoalDiagram">
      <goals id="a200412150449512" name="Goal A" x="45" y="48" width="60" height="40"
        sourceGeneralization="a200412150449584" />
      <goals id="a200412150449553" name="Goal B" x="208" y="57" width="60" height="40"
        targetGeneralizations="a200412150449584" />
      <goalGeneralizations id="a200412150449584" target="a200412150449553"
        source="a200412150449512" />
    </goalDiagram>
  - <organizationDiagram id="a200412150449461" name="OrganizationDiagram">
    - <organizations id="a200412150450045" name="Oraganization ABC" x="124" y="80" width="60"
        height="60" targetDependencies="a2004121504503013">
      - <agentDiagram id="a200412150450046" name="ADOrganization ABC">
          <agents id="a200412150450097" name="Agent 1" x="131" y="28" width="80"
            height="40" />
        </agentDiagram>
      </organizations>
    </organizationDiagram>
  </anote.plugin.model:ANotePluginModel>
```

Figura 14 – Estrutura intermediária do ANote

### 4.2.1. Os Diagramas do ANote

O plug-in de manipulação de diagramas do ANote permite a modelagem visual dos diagramas de todas as visões do ANote. No diagrama de Objetivos são criados os objetivos e os relacionamentos de decomposição (Figura 15).

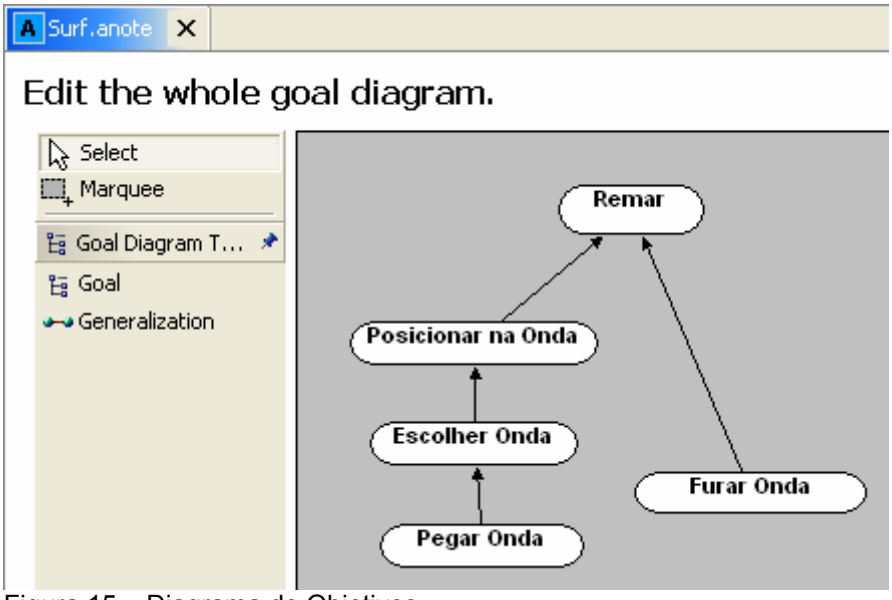


Figura 15 – Diagrama de Objetivos

No diagrama de Organização, são criadas as organizações e suas dependências (Figura 16).

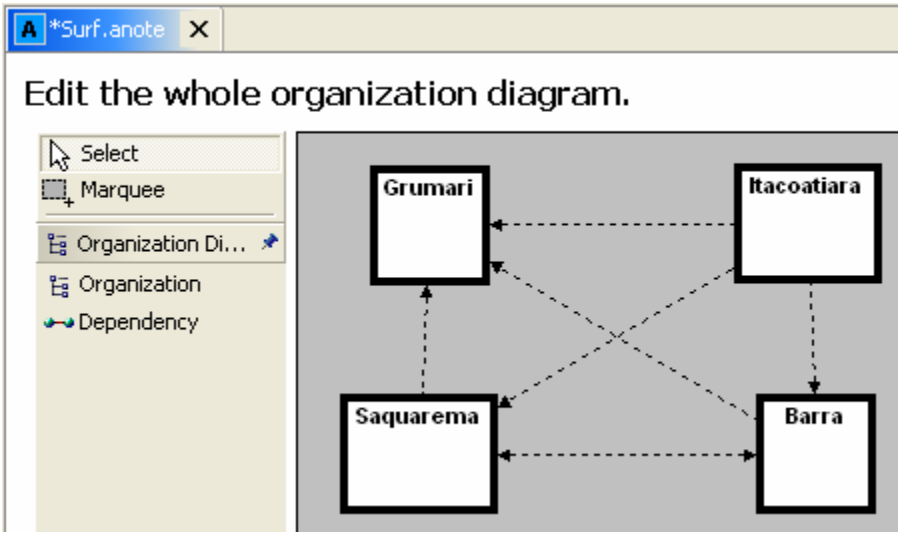


Figura 16 – Diagrama de Organizações

Ao se criar uma organização, automaticamente é criado o diagrama de Classe de Agentes. Desta forma, um agente sempre será criado dentro de uma organização. No diagrama de Classe de Agentes, são criados os agentes, suas associações e sendo obrigatório adicionar, para cada agente, um ou mais objetivos funcionais (folhas da árvore de decomposição de objetivos) pré-existentes no diagrama de Objetivos (Figura 17).

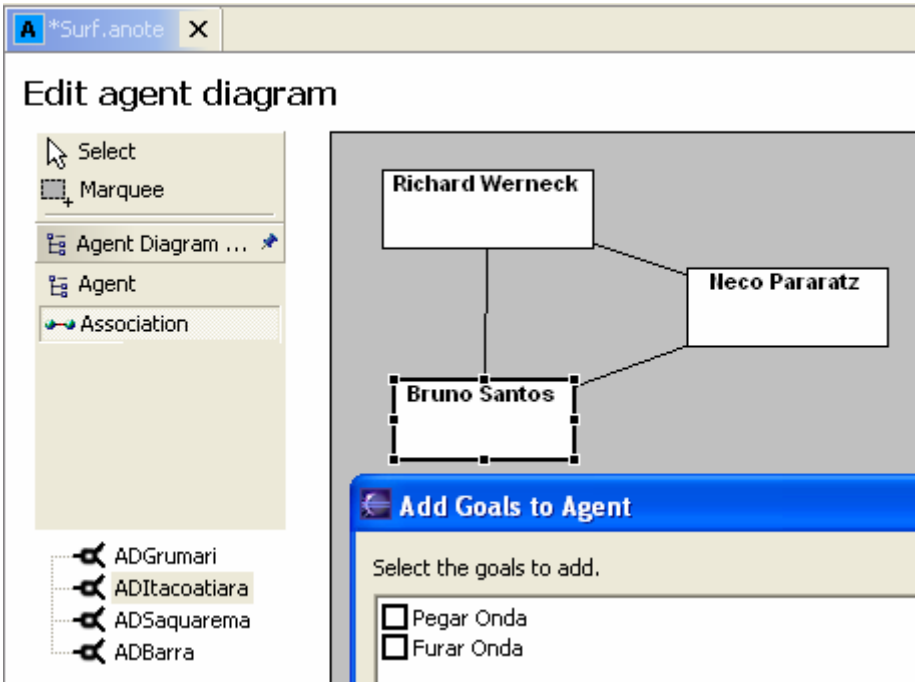


Figura 17 – Diagrama de Classe de Agentes

No diagrama de Ontologias, são criados todos os elementos que não são considerados agentes e seus relacionamentos (dependência, associação, generalização e agregação) (Figura 18). De fato, este diagrama nada mais é do que um diagrama de Classes de UML.

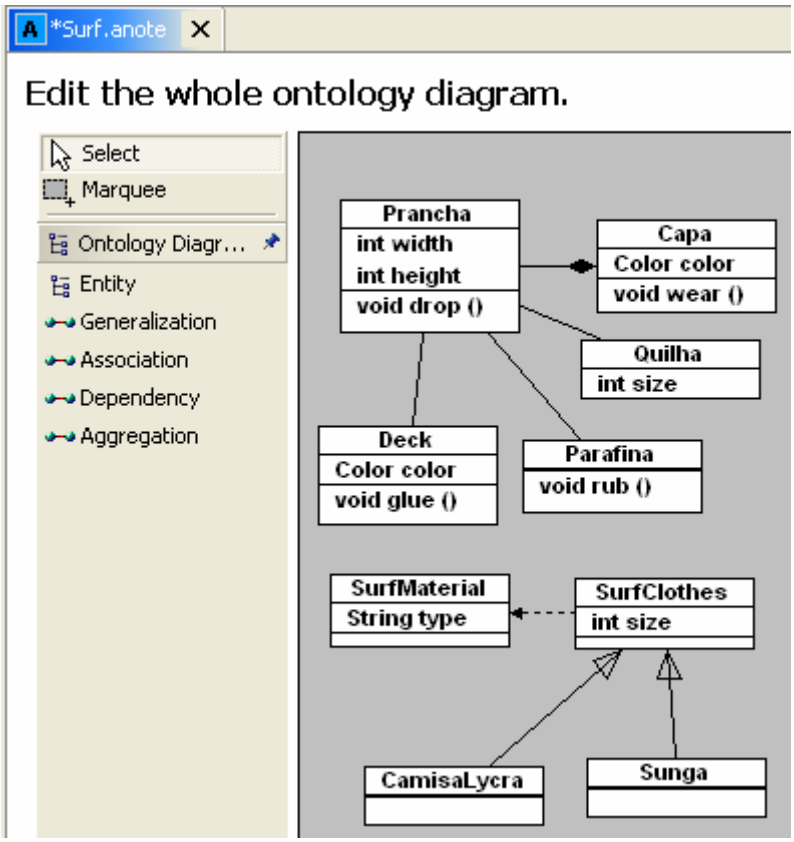


Figura 18 – Diagrama de Ontologias

No diagrama de Cenários, são criados os cenários, podendo adicionar-se para cada cenário um agente principal (leadAgent) e/ou um ou mais agentes de interação pré-existentes no diagrama de Classes de Agentes (Figura 19).

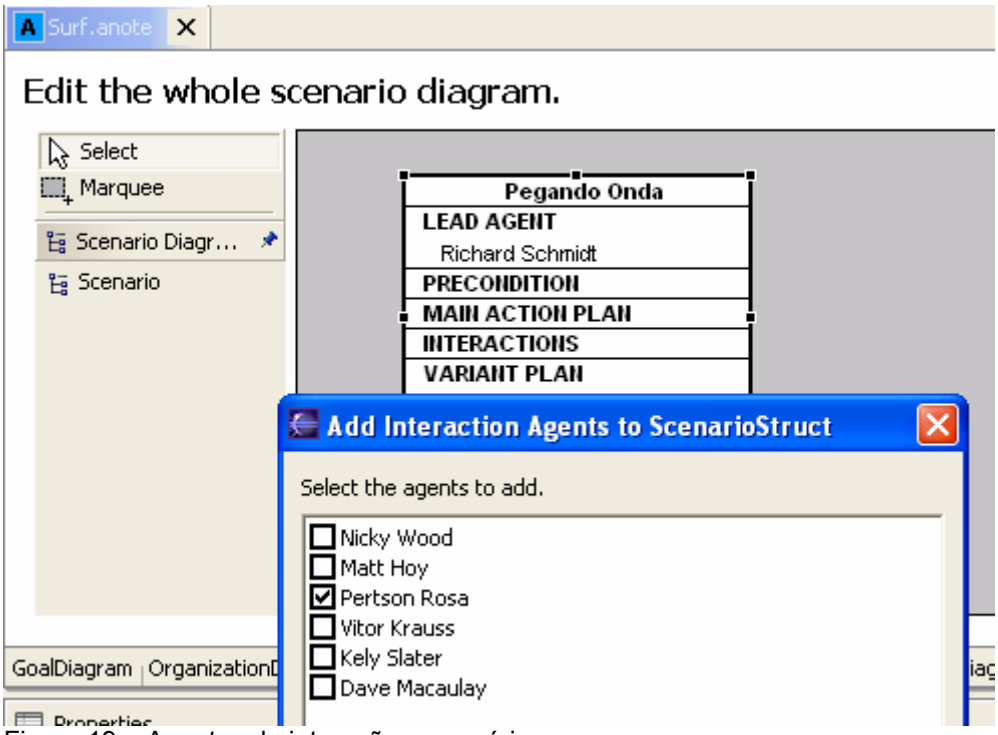


Figura 19 – Agentes de interação no cenário

Ao se criar um cenário, automaticamente são criados os diagramas de Planejamento e Interação. O diagrama de Planejamento será responsável pela elaboração das precondições, do plano principal de ações com suas ações (normais e adaptativas) e transições (também normais e adaptativas), e dos fluxos alternativos do cenário. No diagrama de Planejamento, são criados os estados e suas transições (Figura 20).

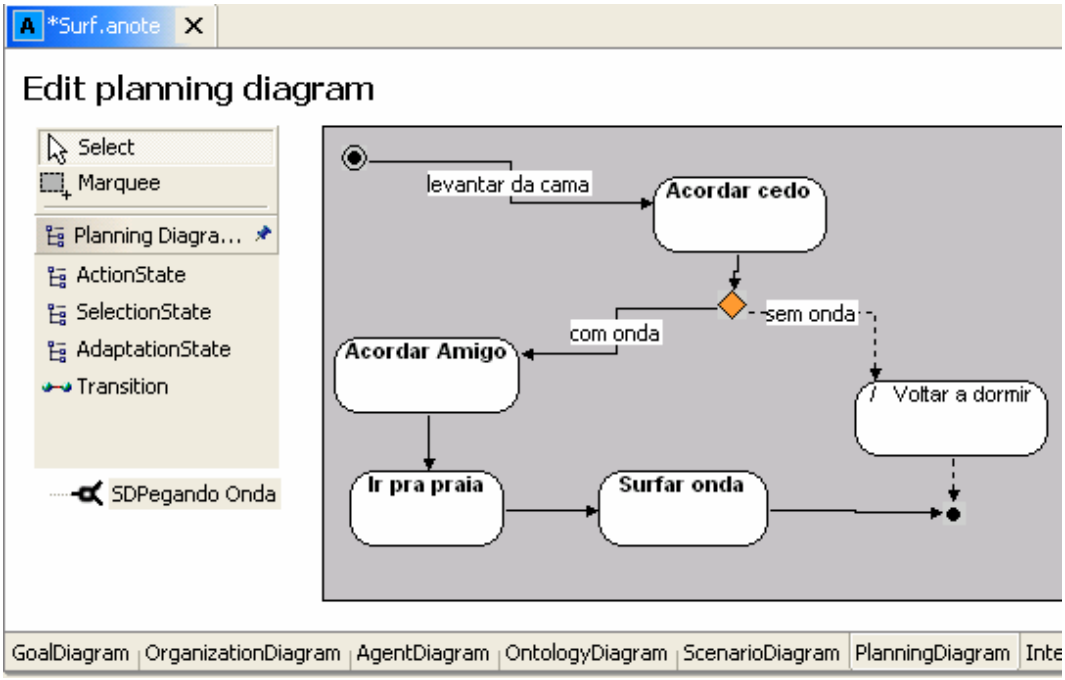


Figura 20 – Diagrama de Planejamento

Ao se adicionar elementos dentro do diagrama de Planejamento, o cenário correspondente é modificado, mantendo a consistência entre estes diagramas (Figura 21).

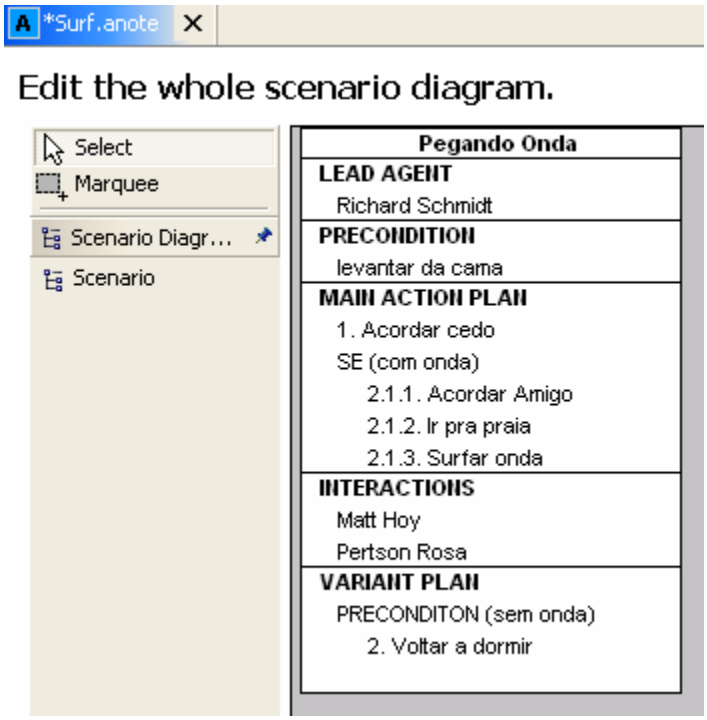


Figura 21 – Diagrama de Cenário

O diagrama de Interação será responsável pela elaboração da interação entre o agente principal e os agentes de interação pré-selecionados no cenário. No diagrama de Interação, são criadas as mensagens de interação dos agentes (Figura 22).

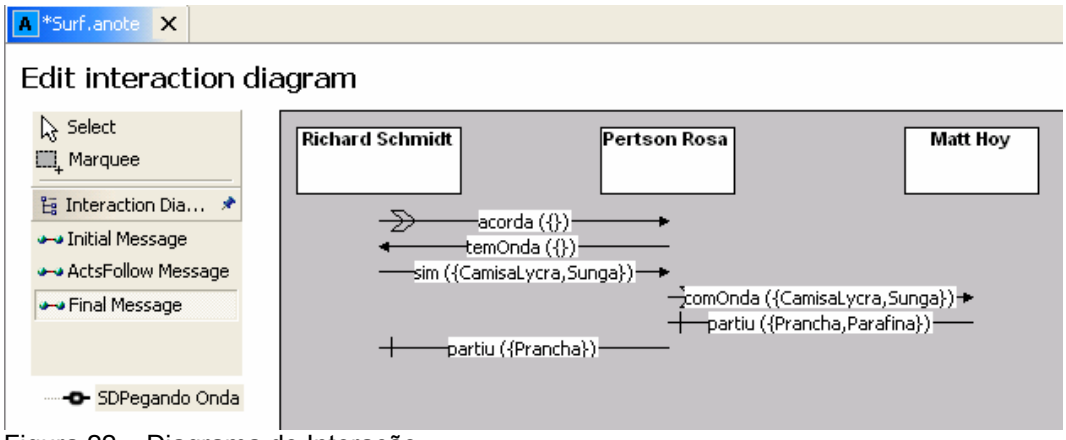


Figura 22 – Diagrama de Interação

O desenvolvedor pode salvar os diagramas antes, durante ou após realizar toda a manipulação visual necessária dos diagramas. Para a ferramenta, salvar os diagramas significa atualizar o XML que os representa (sub-produto da ferramenta que será usado pelos demais plug-ins).

### 4.2.2.
### Funcionalidades Adicionais

Algumas funcionalidades complementam a ferramenta, ajudando o desenvolvedor na elaboração dos diagramas. Estas funcionalidades englobam o uso das visões de propriedades e outline do Eclipse, uso de ampliação (zoom) e um manual do usuário (em português). Na visão de propriedades (Figura 23), pode-se editar as entidades e o diagrama.

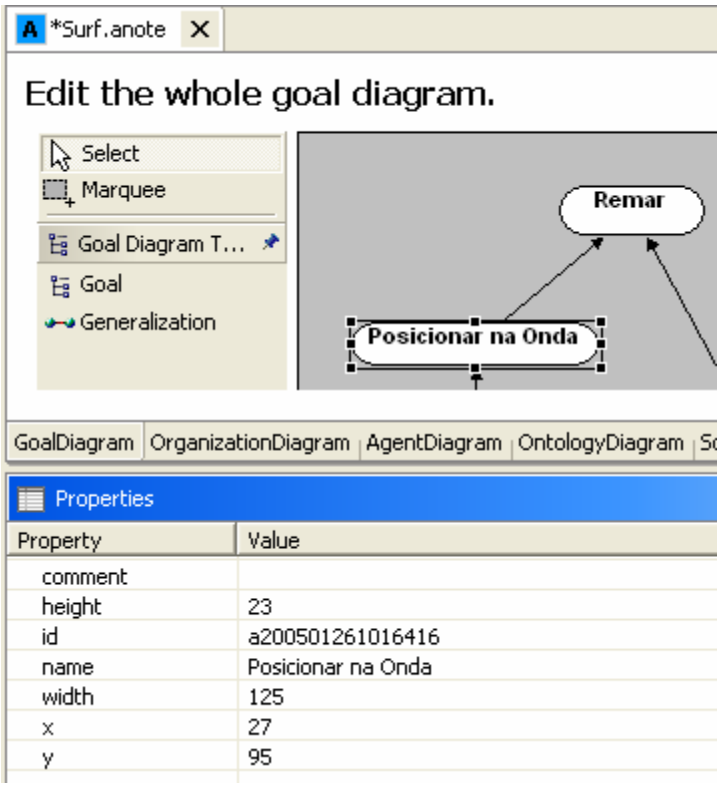


Figura 23 – Visão de Propriedades

Na visão Outline (Figura 24), pode-se encontrar, com mais facilidade, uma entidade em um diagrama.

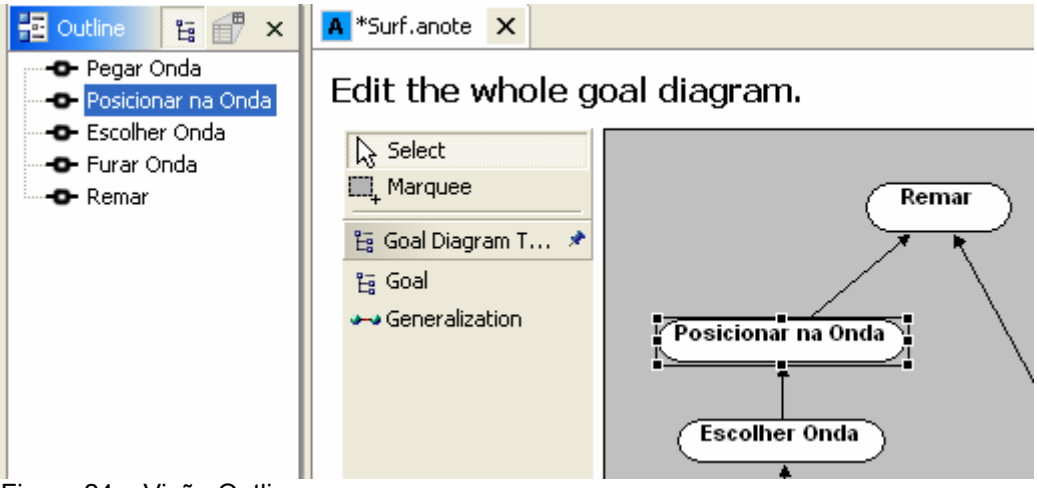


Figura 24 – Visão Outline

O uso de ampliação (Figura 25) é um artifício visual para permitir a manipulação de diagramas grandes e complexos.

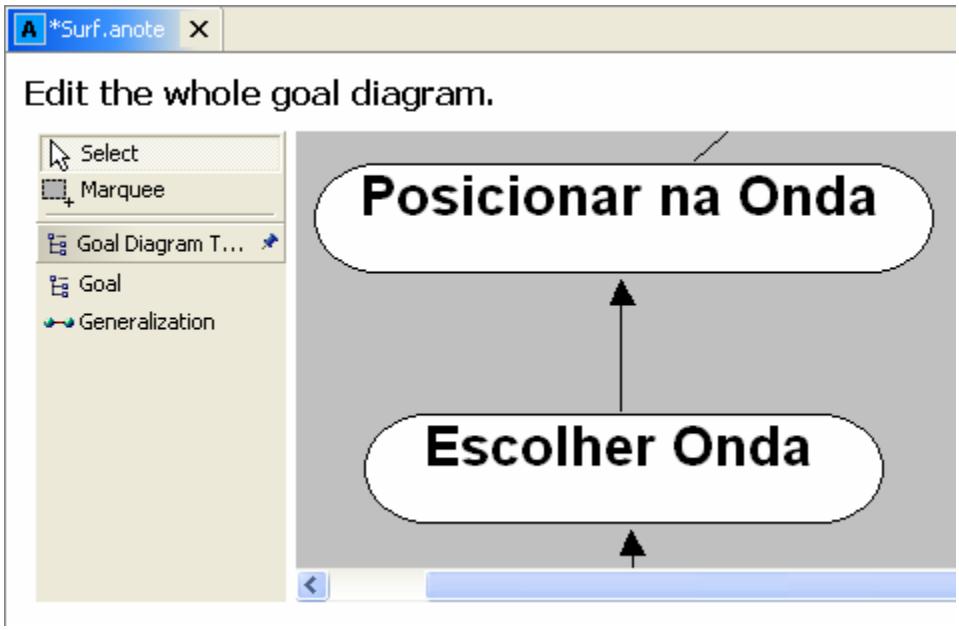


Figura 25 - Zoom

E, por fim, a ferramenta visual conta com uma Ajuda, que mostra um manual do usuário com o objetivo de sanar possíveis dúvidas sobre a linguagem de modelagem ANote (Figura 26).

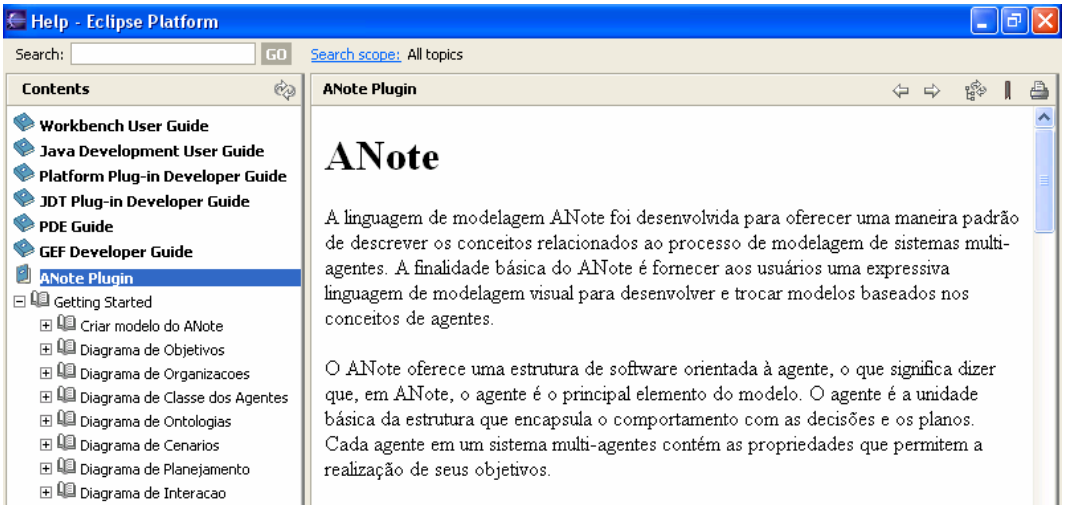


Figura 26 - Help

## 4.3. Plug-in Transform

O plug-in Transform corresponde a uma ferramenta que dará o apoio à transformação da estrutura intermediária dos diagramas da linguagem de modelagem ANote para a arquitetura de desenvolvimento de sistemas multi-agentes ASYNC. Este plug-in transformará o produto gerado pela ferramenta

visual (Figura 14), que é a representação de uma instância do modelo criado do ANote, para o modelo da arquitetura ASYNC (Figura 33) que será entrada para o terceiro plug-in.

A figura abaixo mostra como ocorre a transformação dos modelos. O Modelo A corresponde ao modelo de entrada, o Modelo B corresponde ao modelo de saída e o Modelo C corresponde ao modelo de configuração.

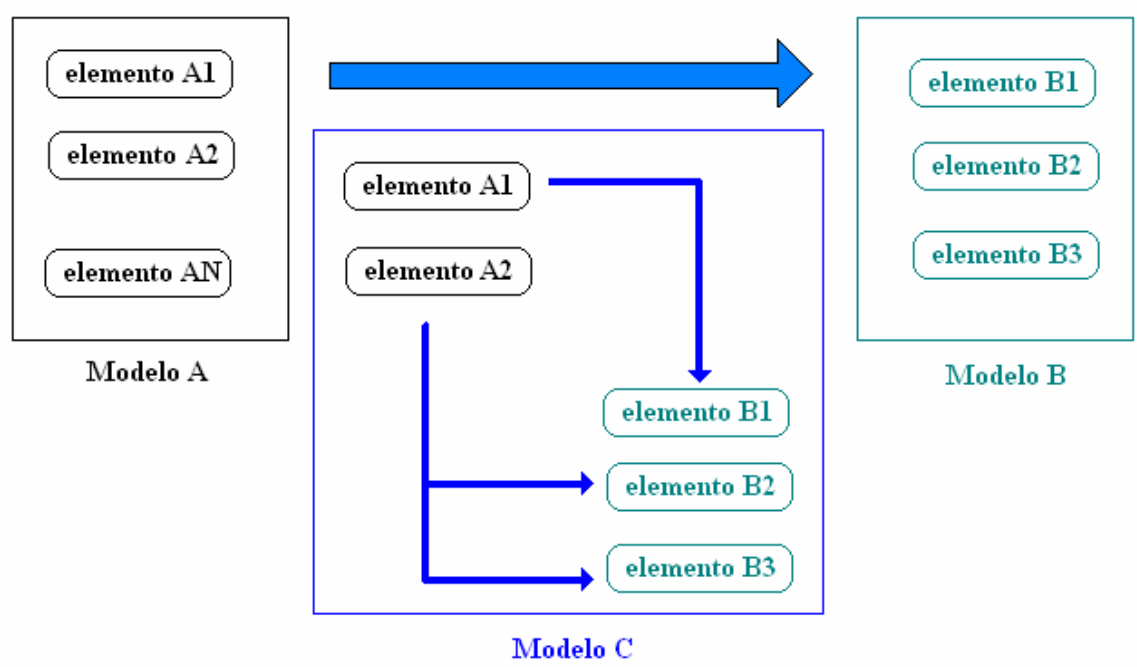


Figura 27- Transformação dos Modelos

Assim, ocorre a comparação do Modelo A com o Modelo C, que será responsável por mapear cada elemento do Modelo A no(s) elemento(s) do Modelo B. Ou seja, para cada elemento do Modelo A, percorre-se o Modelo C em busca dos elementos do Modelo B que devem ser criados segundo o mapeamento definido.

Por isso, um meta-modelo do arquivo de configuração (Fig. 28) foi desenvolvido para armazenamento da configuração do mapeamento para a transformação em formato XMI e seu respectivo DTD (ver Anexo A).

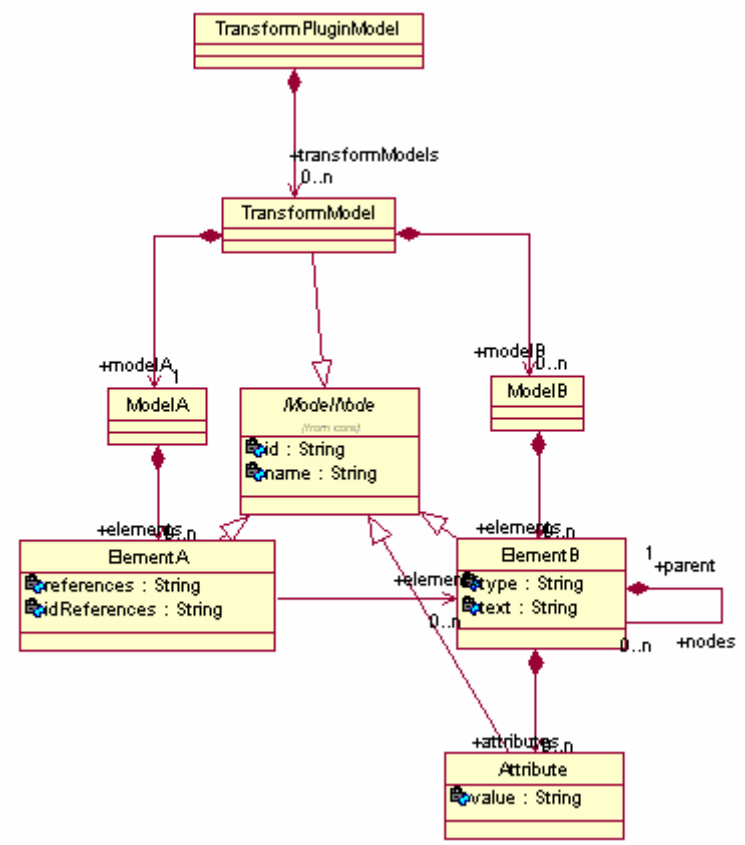


Figura 28 – Meta-modelo do Transform

```
<?xml version="1.0" ?>
- <transformmodel:TransformModel xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI"
    xmlns:transformmodel="http:///transformmodel.ecore">
  - <modelA>
      <elements id="1" name="organizationDiagram.organizations.agentDiagram.agents"
        elements="22 23" />
    </modelA>
  - <modelB>
    - <elements id="10" type="packages">
        <attributes id="21" name="name" value="e.organizations.name" />
      - <elements id="22" type="classes">
          <attributes id="2200" name="name" value="eA.name" />
          <attributes id="2201" name="extends" value="" />
          <attributes id="2202" name="implements" value="" />
          <attributes id="2203" name="actor" value="" />
          <attributes id="2204" name="date" value="" />
        - <elements id="2215" type="attributes">
            <attributes id="2206" name="type" value="eA.type" />
            <attributes id="2207" name="name" value="eA.name" />
            <attributes id="2208" name="value" value="ea.value" />
          </elements>
        - <elements id="2209" type="methods">
```

Figura 29 – Estrutura intermediária do Transform

A tecnologia JDOM [13] foi adotada neste plug-in para facilitar a leitura e escrita dos arquivos XML. JDOM é uma API que facilita a criação e atualização de documentos XML. Por definição, ele é configurado para utilizar o Java API for XML Processing (JAXP) mas pode ser configurado para usar a maioria dos parsers existentes. Assim, prevendo uma futura mudança da maneira como se faz

a leitura e/ou escrita dos arquivos XML foi criado uma estrutura complementar (Figura 31) para o plug-in. Esta estrutura é montada a partir da leitura do arquivo de entrada (Modelo A), e da comparação entre os seus elementos com a estrutura do arquivo de configuração (Modelo C). Em paralelo, é criada a estrutura que representará o arquivo de saída (Modelo B).

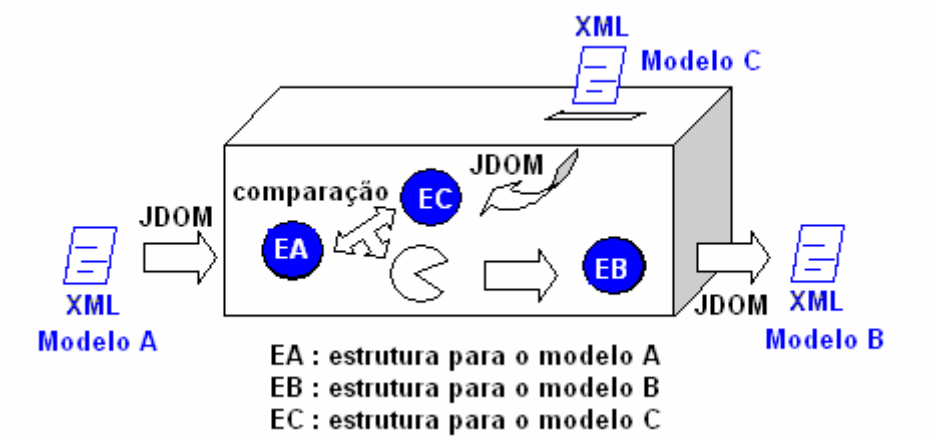


Figura 30 - Transform

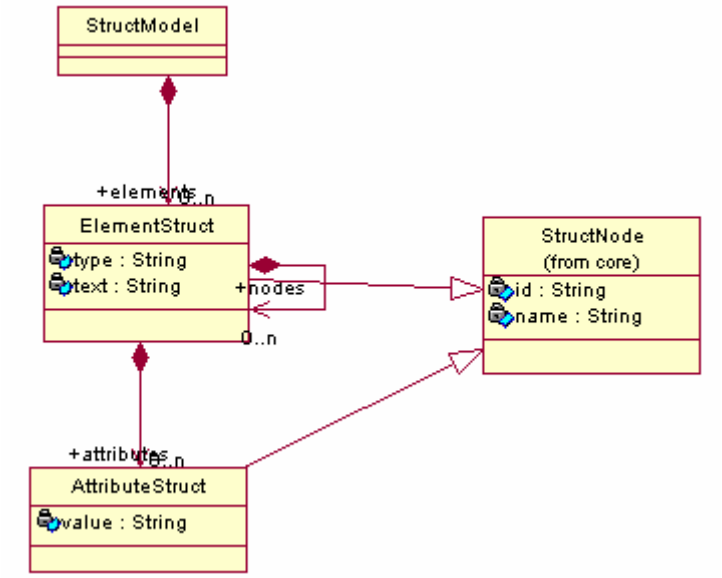


Figura 31 – Estrutura complementar do Transform

Neste plugin, o arquivo de configuração é de extrema importância. Ele será responsável pela corretude da transformação. No fundo, tudo que o desenvolvedor enxerga na modelagem do sistema e espera encontrar no código fonte é inserido e mapeado neste arquivo. As informações inseridas são regras pré-definidas pelo desenvolvedor, podendo ser alteradas de acordo com as necessidades do problema do sistema.

Esta configuração pode variar bastante de acordo com o que se espera do ambiente. Para este trabalho, o arquivo de configuração está pronto para interagir

com a linguagem de modelagem ANote e a arquitetura ASYNC. Todos os elementos relevantes do ANote já estão mapeados para os elementos do ASYNC. Ou seja, foram estabelecidas regras de transformação:

• 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 *Agent* e a outra, com o mesmo nome adicionado da constante "IP" implementando a classe *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 para cada classe nova são mapeados também.

• Regra 2 : para cada cenário (elemento do tipo *scenarioDiagram.scenarios)* encontrado na modelagem do ANote é criado um método para o *agente principal* relacionado com este cenário. Este método é criado dentro da classe criada pela regra anterior que estende 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* e possuirá uma chamada dentro do método *run()*.

• Regra 3 : para cada elemento do diagrama de planejamento (elemento do tipo *scenarioDiagram.scenarios.planningDiagram.transitions* e *scenarioDiagram.scenarios.planningDiagram.states*) encontrado na modelagem do ANote é criado um comando dentro do método criado pela regra anterior com o nome do estado e outro dentro do método *run()* desta classe.

• Regra 4 : para cada entidade (elemento do tipo *ontologyDiagram.entities)* encontrado na modelagem do ANote é criada uma classe correspondente a 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.

• 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, de acordo com o atributo *elements*. Este método é criado dentro das classes criadas pela regra anterior 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.

Desta forma, o modelo gerado por este plugin, seguindo as regras pré-configuradas, estará representando toda a informação necessária para geração parcial do código extraído do modelo de entrada. Esta informação representa os hot spots do framework ASYNC.

### 4.3.1. Flexibilizando a ferramenta Transform para outras Configurações

O plug-in de transformação permite o mapeamento para uma arquitetura que não o ASYNC. Para tanto, basta criar uma nova configuração (Modelo C). Um exemplo de uma outra trnaformação seria utilizar a linguagem de modelagem ANote e fazer o mapeamento para uma Interface Definition Language (IDL) [19]. A IDL é usada no desenvolvimento de sistemas remotos utilizando a tecnologia CORBA [4]. A especificação de CORBA, que faz parte do núcleo do Object Management Architecture (OMA) [18], descreve a infra-estrutura básica para chamadas remotas do procedimento orientada a objetos.

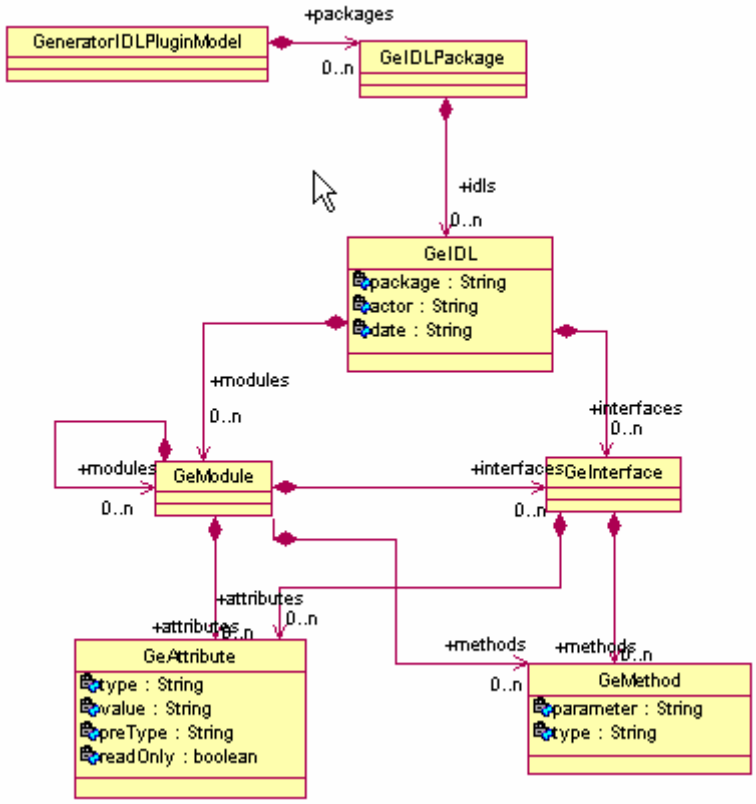


Figura 32 – Meta-modelo do Generator IDL

Neste caso, seria necessário criar um novo modelo de configuração com novas regras de transformação. A título de exemplo, é mostrada abaixo uma dessas regras:

• Para cada agente (elemento do tipo *organizationDiagram.organizations. agentDiagram.agents)* encontrado na modelagem do ANote, é criado uma IDL com o nome do agente que possuirá um *module* com o nome do agente, um atributo com o id do agente (*typedef attribute string id*) e uma interface que terá o nome do agente adicionado da constante "IN". Esta interface terá um método chamado execute com o tipo de retorno *string* e parâmetro *in string id*. Esta IDL nova será colocada num pacote que terá o nome da organização do agente. Exemplo: Esta modelado no ANote uma organização chamada *Org1* que possui dois agentes chamados *AgentA* e *AgentB*. Ocorrendo a transformação são gerados dois pacotes, um com o nome *Org1AgentA* que possuirá a IDL *AgentA* tendo o atributo id com o valor do id do agente e a interface *AgentAIN* com seu método *string executar (in string id)*; e um segundo pacote com o nome *Org1AgentB* que possuirá a IDL *AgentB* tendo o atributo id com o valor do id do agente e a interface *AgentBIN* com seu método *string executar (in string id)*.

## 4.4. Plug-in Generator

O plug-in Generator corresponde à ferramenta responsável pela geração parcial do código, acelerando assim o processo de implementação. É importante salientar que a geração, embora crie código compilável, não cria código completo. Embora a geração automática de código reduza o tempo e a quantidade de programação necessária, ela normalmente gera código de difícil entendimento. Para contornar tal inconveniente, o Generator faz o uso de templates como o EMF. O EMF utiliza duas ferramentas para geração de código: Java Emitter Templates (JET) e Java Merge (JMerge). O JET é utilizado neste trabalho por possuir uma sintaxe simples baseada em JSP, facilitando a escrita dos templates que expressam o código que será gerado (o template utilizado neste plug-in é mostrado no Anexo A).

Não importa qual o framework que será adotado para geração do código, este plugin não se baseia em um framework especifico. Assim, foi desenvolvido um meta-modelo (Figura 33) para armazenamento das informações relevantes

para a criação do código em formato XMI e seu respectivo DTD (ver Anexo A). Buscando gerar código para a arquitetura ASYNC, este meta-modelo é baseado na linguagem de programação JAVA, mas não reflete todos os conceitos e elementos de JAVA (como exemplo Interface, classes abstratas,...), apenas aqueles relevantes às classes que serão hot spots do framework. Ou seja, este plug-in receberá o resultado do plug-in de manipulação de diagramas do ANote como entrada e criará todas as classes necessárias descritas no arquivo de configuração em seus respectivos pacotes, que serão adicionadas a um determinado projeto, formando assim uma casca inicial no auxilio do desenvolvimento das regras de negócio do sistema. Este projeto deverá possuir um arquivo JAVA Archive (JAR) refletindo as classes do framework.

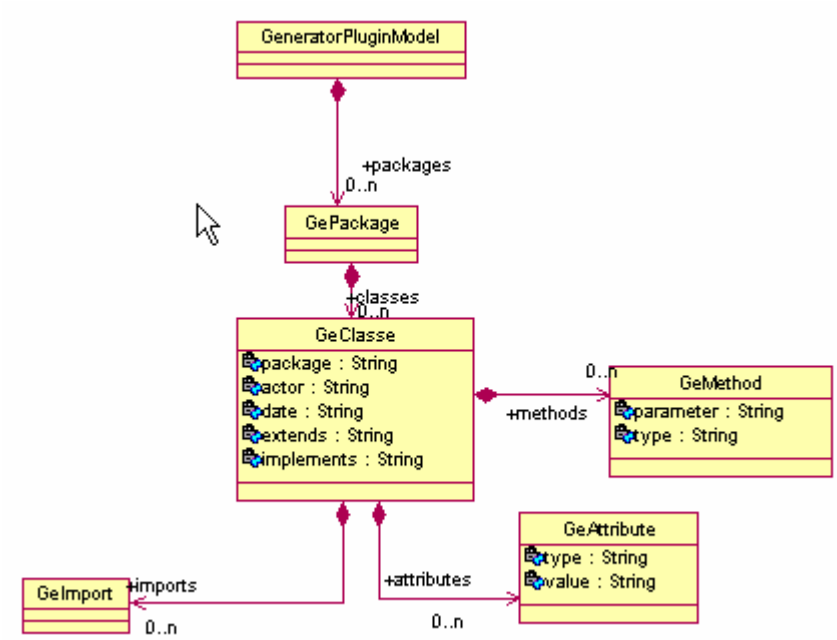


Figura 33 – Meta-modelo do Generator

```
<?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="g1" name="pakageA">
    - <classes name="MyClass1" extends="" implements="" actor="Richard" date="05
        JAN 2005" id="g2">
        <attributes type="int" name="num1" value="123" id="g3" />
        <methods type="String" name="upperCase" parameter="String s"
          id="g4" />
      </classes>
      <classes name="MyClasse1IP" extends="" implements="" actor="rrrr" date="12
        DEZ 2000" id="g5" />
    </packages>
  </generator.plugin.model:GeneratorPluginModel>
```

Figura 34 – Estrutura intermediaria do Generator

O que ainda deverá ser implementado pelo desenvolvedor estará marcado como comentário (Figura 35) na forma de TODO (marcador da visão de tarefas) tornando-se uma ferramenta de apoio no processo de implementação do código completo (ver Anexo A).

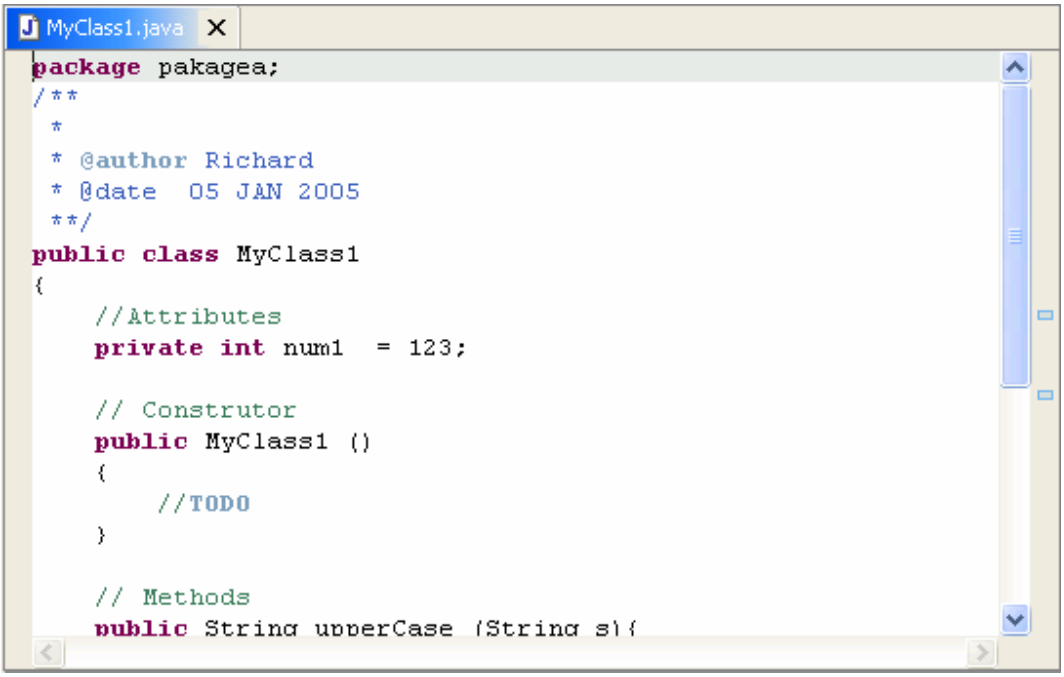


Figura 35 – Classe de exemplo gerada pelo Generator

### 4.4.1. Flexibilizando a ferramenta Generator para outras configurções

Da mesma forma que o plug-in de transformação, o plug-in de geração permite a geração de código para uma outra arquitetura, que, por exemplo, não use JAVA. Voltando ao exemplo da seção 4.3.1, um plug-in GeneratorIDL foi desenvolvido com o intuito de exemplificar uma outra maneira de geração de código e segue os mesmos padrões e tecnologias usadas no Generator Plugin. Este plug-in é responsável pela geração de IDLs que são usadas no desenvolvimento de sistemas remotos utilizando a tecnologia CORBA. Como já demonstrado anteriormente, um meta-modelo (Figura 32) foi desenvolvido para representar a criação das IDLs. Este meta-modelo não procura tratar todos os conceitos e elementos de CORBA, foram modelados apenas aqueles relevantes para a exemplificação. Através da utilização da ferramenta JET do EMF, foi desenvolvido um template mostrado no Anexo A.

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

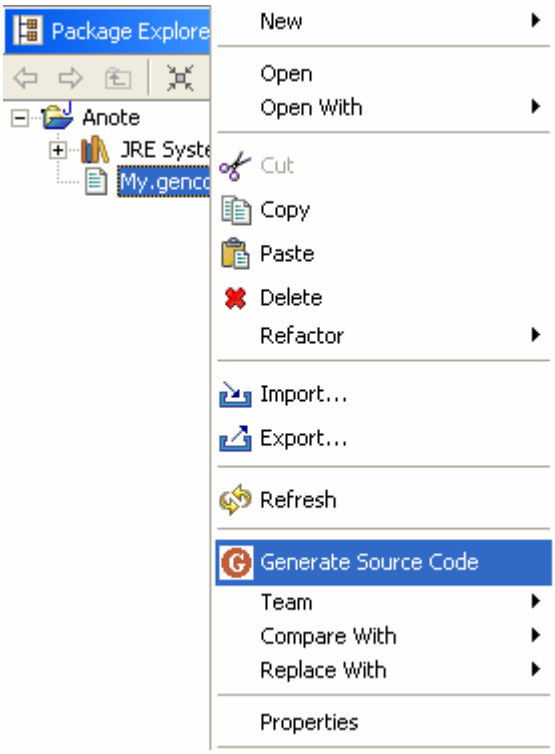


Figura 36 – Chamada para o GeneratorIDL parte 1

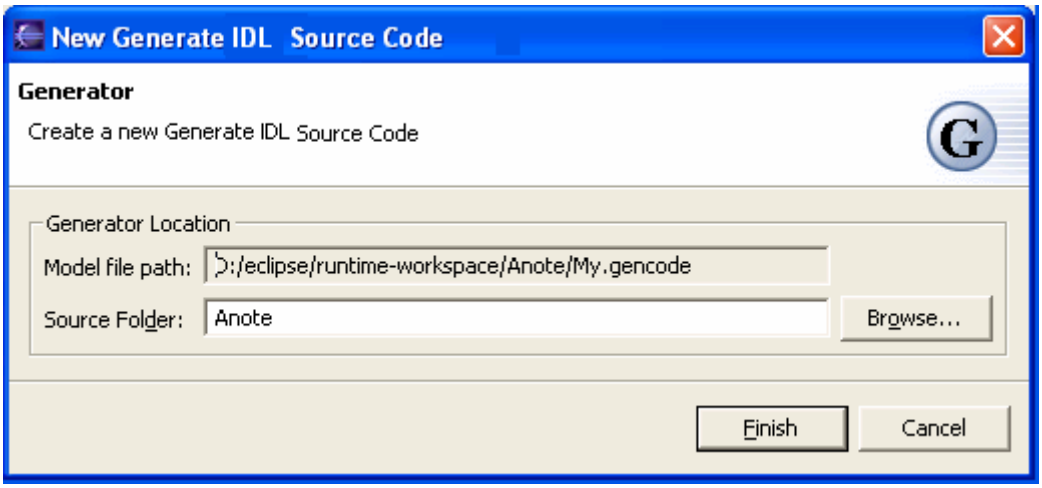


Figura 37 – Chamada para o GeneratorIDL parte 2