3
Fundamentos teóricos e tecnologias básicas

Resumo

Este capítulo apresenta a linguagem de modelagem ANote, a arquitetura de desenvolvimento ASYNC e as tecnologias utilizadas para o desenvolvimento do ambiente

Para um melhor entendimento e elaboração do ambiente de desenvolvimento de software proposto, alguns estudos foram necessários, contribuindo, consideravelmente, com o projeto solução adotado. Neste capitulo, tanto a linguagem de modelagem ANote quanto a arquitetura ASYNC são exploradas mais profundamente e apresentação das técnicas utilizadas para a construção do ambiente.

3.1.
ANote

A linguagem de modelagem ANote [23] foi desenvolvida para oferecer uma maneira padrão de descrever os conceitos relacionados ao processo de modelagem de sistemas multi-agentes. A finalidade básica do ANote é fornecer aos usuários uma expressiva linguagem de modelagem visual para desenvolver e trocar modelos baseados nos conceitos de agentes.

O ANote oferece uma estrutura de software orientada a agente, o que significa dizer que, em ANote, o agente é o principal elemento de modelagem. O agente é a unidade básica da estrutura que encapsula o comportamento com as decisões e os planos. Cada agente em um sistema multi-agentes contém as propriedades que permitem a realização de seus objetivos.

O ANote fornece um conjunto de modelos para a especificação de SMA. Sistemas de larga escala são melhor compreendidos através de um conjunto de modelos quase independentes, ou visões. Uma visão permite que o desenvolvedor do software concentre-se em um único conjunto de propriedades de cada vez.

ANote oferece sete visões que podem ser vistas como uma imagem abstrata de uma parte do sistema multi-agentes que foca em um determinado conjunto de aspectos enquanto que outros aspectos são representados em outras visões. Assim, cada visão do ANote tem uma representação específica fornecendo algumas propriedades. A combinação destas propriedades fornece um conhecimento extensivo sobre a especificação do sistema.

O ANote possui um meta-modelo que funciona como guia para a modelagem do sistema. Um meta-modelo é uma representação dos tipos de entidades que podem existir em um modelo, suas relações e restrições de aplicações. Através de um conjunto de regras de boa formação ocorre a verificação da consistência da modelagem entre as diversas visões durante o desenvolvimento do sistema.

As visões estáticas (estruturais) definem as propriedades estáticas de um sistema multi-agentes. Em ANote, as visões estáticas são usadas para modelar os objetivos, os agentes e o ambiente do sistema.

A visão de objetivos especifica os objetivos do sistema. Um objetivo define um serviço ou uma funcionalidade que algum usuário espera realizar no sistema. Assim, o ANote usa uma orientação baseada em objetivos dos requerisitos como primeiro passo para o processo de modelagem de sistemas multi-agentes.

Esta visão fornece uma identificação inicial de uma árvore de objetivos que esboça as funções executadas pelas entidades que compõem o sistema. Os objetivos complexos podem funcionalmente ser decompostos em objetivos e fluxos constituintes, fornecendo uma descrição como uma árvore hierárquica dos objetivos. Em ANote, um objetivo é um nó na árvore de hierarquia de objetivos e é representado como um retângulo com os cantos arredondados. Os objetivos podem participar em um relacionamento de decomposição que é representado como uma seta que aponta do nó mais específico ao nó mais genérico (veja Figura 1).

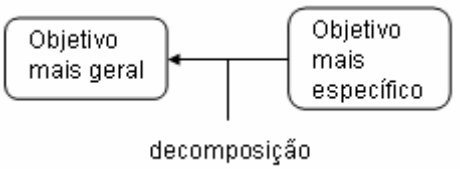

Figura 1 – Visão de Objetivos do ANote

A visão de agentes especifica os tipos de agente que existem em uma solução multi-agentes e em seus relacionamentos, assim definindo a base estrutural do sistema. Como a finalidade desta visão é especificar a estrutura do agente, os agentes definidos aqui são vistos como elementos discretos da modelagem, isto é, nenhum detalhe sobre seu comportamento é fornecido. A estrutura do agente é definida da construção da organização que será responsável pela realização do objetivo selecionado na visão de objetivo. Assim especifica os papéis que executarão funções no processo do fluxo de trabalho do sistema.

Dois ou mais agentes podem interagir em um sistema multi-agente. Se dois agentes necessitarem interagir, há uma ligação estrutural entre eles. Isto é especificado na visão de agente usando o relacionamento de associação. Uma instância de uma associação é uma ligação, que corresponde a uma conexão através da qual os agentes podem interagir no sistema. Conseqüentemente, na modelagem que de dois agentes podem interagir durante a execução do sistema, deve haver uma associação que conecta as duas classes correspondentes do agente. Em ANote, um agente é representado como um retângulo. Uma associação é representada como uma linha como mostrado na Figura 2.

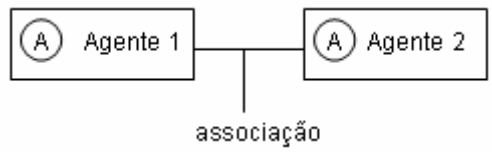

Figura 2 – Visão de Agentes do ANote

As visões dinâmicas definem as propriedades comportamentais de um sistema multi-agentes. Em ANote, as visões dinâmicas são usadas para modelar os cenários, planos e interações do sistema. A visão de cenários captura o comportamento do agente em contextos específicos. O principal elemento do modelo nesta visão é o cenário. Um cenário é uma descrição que denota partes similares de comportamentos possíveis de um agente limitados a um contexto. Um cenário contém a informação de como os objetivos podem ser conseguidos ou não, e descreve também as circunstâncias em que um agente pode se adaptar ou aprender, ou ter um comportamento autônomo. Assim, esta visão também é útil para mostrar características de agência.

Um cenário elabora caminhos para atingir seus objetivos em duas fases. Primeiramente, gera cada curso de comportamento normal, e então identifica

trajetos alternativos para cada seqüência de comportamento emergente (contexto adaptável ou excepcional). Cada caminho torna-se um cenário. Em ANote, os cenários são desenvolvidos como esquemas de alcance do objetivo, tendo por resultado uma representação textual de como os objetivos são executados pelos agentes. Um cenário especifica as seguintes partes: agente principal, pré-condições, plano usual de ação, interação e plano(s) alternativo(s) de ação(ões).

A visão de planejamento especifica os estados de execução, ou as ações, que um agente deve executar para realizar um plano de ação descrito em um cenário. Um plano de ação é modelado de maneira que permita que o agente trace as suas ações internas, seqüenciando os eventos para atingir seu objetivo e tomar as decisões baseadas em seu conhecimento atual. A descrição de plano de ação do agente vem dos cursos de ação (normal e alternativas) descritos na visão de cenário.

Em ANote, os planos de ação são representados como um diagrama de ação muito similar a um diagrama de estado. Tem estados e transições de ação. Adicionalmente, introduz a notação para representar a adaptação do agente com as transições e as ações adaptativas. As transições adaptativas permitem que os desenvolvedores do sistema mostrem quando e sob que circunstâncias um agente deve mudar seu comportamento executando um conjunto de ações especificadas nos planos de ação alternativos de um cenário (Figura 3).

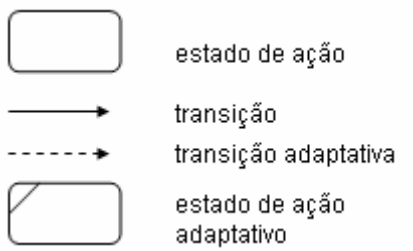

Figura 3 – Visão de Planejamento

A visão de interação é usada para representar o conjunto de mensagens que os agentes trocam ao realizar um plano de ação. Uma mensagem é um fluxo de informação assíncrono entre dois agentes, um remetente e um receptor, e pode ter um conjunto de parâmetros. O nome da mensagem e o conjunto de parâmetros definem o protocolo da mensagem.

Em ANote, as interações são representadas como um diagrama da conversação a fim de descrever o discurso entre os agentes (veja Figura 4). Este diagrama permite que o desenvolvedor mostre o estado atual de uma conversação

e faça uma consistência entre as mensagens emitidas por um agente e as mensagens recebidas por outros.

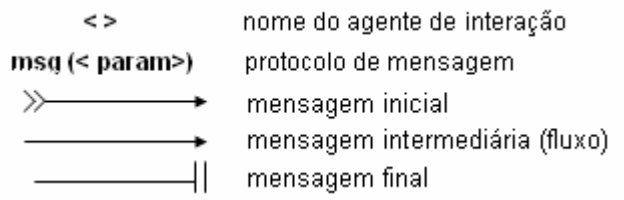

Figura 4 – Visão de Interação

A visão de organização define a estrutura de um sistema multi-agentes. Esta visão especifica as organizações do sistema e seus relacionamentos. Uma organização é uma unidade da execução que oferece serviços (conjunto dos objetivos), alcançados por uma relação (conjunto dos protocolos da mensagem). Esta visão permite que o desenvolvedor visualize um sistema multi-agentes como um conjunto de componentes ou de unidades de execução.

Em ANote, a visão de organização serve para modelar as organizações do agente do sistema. São representados como caixas (veja Figura 5) que podem mostrar o conjunto dos agentes que lhe pertencem. As organizações podem participar em um relacionamento de dependência. Uma dependência mostra que as organizações estão arranjadas em um modelo cliente-servidor. Expressa que um agente de uma organização requer o serviço de um agente em uma outra organização. Uma dependência é representada como uma seta tracejada do cliente à organização do usuário.

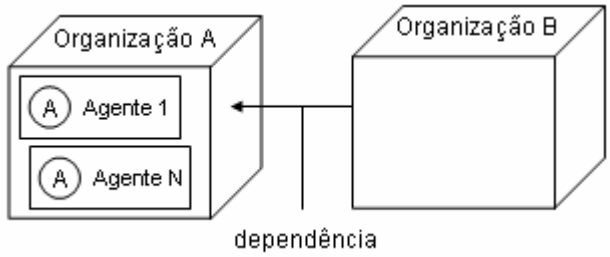

Figura 5 – Visão de Organização

Um sistema multi-agente não é composto somente dos agentes. Existem outros componentes não-agentes que constróem os recursos do ambiente e a base de conhecimento do agente. Em ANote, um ambiente de sistema multi-agente é especificado com uma ontologia que descreve a informação (ou o conhecimento) manipulado pelos agentes e pelos relacionamentos entre eles.

A especificação dos recursos do ambiente é baseada na identificação do mundo do agente. Como estes são componentes não-agente, podem ser modelados

como objetos, com estado e interações definidas com outros objetos. Assim, UML, associada a Object Constraint Language (OCL), pode ser usada como uma alternativa para representar o ambiente do sistema.

Assim a visão de Ontologia é descrita como um diagrama de classe da UML. As entidades podem participar nos mesmos relacionamentos definidos no diagrama de classe da UML. Entretanto, se os recursos do sistema incluírem entidades externas, tais como sistemas da base de dados, outros programas, e assim por diante, esta visão pode ser descrita como um diagrama de componente de UML.

3.2. ASYNC

O ASYNC [12] é um framework orientado a objetos para a criação de agentes de software em um ambiente distribuído. O objetivo principal do framework é reduzir o tempo de desenvolvimento e a complexidade de implementar um sistema multi-agentes distribuído.

O domínio das aplicações cobertas pelo ASYNC é restrito aos sistemas com agentes em um ambiente distribuído e com as seguintes propriedades da agência: autonomia, adaptação, interação e aprendizagem. A arquitetura trata do nível intra-agente, e foca na construção dos agentes que se transformarão em uma parte de um sistema multi-agentes. O ASYNC não trata do nível inter-agente, i.e. operações como registro de agentes, anúncios de serviços, buscas (por nome ou serviço) e etc, que seriam características necessárias para uma plataforma de desenvolvimento de SMA.

O ASYNC utiliza o software TSpaces [11] da IBM, responsável pela infra-estrutura de comunicação. O IBM TSpaces é uma arquitetura reflexiva do espaço de tuplas que fornecem o suporte a todas as operações associativas básicas do quadro-negro (ler, escrever, e capturar). O TSpace pode também ser programado para reagir a estímulos específicos, usando esta característica para permitir a troca de mensagens entre os agentes do software. De fato, no ASYNC, a infra-estrutura de comunicação é uma camada em cima do TSpace que fornece o quadro-negro e as mensagens de comunicação entre os agentes.

O ASYNC possui pontos de flexibilização responsáveis por implementar os objetivos do agente, ações e protocolos de interação e algumas propriedades de agência, tais como a autonomia, a adaptação, a interação, e a aprendizagem. A Figura 6 descreve a arquitetura do ASYNC que é composta de duas classes abstratas - Agent e InteractionProtocols, duas classes finais - ProcessMessageThread e AgentCommunicationLayer, e três interfaces - AgentMessage, AgentBlack-BoardInfo e AgentInterface.

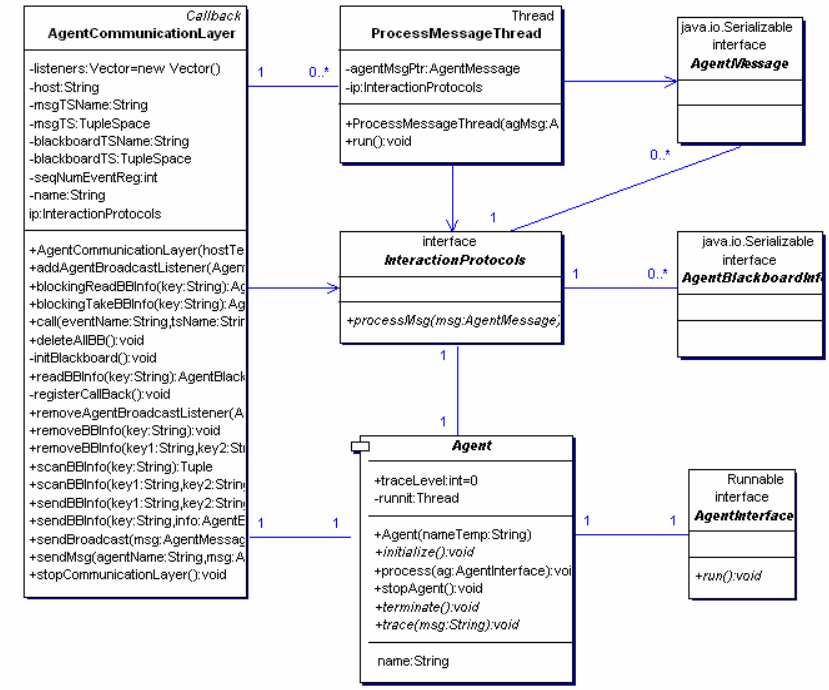

Figura 6 – Diagrama de Classe do ASYNC

Agent é uma classe abstrata, e a sua sub-classe deve executar as ações ou as atividades próprias do agente de software. Estas ações próprias não devem depender de nenhuma interação com outros agentes a fim de realizar suas tarefas específicas. O desenvolvedor é obrigado a implementar os métodos relacionados ao código de inicialização (o método initialize), o código de finalização (o método terminate), e a exposição das mensagens (o método trace). Os métodos process e o stopAgent devem começar e parar o agente. O atributo name especifica o nome do agente no sistema, e devido aos detalhes da implementação, deve ser único. AgentInterface é responsável por transformar a sub-classe de Agent em uma thread. Esta sub-classe executa um método chamado run, que é responsável por iniciar as atividades confidenciais do agente.

InteractionProtocols é uma classe abstrata que define a maneira como um agente de software pode interagir com outros agentes no sistema. Todo o código

relacionado com a interação é colocado nesta classe. A classe executada requer também a execução de um método chamado processMsg. Este método é chamado cada vez que uma mensagem nova é recebida de um outro agente. ProcessMessageThread se encarrega de processar as mensagens recebidas pelos agentes. De fato, será criada uma thread para cada nova mensagem recebida, que chamará automaticamente o método abstrato processMsg. AgentMessage é uma interface usada pela classe que especifica o formato da mensagem, e AgentBlackBoardInfo é uma interface usada pela classe que especifica o formato da mensagem do quadro-negro. Assim, toda a informação do blackboard e das mensagens do sistema devem executar estas interfaces.

AgentCommunicationLayer é uma classe que implementa a infra-estrutura de comunicação necessária para que os agentes interajam em um sistema distribuído sobre uma rede. Esta infra-estrututra é uma camada em cima do TSpaces da IBM.

O projeto da arquitetura orientada a objetos pode ser dividido em duas partes: o núcleo do subsistema e em pontos de flexibilização do subsistema. O projeto do núcleo do subsistema é comum a todas as instâncias das aplicações, e na arquitetura ASYNC as classes AgentCommunicationLayer, ProcessMessageThread e AgentIntelligence formam o núcleo. O projeto dos pontos de flexibilização descreve características diferentes para cada instanciação da aplicação. No ASYNC os pontos de flexibilização são as classes Agent, InteractionProtocols, AgentMessage, AgentBlackBoardInfo e AgentInterface.

3.3. Tecnologias Utilizadas

3.3.1. XML

Extensible Markup Language (XML) é uma meta-linguagem de marcação desenvolvida pelo consórcio World Wide Web Consortium (W3C), sendo considerada um padrão para a publicação e transferência de dados de diferentes domínios de aplicação na Web [30]. Sendo uma linguagem de marcação, XML utiliza delimitadores (ou tags) para descrever dados, assim como a linguagem HTML. A diferença entre HTML e XML é que tags em HTML descrevem

instruções para apresentação de dados em browsers Web, enquanto tags em XML denotam uma interpretação semântica para o dado delimitado por ela. Como esta interpretação é particular para cada domínio de aplicação, XML é dita uma meta-linguagem, pois cada aplicação tem a liberdade de definir a sua própria linguagem de marcação de dados, descrevendo a nomenclatura e a estruturação das tags que julgar apropriada.

A organização ou esquema dos elementos em um documento XML pode ser definida basicamente de duas maneiras: através de um Document Type Definition (DTD) ou de um XML Schema Definition (XSD) [28]. Ambos especificam elementos válidos que podem ocorrer em um documento, a ordem na qual eles ocorrem e algumas restrições sobre a estrutura e o conteúdo destes elementos. Um documento XML é dito válido se estiver de acordo com um esquema DTD ou XSD associado a ele. DTD foi a primeira recomendação da W3C para a especificação de esquemas de documentos XML. Um DTD é uma gramática formada basicamente por cláusulas ELEMENT e ATTLIST. Estas cláusulas descrevem, respectivamente, um elemento e uma lista de atributos de um elemento.

3.3.2. JAVA

Java foi desenvolvida por um grupo de pesquisadores da SUN Microsystems por volta de 1990, pouco antes da explosão da Internet. Java é uma linguagem orientada pelos objetos, baseada em classes e tipada. Também suporta concorrência, mediante a utilização de métodos synchronized e de objectos da classe predefinida Thread. Java resulta, em grande parte, duma simplificação do C++ (concretamente da primeira versão desta linguagem).

A linguagem Java, sendo relativamente pequena, herda muito do seu poder duma grande biblioteca de classes predefinidas, chamada plataforma Java (também conhecido por core Java APIs e Java Runtime Environment). Esta biblioteca é robusta, intuitiva e bem desenhada. Tem vindo a crescer e a amadurecer com cada nova versão do sistema.

Atualmente, a plataforma Java é tão vasta que os programadores já não precisam acessar diretamente aos serviços do sistema operacional. Assim, a plataforma Java apresenta-se como uma plataforma de desenvolvimento universal.

Sobre ela correm todos os programas Java, independentemente do sistema operacional subjacente.

O ambiente de desenvolvimento Java Developer's Kit (JDK) consiste num conjunto de ferramentas que ajudam a desenvolver, testar, documentar e executar programas em Java. O JDK é gratuitamente disponibilizado pela empresa Sun e existem versões para os mais variados sistemas: Linux, Windows, MacOS, etc. A versão utilizada para este ambiente é J2SDK 1.2.4.

3.3.3. Plataforma Eclipse

A plataforma Eclipse [10] é projetada para construção de ambientes integrados de desenvolvimento (IDE) podendo ser usada para criação de aplicações tão diversas quanto web sites, programas em Java, programas em C++ e Enterprise JavaBeans (EJB). Sua plataforma se encontra baseada num modelo de plug-ins, ou seja, cada ferramenta na verdade é um plug-in integrado com a plataforma.

O Eclipse fornece um modelo comum de interface com o usuário (IU) para trabalhar com suas ferramentas. É projetado para funcionar em vários sistemas operacionais (SO) enquanto fornece integração robusta com cada SO subjacente.

No núcleo do Eclipse está uma arquitetura para a descoberta dinâmica dos plug-ins. A plataforma segura a logística do ambiente baixo e fornece um modelo padrão da navegação do usuário. Cada plug-in pode então focalizar em fazer um número de tarefas bem pequeno (como desenvolvendo, testando, publicando, compilando, eliminando erros, diagramatizando...).

A plataforma do Eclipse é estruturada em torno do conceito de pontos de extensão. Os pontos de extensão são lugares bem definidos no sistema onde outras ferramentas (chamadas plug-ins) podem contribuir com sua funcionalidade.

Cada subsistema principal na plataforma é propriamente estruturado como um conjunto de plug-ins que executam alguma função chave e definem pontos de extensão. O sistema do Eclipse é propriamente construído por contribuições dos mesmos pontos de extensão que seus fornecedores de plug-ins utilizam. Os Plug-ins podem definir seus próprios pontos de extensão ou simplesmente adicionar extensões aos pontos de extensão de outros plug-ins.

Os subsistemas da plataforma adicionam tipicamente características visíveis à plataforma e fornecem APIs estendendo suas funcionalidades. Alguns destes componentes fornecem bibliotecas adicionais que não se relacionam diretamente a um ponto de extensão, mas podem ser usados para executar extensões. Para o exemplo, a bancada UI fornece a JFace UI framework e o Standard Widget Toolkit (SWT).

O Eclipse SDK inclui a sua plataforma básica mais duas ferramentas principais que são úteis para o desenvolvimento de plug-in. O Java development tooling (JDT) que executa um ambiente caracterizado no desenvolvimento de Java. E o Plug-in Developer Environment (PDE) adiciona as ferramentas especializadas que tornam mais eficientes o desenvolvimento dos plug-ins e das extensões.

Estas ferramentas servem não somente a uma finalidade útil, mas fornecem também um exemplo grande de como as ferramentas novas podem ser adicionadas à plataforma construindo plug-ins que estendem o sistema.

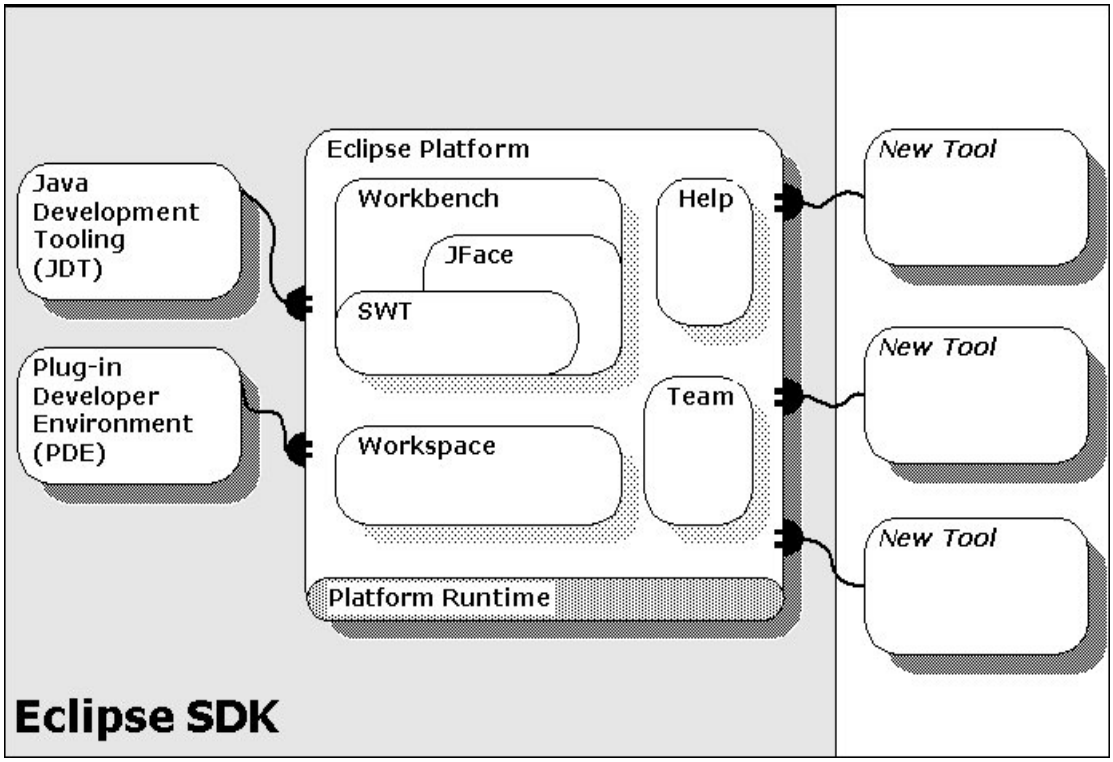

Figura 7 – Plataforma do Eclipse

Este trabalho utiliza dois importantes plug-ins do Eclipse, o Graphical Editing Framework (GEF) [9] e o Eclipse Modeling Framework (EMF) [8].

3.3.3.1. Graphical Editing Framework - GEF

O GEF [9] permite que o desenvolvedor crie um rico editor gráfico baseado em um modelo existente. Para fazer isto, GEF fornece os visualizadores (do tipo EditPartViewer) que podem ser usados em qualquer lugar no workbench do Eclipse. Como os visualizadores do JFace, os visualizadores de GEF são adaptadores em um controle de SWT. Cada controlador, ou EditPart como são chamados, são responsáveis por fazer mudanças no modelo. EditParts são os objetos com os quais o usuário interage.

GEF fornece dois tipos de visões: gráfica e baseada em árvore. Cada um hospeda um tipo diferente de visão. As figuras são definidas no plug-in Draw2D, que é incluído como parte do GEF. O TreeViewer usa uma árvore de TreeItems de SWT para sua view.

A maioria de aplicações de GEF são editores do Eclipse (IEditorPart). O EditorPart é o único ponto de entrada no código das aplicações. Uma vez criado, construirá o volume das partes restantes. Um editor gráfico completamente fundido consistiria geralmente no seguinte:

- EditorPart, que constrói o controle preliminar e a maioria das outras peças
- Controles de SWT e visores de GEF para aqueles controles
- um PaletteViewer e um palette modelado para ele
- um OutlinePage, que pudesse hospedar um dos visores de GEF
- um PropertySheetPage, que, entre outras coisas, deva integrar potencialidades de undo/redo.
- um EditDomain, que amarre junto tudo, e hospeda a pilha do undo

O seguinte diagrama descreve o modelo em memória de um editor típico de GEF. O Eclipse cría o EditorPart e emite-lhe então o IEditorInput. O EditorPart inicializa o EditDomain, os visualizadores, a palette, etc. O EditorPart amarra então o EditPartViewers fornecendo um EditPart como os índices.

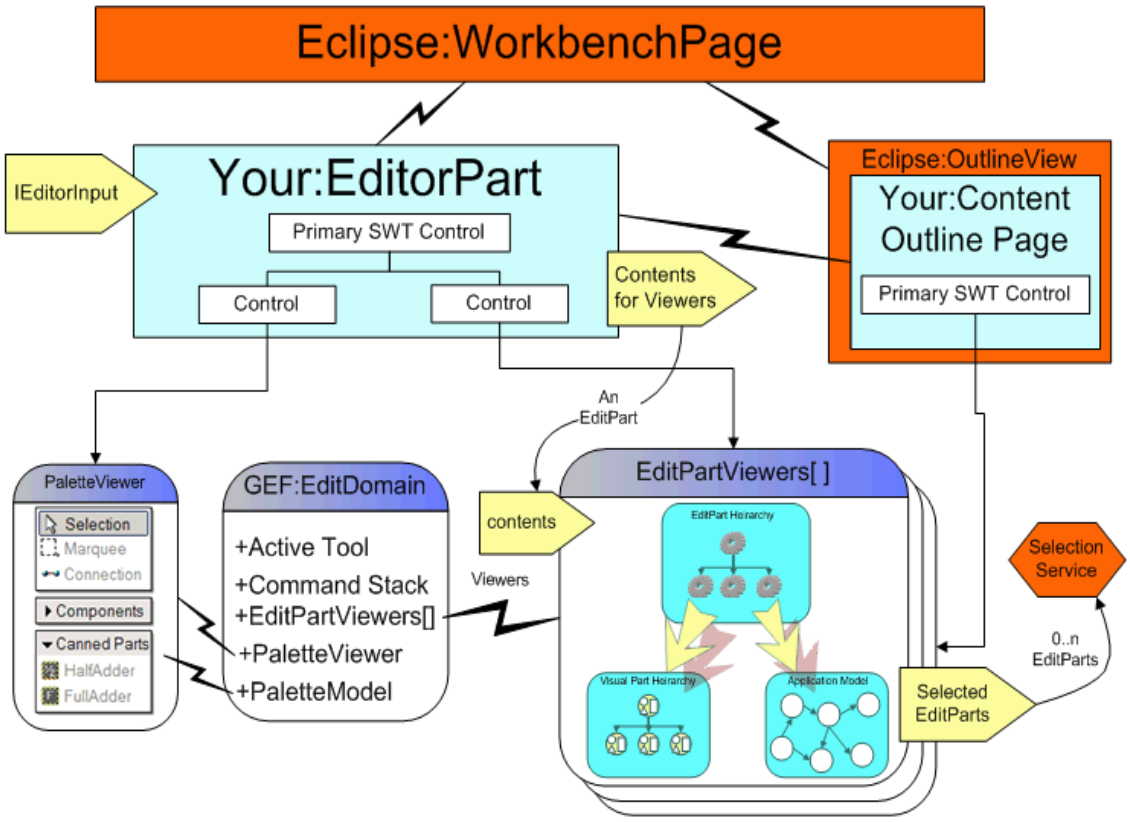

Figura 8 – Diagrama do GEF

3.3.3.2. Eclipse Modeling Framework - EMF

Já o EMF [8] é um framework de modelagem e geração de código para construção de ferramentas e outras aplicações baseadas na estrutura do modelo de dados. De uma especificação do modelo descrita em XMI, EMF fornece ferramentas e sustentação em tempo de execução (runtime) para produzir um conjunto de classes Java para o modelo, um conjunto de classes do adaptador que permitem a visualização e a edição do modelo, e um editor básico.

Assim, ao falarmos de modelagem, pensamos geralmente sobre coisas como diagramas de classe, diagramas de colaboração, diagramas de estado, e assim por diante. Unified Modeling Language (UML) define uma notação padrão para estes tipos de diagramas. Usando uma combinação dos diagramas da UML, um modelo completo da aplicação pode ser especificado. Este modelo pode ser usado puramente para a documentação ou, dadas ferramentas apropriadas, pode ser usado como entrada para geração parcial ou, em casos simples, completa da aplicação.