Um Modelo para Construção e Integração de Ferramentas

Guilherme Horta Travassos; Ana Regina Cavalcanti da Rocha
Universidade Federal do Rio de Janeiro
COPPE - Sistemas
Caixa Postal 68511
21945- 970 Rio de Janeiro - Brasil
ght@cos.ufrj.br

Resumo: *Este artigo descreve o modelo utilizado na Estação TABA para a construção e integração de ferramentas. A Estação TABA, uma Estação de Trabalho configurável para desenvolvimento de software, utiliza esta filosofia de integração para realizar a instanciação dos ambientes especificados pelo meta-ambiente TABA, a construção de novas ferramentas na Estação e a integração de ferramentas externas desenvolvidas fora da Estação.*

Palavras-chave: ADS, CASE, Integração

1. Introdução

O projeto TABA [Rocha90] tem como objetivo a construção de uma Estação de Trabalho configurável para o desenvolvimento de software. A motivação para o projeto é prover desenvolvedores de software com Ambientes de Desenvolvimento de Software (ADSs) que atendam às particularidades de domínios de aplicação e projetos específicos. Para atender a este objetivo a Estação TABA tem quatro funções:

I) auxiliar o engenheiro de software na especificação e instanciação do ambiente mais adequado ao desenvolvimento de um produto específico;

II) auxiliar o engenheiro de software a implementar as ferramentas necessárias ao ambiente definido em (I);

III) permitir aos desenvolvedores do produto (software) o uso da Estação através do ambiente instanciado em (I) ;

IV) permitir a execução do software, dado que, eventualmente, o software produto poderá ser executado na própria Estação para ele configurada (o que é sempre verdade pelo menos na fase de testes).

Estas funções desejadas para a Estação TABA fazem com que, na definição de sua estrutura, seja criado um conjunto de serviços, agrupados em ambientes distintos, responsáveis pela efetiva realização das funções. Desta forma tem-se quatro ambientes que são:

- ***Ambiente Especificador e Instanciador de ADSs:*** é o meta-ambiente TABA. Sua função é especificar o ADS mais adequado para o desenvolvimento de um produto de software num determinado domínio de aplicação e instanciar o ADS. O instanciador é responsável por selecionar entre os componentes disponíveis na Estação aqueles que foram indicados para compor a instância de ADS, e tornar operacional o ambiente. Além de tornar operacional o ADS cabe ao instanciador agregar ao ADS instanciado um assistente baseado em conhecimento de auxílio à sua utilização. A construção do especificador de ambientes, XAMÃ [Aguiar92], foi realizada e está operacional em ambiente Sun.
- ***Ambiente para Construção de Ferramentas*** : os componentes disponíveis na Estação foram, em algum momento, incorporados à mesma. Entretanto, pode ocorrer

que a especificação de um ADS contenha requisições a componentes ainda não disponíveis no meta-ambiente. Neste caso, os componentes devem ser construídos, permitindo a instanciação completa do ADS especificado. Além disso, novas ferramentas podem, a qualquer momento, ser sugeridas (especificadas) pelo meta-usuário. O Ambiente para Construção de Ferramentas é o responsável por auxiliar na construção destas novas ferramentas e na sua incorporação ao meta-ambiente.

- ***Ambiente de Desenvolvimento:*** é o ADS que foi especificado e instanciado, através do meta- ambiente.
- ***Ambiente de Execução***: é o local onde o produto de software poderá ser executado.

Assim sendo, para atender a suas funções, a Estação TABA deve possuir, entre outros, os seguintes requisitos:

- ***possuir um mecanismo de integração,*** de forma a permitir, e facilitar, a integração de ferramentas, sejam elas desenvolvidas com a tecnologia utilizada no próprio Projeto TABA (ferramentas internas), ou então, desenvolvidas por terceiros (ferramentas externas);
- ***apoiar na construção de novas ferramentas,*** possuindo funcionalidades que permitam ao Engenheiro de Software construir, ou adaptar, novas ferramentas para a Estação.

Baseado nestes requisitos e tendo em vista que as estruturas convencionais de ADS encontradas na literatura [Penedo92], [Ecma90], [Brown92] apresentam algumas dificuldades [Travassos94] foi adotado, para a Estação TABA, a distribuição das funcionalidades necessárias em *componentes*, que conseguem interagir diretamente entre si, permitindo assim a obtenção de toda funcionalidade do ambiente.

Estas funcionalidades, ou componentes, do ADS representam a forma segundo a qual ele consegue atender aos requisitos especificados para sua construção. Os componentes presentes na Estação TABA são:

- O ***Sistema Computacional***, representado por uma dimensão de hardware e outra do sistema operacional;
- O ***componente de Interface com o Usuário,*** que é o responsável pela gerência e controle da interação do usuário com o ADS. É de sua responsabilidade garantir a consistência da apresentação das ferramentas e permitir, na medida do possível, que o usuário ajuste a aparência do ADS às suas preferências pessoais;
- O ***componente Cooperação*** trata da comunicação entre os usuários e suas necessidades de interação com outros membros da equipe. Possui definições de protocolos de comunicação que devem ser utilizados no convívio no ADS. Este componente precisa auxiliar o componente de Interface com o Usuário no sentido de prover funcionalidades e recursos que o estendam de forma a suportar a interface de grupo;
- O ***componente Controle dos Processos*** é o responsável pelo controle e gerenciamento do processo de desenvolvimento. Identifica atividades, gerencia a execução das ferramentas e controla os papéis e atividades dos usuários, levantando dados sobre a realização do processo para o meta-ambiente.
- O ***componente Suporte Inteligente*** fornece inteligência global ao ambiente, assistindo o usuário na utilização do meta-ambiente e dos ADSs.

- O ***componente Reutilização*** provê o ADS de mecanismos que possibilitam o reaproveitamento de trabalhos anteriores. A reutilização se dá a nivel da especificação do ADS, de componentes para a construção de ferramentas, bem como de componentes de todo tipo para a construção da aplicação. Este componente possibilita ao instanciador e ao construtor de ferramentas o acesso a itens de software e ferramentas previamente desenvolvidos;
- O ***componente Conhecimento*** incorpora mecanismos de inferência que possibilitam a integração do conhecimento armazenado no ADS. O meta-ambiente se utiliza deste componente para a obtenção de características de ADS a serem instanciados. Ferramentas também usam este componente para obterem informações sobre o significado dos itens de software que estão manuseando. Embora possa parecer, à primeira vista, que este componente possui as mesmas funcionalidades do componente Suporte Inteligente, na realidade, o que ocorre é um forte relacionamento entre eles. Neste componente existem mecanismos que permitem o gerenciamento e armazenamento do conhecimento. O componente funciona, para o ambiente, auxiliando outros componentes em seu funcionamento, ao passo que no anterior, o objetivo principal é apoiar o usuário na utilização do ADS;
- O ***componente Repositório Comum do ADS*** é o responsável pelo controle, gerenciamento e armazenamento dos objetos manuseados pelo ambiente e meta-ambiente. É de sua responsabilidade manter a consistência e a integridade das informações;
- O ***componente Controle de Versões***, que está muito relacionado com o repositório comum, controla e gerencia versões de documentos e itens de software produzidos. A partir dele o usuário tem condição de obter informações sobre um determinado momento no desenvolvimento.

A utilização, em conjunto, destes componentes permite a integração de ferramentas ao ambiente. Estes componentes definem políticas de armazenamento, a forma de interface com o usuário, embora não interferindo em sua funcionalidade, provêm recursos para a incorporação de ferramentas externas ao ambiente, e, de forma mais específica, definem o modelo, ou filosofia, de integração da Estação TABA e de seus ambientes instanciados.

2. A Filosofia de Integração da Estação TABA

Ferramentas em ADS devem suportar algum método de desenvolvimento, e ainda, possibilitar que as informações a elas associadas estejam disponíveis para todo o ambiente, facilitando que outras ferramentas tomem conhecimento sobre o processo de desenvolvimento e o significado das informações existentes, e sobre a forma em que estas informações podem ser utilizadas. Para que isto seja possível entendemos que é necessário considerar, além da integração de dados, controle e apresentação, conforme proposto por diferentes autores e em diversos ambientes [Brown92], [Thomas92], [Chen92], [Meyers91], [Mi92], [Yang93], [Penedo92], ***a integração do conhecimento.***

A ***integração de apresentação e interação*** é responsável por proporcionar ao usuário a sensação de integração no ambiente. É através dela que será possível a homogeneização das formas de apresentação da informação e das técnicas de interação do usuário. Neste aspecto, concordamos com Thomas [Thomas92], acrescentando uma terceira característica, a

customização da interface. Este tipo de integração é responsabilidade do componente Interface com o Usuário.

A ***integração de gerenciamento e controle*** é responsável por prover, às ferramentas e ao ambiente, serviços e funcionalidades que permitam seu funcionamento de forma organizada. Este tipo de integração é responsabilidade dos componentes Cooperação, Controle dos Processos, Reutilização e Controle de Versões.

A ***integração de dados*** estabelece a forma como as ferramentas realizarão o tratamento das informações. Deve prover os serviços básicos de armazenamento e gerenciamento de estruturas de informação (responsáveis por descrever os documentos gerados durante o desenvolvimento) obtidas a partir das ferramentas componentes do ambiente. Este tipo de integração é responsabilidade do componente Repositório Comum do ADS.

A ***integração do conhecimento*** possibilita às ferramentas um melhor entendimento da semântica das informações e dos métodos existentes para tratar a informação. Este conhecimento deve estar disponível no ambiente, de forma a poder ser reutilizado por outras ferramentas. Consideramos, também, que integração é uma filosofia de construção e como tal deve ser considerada como um todo, levando-se em consideração que os próprios mecanismos de integração têm que estar integrados. A integração do conhecimento torna disponíveis os serviços básicos de armazenamento, gerenciamento e utilização do conhecimento descrito e adquirido ao longo do processo de desenvolvimento. O conhecimento compartilhado pelas ferramentas descreve o significado das informações construídas ao longo do processo de desenvolvimento, e permite a descrição da sintaxe apropriada para a apresentação destas informações por parte das ferramentas. Este tipo de integração é obtido a partir da utilização dos componentes Suporte Inteligente e Conhecimento.

3. O Modelo de Construção e Integração de Ferramentas

Considerando os requisitos da Estação TABA podemos identificar duas questões. A primeira diz respeito ao meta modelo TABA, e trata do meta ambiente e suas necessidades funcionais. A segunda se apresenta no momento em que, a partir da utilização da Estação TABA, ambientes são instanciados e tornam-se passíveis de utilização para o desenvolvimento de produtos. Estas duas situações precisam ser consideradas no contexto da construção e integração de ferramentas.

Para a descrição e especificação dos dois contextos descritos acima será utilizado o método Análise Orientada a Objetos [Coad92].

3.1 O Meta Modelo TABA

O principal objetivo da Estação TABA é permitir a especificação de ADSs adequados a diferentes contextos e sua instanciação como um ambiente integrado. Esta instanciação se faz a partir de um conjunto de facilidades e funcionalidades que suportam o Engenheiro de Software na definição e construção de novos ambientes. O meta modelo da Estação TABA está representado no diagrama de assuntos da figura 1. Neste diagrama são encontrados os seguintes assuntos que representam, a partir das classes que os compõem, a funcionalidade da Estação:

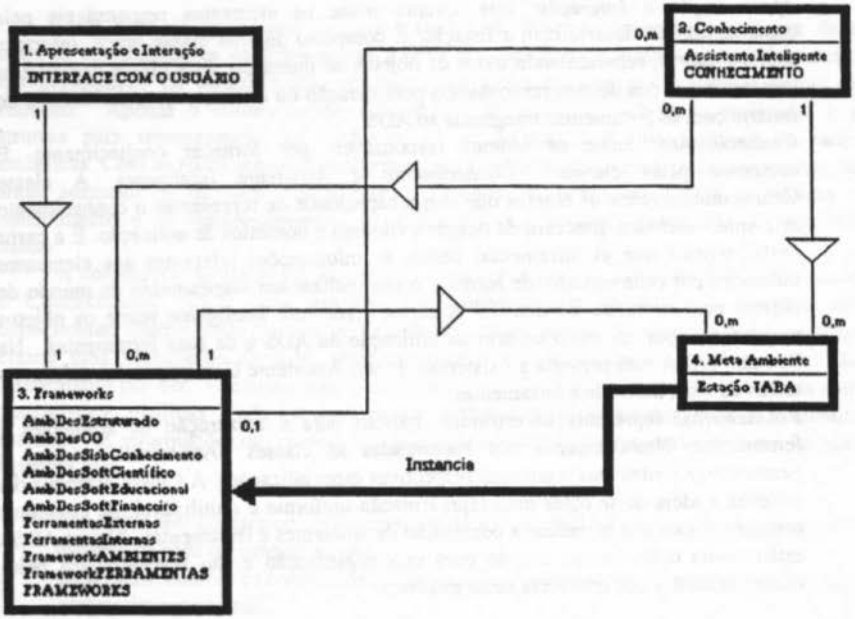

Figura 1 - Diagrama de Assuntos do Meta Ambiente TABA

- ***Meta Ambiente:*** este assunto representa a essência do modelo. É composto apenas pela classe *Estação TABA*. O objeto descrito por esta classe é o responsável pela coordenação das tarefas do meta ambiente, tornando disponíveis, quando necessário, conhecimento, padrões de interface com o usuário e estruturas para a construção de ADS's e ferramentas.

  A classe Estação TABA é descrita a partir do conjunto de funcionalidades necessárias para o funcionamento da Estação. A instância da classe TABA é responsável pela coordenação dos trabalhos no meta ambiente e possui um conjunto de serviços associados que permitem ao meta-usuário utilizar a Estação:

  Definir Ambiente: este serviço realizado com o suporte de XAMÃ [Aguiar92], permite a especificação de ADSs que atendam às características de projetos específicos.

  Instanciar ambiente: a partir da definição do ADS instancia um novo ADS e o torna disponível para uso.

  Testar Ambiente: permite testar o ambiente instanciado e verificar se atende aos requisitos. *Descrever Ambiente*: permite a descrição de novos ADSs para a Estação.

  Sugerir Ferramenta: permite descrever uma nova ferramenta que deve, no futuro, ser desenvolvida, ou acrescentada, à Estação TABA.

  Construir Ferramenta Interna: permite a construção de ferramentas que tenham sido, anteriormente, sugeridas (descritas) para a Estação, e que irão utilizar o modelo de integração TABA.

  Acrescentar Ferramenta Externa: permite tornar componentes da Estação TABA, ferramentas que tenham sido desenvolvidas sem utilizar sua filosofia de integração.

- ***Apresentação e Interação***: este assunto reúne os elementos responsáveis pela comunicação do usuário com a Estação. É composto de uma classe única, *Interface com o Usuário*, representando todos os objetos de interação. Esta classe descreve as características dos objetos responsáveis pela iteração do usuário com o ambiente e do usuário com as ferramentas integradas ao ADS.
- ***Conhecimento:*** reúne os objetos responsáveis por fornecer conhecimento. É composto pelas classes *Conhecimento* e *Assistente Inteligente*. A classe Conhecimento reúne os objetos que têm a capacidade de representar o conhecimento referente a métodos, processo de desenvolvimento e domínios de aplicação. É a partir destes objetos que as ferramentas obtêm as informações referentes aos elementos utilizados por cada método, de forma a poder realizar um mapeamento do mundo de objetos para o mundo do usuário. A classe Assistente Inteligente reúne os objetos responsáveis por assistir o usuário na utilização do ADS e de suas ferramentas. Na Estação TABA está prevista a existência de um Assistente Configurável a diferentes ciclos de vida, métodos e ferramentas.
- ***Frameworks:*** representa as estruturas básicas para a construção de ambientes e ferramentas. Neste assunto são encontradas as classes *FrameworkAmbientes* e *FrameworkFerramentas*, com suas respectivas especializações. A Classe Frameworks sintetiza a idéia de se obter uma representação uniforme e reutilizável, de estruturas organizacionais que permitam a construção de ambientes e ferramentas. Alguns ADS's estão sendo definidos de acordo com esta organização e são representados pelas classes restantes que aparecem nesta estrutura.

Devido à importância das classes Conhecimento e Frameworks, no contexto da Estação TABA, estas classes serão descritas com mais detalhe a seguir.

3.1.1 A Classe Conhecimento

A instanciação da classe Conhecimento é realizada a partir da representação e modelagem do conhecimento de cada método, através do preenchimento de uma estrutura de conhecimento, definida em Duarte [Duarte91a], [Duarte91b]. De acordo com esta abordagem, para expressar os métodos e suas características particulares, é necessário:

- estabelecer a forma de proceder do método, ou seja, como se origina e se deriva. Deve-se expressar o seu comportamento através de um conjunto de atitudes. Este tipo de conhecimento é denominado "Procedimentos";
- identificar todo o conhecimento que pode ser apreendido, ou seja, tudo aquilo que pode ser manipulado e perceptível para o entendimento do método. A partir daí, é necessário e possível expressar o tipo de associação entre esses conhecimentos identificados. Este tipo de conhecimento é denominado "Objetos/Relações";
- permitir se fazer a descrição dos objetos do método, ou seja, expor de forma circunstanciada e narrativa o objetivo e/ou característica do conhecimento identificado. Este tipo de conhecimento é denominado "Descrição";
- determinar a distribuição em classes e/ou grupos, segundo um sistema ou método de classificação, ou seja, os conhecimentos que representam categorias. Este tipo de conhecimento é denominado "Classificação", e,
- especificar verdades de formação irregular a partir de experiências de especialistas. Este tipo de conhecimento é denominado "Heurísticas".

Uma base de conhecimento para métodos de desenvolvimento de software deve conter os conhecimentos descritos acima, e deve ser construída utilizando um modelo misto de representação do conhecimento composto por redes semânticas, quadros e regras de produção. Apenas o conhecimento relativo a "Objetos/Relações" foi considerado, e a estrutura para representação do conhecimento se restringiu a uma estrutura de redes semânticas. Com as redes semânticas obtém-se uma visão macroscópica do conhecimento, de forma adequada e de fácil entendimento. Os nós da rede identificam e rotulam o conhecimento e os relacionamentos, e expressam as conexões apropriadas ao domínio tratado.

A construção da classe Conhecimento é realizada através da utilização de Estruturas de Conhecimento (EC). Uma estrutura de Conhecimento identifica e descreve um conhecimento representado por um nó na rede de conhecimento. Sua função organizacional permite uma otimização da rede, onde características particulares do conhecimento são representadas por ela. A Estrutura de Conhecimento (EC) é formada por seis estruturas com características distintas. Cada estrutura possui atributos para representar, num determinado momento, as informações das características do conhecimento representado pelo nó. As suas estruturas são:

- **Identificação**: contém os conhecimentos de identificação da rede e do domínio, para cada nó da rede de conhecimento;
- **Descrição**: contém os conhecimentos necessários para o entendimento do nó e de suas características básicas;
- **Construção**: contém as informações relativas à construção da rede de conhecimento;
- **Estatística**: contém conhecimentos quantitativos resultantes de toda a rede;
- **Classificação**: identifica os nós da rede que têm conhecimento que se referem a classes de métodos, instrumentos e técnicas;
- **Especificação**: expressa os conhecimentos relativos à descrição de procedimentos, regras e heurísticas.

Com o objetivo de permitir a instanciação de redes de conhecimento e, ao mesmo tempo, o acesso a este conhecimento pelas ferramentas existentes no ambiente, solucionando a questão do comportamento associado aos elementos, definimos uma **L**inguagem para **D**escrição e **M**anipulação de **M**étodos (**LDMM**). A definição da LDMM foi realizada tomando-se por base métodos pertencentes a dois paradigmas de desenvolvimento. Utilizou-se, para sua construção, os métodos SADT [Ross85] e SSA [Gane82], representando o paradigma de desenvolvimento estruturado e os métodos OOA [Coad92] e Booch [Booch91], representando os métodos orientados a objetos. A partir do estudo destes métodos, foi possível identificar os papéis (funções) exercidos pelos elementos pertencentes a cada método em questão, que deveriam ser representados pela LDMM. Esta forma de representação permite que seja representada a semântica do documento gerado, e facilita o aproveitamento das informações geradas por uma ferramenta pelo ambiente. Foram identificadas as seguintes funcionalidades descritas na tabela I. A LDMM se compõe de dois blocos que são:

Bloco de Definições: permite a definição do conhecimento do método propriamente dito. É o responsável direto pela instanciação da classe Conhecimento;

Bloco de Consultas: permite a obtenção do conhecimento armazenado num objeto. É através dele que as ferramentas conseguem obter o conhecimento armazenado.

A definição da sintaxe, feita em BNF, onde o sinal ? representa o retorno de uma informação, está apresentada na figura 2. Em [Travassos 94] pode ser encontrada uma descrição dos métodos utilizados para a realização deste estudo e suas respectivas definições em LDMM.

Elemento	Função
DESCRITIVO	permite a descrição de alguma informação textual complementar que auxilie a compreensão do documento.
TRANSFORMADOR	recebe um determinado conjunto de informações, transforma-as de acordo com uma determinada lógica, e torna estas informações disponíveis para outros elementos.
FONTE	fornece informações para o sistema. A informação fornecida por uma fonte é sempre recebida por um transformador.
ARMAZENADOR	guarda as informações relevantes ao sistema, e fornece-as quando solicitado.
ORGANIZADOR	permite a organização de outros elementos. Pode ser utilizado para representar elementos que devem estar agrupados e apresentados de forma homogênea.
ENCAPSULADOR	possui propriedades de encapsular informações e funcionalidades.
HIERARQUIA	permite mostrar a relação hierárquica existente entre elementos encapsuladores.
COMPOSIÇÃO	permite relacionar elementos encapsuladores, demonstrando uma relação de composição (estruturação) entre eles.
ESTÍMULO	permite representar a ocorrência de algum evento (estímulo) que acontece no sistema.
DEPENDÊNCIA	permite representar uma relação de dependência entre elementos encapsuladores.
TRANSPORTADOR	transporta um conjunto de informações entre elementos transformadores, fonte, destino e armazenadores.
DESTINO	recebe informações do sistema.
ESTADO	permite a representação de algum estado em que outro elemento, ou o sistema, se encontra.

Tabela I - Elementos da LDMM

3.1.2 A Classe Frameworks

Segundo Wirfs-Brock [Wirfs-Brock90], a utilização de "frameworks" como elemento de definição e construção de modelos de interface com usuário tem sido realizada maciçamente a partir do trabalho de Krasner [Krasner88], que demonstrou a viabilidade de utilização de programação orientada a objetos para a construção da interface com o usuário. A utilização de "frameworks", entretanto, não é restrita à área de interface com o usuário. Podem ser aplicados em qualquer área de projeto de software [Johnson88], [Gossain89], [Jindrich90], [Madany89], [Russo89], [Zweig90], e permitem aumentar, nas linguagens orientadas a objetos, a capacidade de reutilização.

A formalização e conceituação de "frameworks" foi realizada no trabalho de Deutsch [Deutsch89] que destacou a importância da reutilização da interface dos "frameworks" e da especificação de seus componentes.

Um "framework" é projetado para ser refinado, ou especializado. Ele se constitui num conjunto de classes abstratas[1] e classes concretas[2], juntamente com seus relacionamentos, que representam o comportamento necessário para algum tipo de subsistema. Ao longo de sua

[1] classes abstratas são classes de alto nivel que, a princípio, não permitem a instanciação de objetos

[2] classes concretas são classes passiveis de instanciação

vida um "framework" pode evoluir, incorporando novas características e funcionalidades, a partir da inclusão de novas especializações na sua estrutura. A maturidade de um "framework" pode ser observada pelo número de classes concretas, especializadas a partir das classes abstratas, e incluídas na estrutura. Esta inclusão é bastante facilitada pela reutilização de toda a funcionalidade disponível nas classes abstratas superiores.

```
<Nome> ::= <Identificador> | <Identificador> <Nome>
<Identificador> ::= A | B | ... | Z
<Nome_Método> ::= <Nome>
<Nome_Documento> ::= <Nome>
<Nome_Componente> ::= <Nome>
<Método> ::=          Def_Método <Nome_Método>
                              <Definição_Documento>
                      Fim_Def_Método | &
<Definição_Documento> ::= Def_Doc <Nome_Documento>
                                  <Definições>
                      Fim_Def_Doc | <Definição_Documento> | &
<Definições> ::=      Def_Comp <Nome_Componente>
                              <Define_função>
                              <Define_forma>
                      Fim_Def_Comp | <Definições> | &
<Define_função> ::=   Def_Função <Função> | <Define_função> | &
<Define_forma> ::=    Def_Forma <Forma> | &
<Forma> ::= Importe <Nome_Primitiva> <Lista_Parâmetros>
<Nome_Primitiva> ::= <Nome>
<Lista_Parâmetros> ::= <Parâmetro> | <Parâmetro> <Lista_Parâmetros>
<Parâmetro> ::= <Nome> | &
<Função> ::= DESCRITIVO | TRANSFORMADOR | FONTE | DESTINO | ESTADO |
ARMAZENADOR | ORGANIZADOR | ENCAPSULADOR | HIERARQUIA | COMPOSIÇÃO | ESTÍMULO
| DEPENDÊNCIA | TRANSPORTADOR
<Seleções> ::=       Sel_Método <Nome_Método>
                              <Consultas>
                      Fim_Sel_Método | <Seleções> | &
<Consultas> ::=       Sel_Doc <Nome_Documento>
                              <Perguntas>
                      Fim_Sel_Doc | <Consultas> | &
<Perguntas>::= Qual_Forma {<Nome_Componente> | <Função>} ? <Forma> | Qual_Função
{<Nome_Componente> | <Forma>} ? <Função> | Qual_Comp      {<Função> | <Forma>} ?
<Nome_Componente> | &
<Aplicação> ::= <Método> | <Consulta>
```

Figura 2 - Sintaxe LDMM

Podem ser identificadas três atividades fundamentais quando se trabalha com "frameworks". Estas atividades, que determinam o tipo de pesquisa e trabalho a ser desenvolvido, relacionam-se a aspectos de projeto, uso e descrição de "frameworks". Projetar "frameworks" envolve a identificação de características comuns, dentre determinados domínios de aplicação. Estas similaridades representam, na maioria dos casos, subsistemas que compartilham funcionalidades e estruturas semelhantes. Um "framework" representa, então, uma generalização destes subsistemas, e permite que novos subsistemas sejam construídos a partir de sua definição. Utilizar "frameworks" envolve a definição, no caso de necessidade[3], de novas classes que devam ser incorporadas à estrutura já existente, e na configuração de um conjunto de objetos a partir do fornecimento de parâmetros para cada um

[3] Idealmente nenhuma nova classe seria necessária num "framework"

deles, permitindo sua posterior conexão ao modelo. Descrever "frameworks" envolve a definição de formalismos que permitam a perfeita descrição dos modelos. As linguagens orientadas a objetos convencionais, em geral, não fornecem nenhum suporte direto que permita a formalização da descrição. Uma linguagem orientada a objetos que pode ser utilizada para a descrição de "frameworks" é EIFFEL [Meyer92].

Adotando esta conceituação, a classe Frameworks sintetiza a idéia de se obter uma representação uniforme, e reutilizável, de estruturas organizacionais que permitam a construção de ambientes e ferramentas, no contexto da Estação TABA.

Existem duas especializações abstratas desta classe. A primeira diz respeito aos ambientes de desenvolvimento e representa o conjunto de informações e funcionalidades necessários num ADS. É identificada no modelo com o nome de FrameworkAmbientes. A segunda diz respeito às ferramentas existentes e que, possivelmente, podem estar integradas num ADS. É representada, no modelo, pela classe abstrata FrameworkFerramentas.

As especializações de mais baixo nível representam os ambientes e as ferramentas, existentes na Estação TABA e passíveis de utilização. A utilização de estruturas deste tipo simplificam o processo de instanciação de ambientes e de construção de ferramentas, fornecendo um conjunto de funcionalidades e características básicas, associadas aos elementos internos que constituem a Estação, que podem ser reutilizadas a cada novo desenvolvimento.

A **classe FrameworkAmbientes** contém a descrição das características comuns aos ADSs. A identificação das especificidades foi realizada a partir da definição de um conjunto de ADSs específicos para cada domínio de aplicação. A instanciação desta classe representa um ADS genérico, contendo um conjunto mínimo de funcionalidades que permita sua utilização. Isto é verdade se forem consideradas as atividades de planejamento e repetitivas[4], comuns a todo processo de desenvolvimento.

A **classe FrameworkFerramentas** reúne as características básicas e necessárias para que ferramentas sejam integradas aos ambientes instanciados, de acordo com o modelo para integração TABA. Na estação TABA são identificados dois tipos básicos de ferramentas: ferramentas internas e ferramentas externas.

As ferramentas internas são aquelas que foram construídas levando em consideração toda a estrutura TABA. Estas ferramentas estão incorporadas ao ambiente de forma natural, fazendo parte do mundo de objetos TABA e gerando informações que podem ser diretamente manuseadas no ambiente, e conforme a necessidade, reutilizadas por outras ferramentas. A forma de organização das informações será descrita na seção 3.2 quando for discutido o modelo dos ambientes instanciados TABA.

As ferramentas externas representam aquelas ferramentas desenvolvidas sem utilizar a estrutura TABA. Quando é necessária sua integração a um ambiente, estas ferramentas são encapsuladas na estrutura TABA, e passam a fazer parte do mundo de objetos TABA. O grau de integração destas ferramentas aos ambientes depende da forma como se consegue realizar o tratamento das informações geradas pela ferramenta no contexto do ambiente. Para isto pode ser utilizado na integração da ferramenta externa um tradutor ("driver"), que permita a transcrição das informações da ferramenta para o ambiente e do ambiente para a ferramenta. Todo o controle de ativação destas ferramentas passa a ser feito pelo ambiente instanciado, permitindo o acompanhamento de sua utilização e sua participação nas atividades do processo de desenvolvimento.

---

[4] "clerical activities"

3.2 O Modelo dos Ambientes Instanciados TABA

O meta ambiente TABA é o responsável pela definição e instanciação do ADS. Estes ambientes, por sua vez, são compostos de ferramentas, sejam elas internas ou externas, que necessitam trabalhar de forma integrada. Esta integração pode ser obtida conforme mostra o diagrama de assuntos apresentado na figura 3, que descreve a seguinte situação, onde as classes encontradas no modelo estão escritas em *itálico*:

> Um *ADS* existe para o desenvolvimento de algum *projeto*. Este *projeto*, por sua vez, pode ser desenvolvido por *equipes*, que são compostas por *pessoas*, responsáveis ou não, pelas *atividades* ao longo do processo de desenvolvimento. Cada *atividade*, ao longo do processo, gera um conjunto de *documentos*. Estes documentos, que representam informações especificas do processo, são compostos por *grafos* manuseados pelas *ferramentas*, construidas para terem *conhecimento* sobre um determinado método de desenvolvimento. A forma de comunicação destas ferramentas com o usuário é determinada pela *interface com o usuário*, que é comum a todas as *ferramentas*, e ao próprio *ADS*.
>
> O usuário pode solicitar, ao longo do processo de desenvolvimento, alguma assistência ao *ADS*, que possui um *assistente inteligente* especifico para o domínio da aplicação considerado pelo ADS.
>
> A ordem de execução das *ferramentas* está relacionada à ordem de acontecimento das *atividades*, ao longo do processo de desenvolvimento. Cabe ao ADS certificar a ocorrência e término das *atividades*, e efetivar a manutenção das informações necessárias à realização de ensaios e medidas, a respeito da execução da atividade, que permitam um planejamento futuro mais adequado.

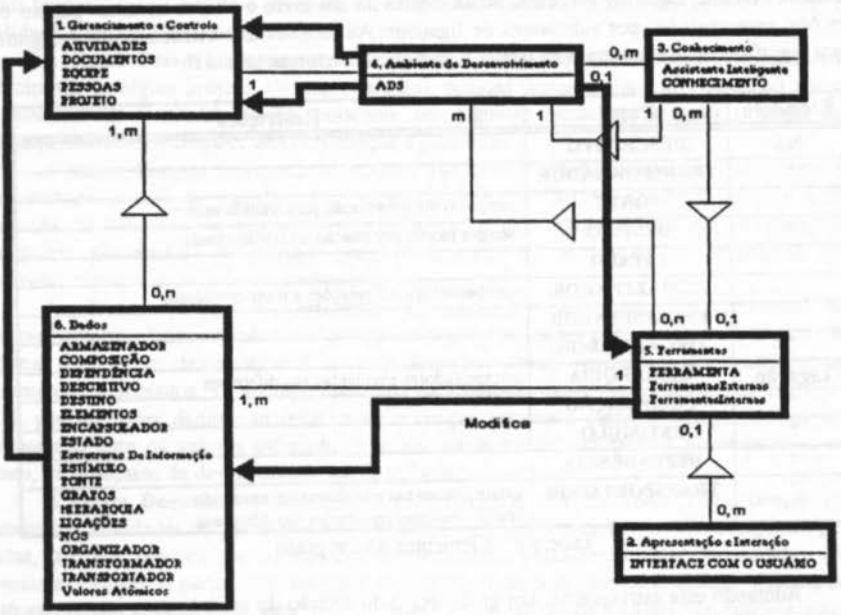

Figura 3 - Diagrama de Assuntos dos Ambientes Instanciados TABA

Adotando este enfoque de trabalho as classes necessárias para representar os elementos existentes num ambiente instanciado TABA foram organizadas de acordo com a figura 3. Neste modelo encontramos as seguintes classes:

A **classe GRAFOS** representa a estrutura de representação da informação do modelo para integração de ferramentas da Estação TABA. Este tipo de organização da informação baseia-se num modelo de grafos, onde o conhecimento necessário para a descrição e representação de cada elemento pertencente a um determinado método é descrito a partir de uma linguagem específica e armazenado numa base de conhecimento. A estrutura de grafos é representada através da utilização do paradigma de orientação a objetos, permitindo que o comportamento desta estrutura reflita as modificações sofridas ao longo do processo de desenvolvimento de um produto de software. Outros ambientes [Jino85], [Giavitto89], [Kiesel93] utilizam esta forma de organização da informação, sem se preocupar, entretanto, em representar o conhecimento associado.

Um grafo é composto por *Elementos*. Um elemento de um grafo é reconhecido no ambiente através da atribuição de seu *tipo* e *identificação*. A estrutura de tratamento de informação declarada e composta desta forma define então um *grafo tipado*. O *tipo* do elemento identifica o conjunto de funções primitivas necessárias à representação de seu comportamento, ao mesmo tempo que qualifica *subclasses* da classe primitiva *Elementos* que existam no ambiente. Os *Elementos* de um grafo podem ser de dois tipos: *nós* e *ligações*. *Nós* representam os vértices do grafo e podem assumir papéis, ou funções, representados por subclasses de nós. *Ligações* representam as arestas de um grafo e podem assumir papéis, ou funções, representados por subclasses de ligações. As funções dos elementos de um grafo, cujos significados se encontram na tabela I, podem ser vistos na tabela II.

Elemento	Papel	Restrições
Nó	DESCRITIVO	
"	TRANSFORMADOR	
"	FONTE	sempre envia informação para transformador
"	DESTINO	sempre recebe informação do transformador
"	ESTADO	
"	ARMAZENADOR	recebe/entrega informações a transformadores
"	ENCAPSULADOR	
"	ORGANIZADOR	
Ligação	HIERARQUIA	encapsuladores envolvidos são diferentes
"	COMPOSIÇÃO	
"	ESTÍMULO	
"	DEPENDÊNCIA	
"	TRANSPORTADOR	existe sempre um transformador envolvido transformadores envolvidos são diferentes

Tabela II - Elementos de um grafo

Adotando esta estruturação, um grafo, segundo a visão do modelo para integração de ferramentas TABA, pode ser identificado por um conjunto G, definido como:

$G = \{E\}$

onde $E = \{ N, L \}$, e

$N = \{ N_1, N_2, ..., N_i \}$, $i >= 1$; e

$$L = \{ (N1, N2), \ldots, (Ni, Nj) \};\ i >= 1,\ j >= 1 ;$$
$$\text{ou } L = \{ \} ;\ i = 0, j = 0;$$

onde **N** representa o conjunto não vazio formado pelos **nós** N_i e **L** representa as ligações (N_i, N_j) existentes entre o nó N_i e o nó N_j, nesta ordem. Este tipo de estrutura é denominado **grafo direcionado (dígrafo)**. A possibilidade da existência de uma **ligação** entre um mesmo nó (laço) é garantida através da condição i = j associada à ligação (N_i, N_j). Além disto, pode-se observar que, a um **nó** é dada a possibilidade de se relacionar com mais de um **nó** a partir da condição **i >= 1, j >= 1**.

Considerando as extensões realizadas na estrutura de grafos definimos a estrutura utilizada no modelo de integração como sendo um **dígrafo tipado direcionado configurável**. Adotaremos a partir deste ponto, apenas, o nome **grafo** para definir uma estrutura deste tipo.

Um documento **D**, produzido numa fase **f**, é composto pelos vários **grafos** gerados nesta fase. Assim podemos definir : $D_{i,f} = \bigcup G_{k,f}$, $f > i >= 0$, $k >= 0$. O conjunto de documentos pertencentes a uma determinada atividade do processo de desenvolvimento constitue o subproduto gerado nesta atividade e é identificado por: $P_f = \bigcup D_{i,f}$, $f > i >= 0$. Uma fase que ainda não tenha sido desenvolvida é satisfeita pela relação: $P_f = D_{0,f}$.

Estas relações podem ser utilizadas pelo ADS para a verificação do desenvolvimento de alguma atividade e o controle de ativação das ferramentas para a execução de atividades subsequentes do processo de desenvolvimento.

A **classe Equipe** descreve as características das equipes envolvidas no desenvolvimento de algum produto de software (projeto). Estas equipes são compostas por várias pessoas, com o objetivo de cumprir o melhor possível as estimativas de prazo, custo e qualidade atribuídas para um determinado projeto.

A **classe Pessoas** representa os integrantes das equipes de desenvolvimento que participarão de algum projeto. De maneira geral, pessoas desempenham atividades ao longo do processo de desenvolvimento, podendo, em algumas situações, assumir o papel de liderança, assumindo funções de coordenação e gerência.

A **classe Projeto** representa o produto que está sendo desenvolvido. Um projeto é desenvolvido através do trabalho de equipes de desenvolvimento, apoiadas por um ADS integrado. O processo de desenvolvimento ocorre com a execução de um conjunto de atividades, planejadas de acordo com o domínio da aplicação, o paradigma de desenvolvimento e o ciclo de vida adotados.

A **classe Atividades** representa as atividades ao longo do processo de desenvolvimento. Estas atividades são, normalmente, realizadas por pessoas, integrantes de alguma equipe de desenvolvimento, e podem ter, como resultado, um conjunto de documentos associados. O acompanhamento da execução destas atividades, por parte do ADS, permite obter dados concretos sobre o tempo, esforço e custo, o que possibilita sua comparação com os valores estimados e, consequentemente, um melhor planejamento, no futuro, de processos de desenvolvimento de software.

A **classe Documentos** descreve as características dos documentos que devem ser gerados nas atividades do processo de desenvolvimento. Estes documentos são compostos por grafos, que por sua vez são construídos e manuseados pelas ferramentas, possuindo um formato associado. A partir deste formato, o Engenheiro de Software pode definir a forma de composição dos documentos, garantindo, a partir da utilização de padrões pré-estabelecidos, a qualidade ao longo do processo de desenvolvimento.

A **classe ADS** representa um ambiente que tenha sido definido, instanciado e testado pelo meta-ambiente TABA. A descrição desta classe representa uma instância da classe *FrameworkAmbientes*.

A **classe Ferramentas** representa as ferramentas disponíveis na Estação TABA, e que se encontram integradas ao ADS. A descrição desta classe representa uma instância da classe *FrameworkFerramentas*.

A **classe Estruturas de Informação** representa as informações que são transportadas pelos *Transportadores*. Estas estruturas se prestam à descrição de agregados de informações tratados pelo produto de software que está sendo desenvolvido a partir da utilização do ADS. Na composição de estruturas de informação são utilizados *valores atômicos*, que representam as informações elementares no contexto do produto.

Estas classes possibilitam a instanciação de ADSs que compartilham de características comuns, acrescentadas quando da instanciação do ADS pela Estação TABA, responsáveis pela integração do ambiente. Estas características, que se encontram distribuídas entre as classes, fornecem recursos para que as ferramentas, internas ou externas, entendam o significado e a representação das informações geradas ao longo do processo de desenvolvimento.

4. Conclusão

Este artigo descreveu o modelo para integração de ferramentas utilizado na Estação TABA. Através deste modelo é possível a instanciação dos ambientes especificados pelo meta-ambiente TABA, a construção de novas ferramentas na Estação e a integração de ferramentas externas desenvolvidas fora da Estação.

Um protótipo da Estação TABA, que valida o modelo, está implementado em O2 [$O_2$93], executável em estações de trabalho Sun Sparc. Neste momento, trabalhos estão sendo realizados no sentido de estender a representação das classes "Frameworks" e a implementação final da Estação TABA utilizando a linguagem EIFFEL [Meyer92]. Uma descrição completa do modelo de integração e do protótipo pode ser encontrada em [Travassos94].

Referências

Aguiar92 T.C. Aguiar; Um Sistema Especialista de Suporte à Decisão para Planejamento de Ambientes de Desenvolvimento de Software, Tese de Doutorado, Programa de Engenharia de Sistemas e Computação, COPPE/UFRJ, março 1992

Booch91 G. Booch; *Object Oriented Design with Applications*, The Benjamin Cummings Publishing Company,Inc., 1991

Brown92 A.W. Brown, A.N. Earl, J.A. McDermid; *Software Engineering Environments: Automated Support for Software Engineering*, McGraw-Hill Book Company, 1992

Chen92 M. Chen and R.J. Norman; "A Framework for Integrated CASE", IEEE Software, Março 1992

Coad92 P. Coad e E. Yourdon; *Análise Baseada em Objetos*, Editora Campus, 1992

Deutsch89 L.P. Deutsch; " Design reuse and frameworks in the Smalltalk-80 system", Software Reusability, Vol. II, T.J. Biggerstaff and A.J. Perlis, eds. ACM Press, 1989

Duarte91a M. A. Duarte; Um Sistema para representação do Conhecimentode Métodos de Desenvolvimento de Software, Tese de Mestrado, COPPE/UFRJ, Programa de Engenharia de Sistemas e Computação, Janeiro 1991

Duarte91b M.A. Duarte, A.R.C. da Rocha, G.H. Travassos, J.C.C. Pena; Um Sistema de Representação do Conhecimento Adequado a Métodos de Desenvolvimento de Software, Relatórios Técnicos do Projeto TABA, TABA RT-6/91, Programa de Engenharia de Sistemas e Computação, COPPE/UFRJ, 1991

Ecma90 ECMA - European Computer Manufacturers Association. A Reference Model for Frameworks of Computer-Assisted software Engineering Environments. ECMA TR/55, Dezembro 1990

Gane82 C. Gane and T. Sarson; *Structured Systems Analysis*, McDonnell Douglas, 1982

Giavitto89 J. Giavitto, A. Devarenne,G. Rosuel,Y. Holvoat; "ADAGE: Utilisation de la Genericite pour Construire des Environnements Adaptables", Laboratories de Marcoussis, Centre de recherches de la CGE, Marcoussis, França, 1989

Gossain89 S. Gossain and D.B. Anderson; "Designing a class hierarchy for domain representation and reusability", Proceedings of Tools'89, Paris, França, Novembro 1989

Jindrich90 W.A. Jindrich; "FOIBLE: A framework for visual programming languages", Master's Thesis, University of Illinois at Urbana-Champaign, 1990

Jino85 M. Jino, et Alli; "SIPS - An Extensible and Integrated Environment for Software Development and Production", Centro Tecnológico para Informática, Instituto de Automação, 1985, Campinas, SP

Johnson88 R.F. Johnson et alli; "TS: An optimizing compiler for Smalltalk", Proceedings of OOPSLA'89, SIGPLAN Notes, Vol. 23, No. 11, San Diego, Califórnia, Setembro 1988

Kiesel93 N. Kiesel, A. Schürr, B. Westfechtel; "GRAS, a Graph-Oriented Database System for (Software) engineering Aplications", IEEE CASE'93 - Proceedings Sixth International Workshop on Computer-Aided Software Engineering, Singapore, Julho 19-23, 1993

Krasner88 G.E. Krasner, S.T. Pope; "A cookbook for using the model-view-controller user interface paradigm in Smalltalk-80", Journal of Object-Oriented Programming 1, 3 (Agosto/Setembro 1988)

Madany89 P.W. Madany et alli; "A Class Hierarchy for Building Stream-Oriented File Systems", Proceedings of the 1989 European Conference on Object-Oriented Programming, Nottingham, UK, 1989

Meyer92 B. Meyer; "Applying "Design by Contract" ", IEEE-COMPUTER, Outubro 1992, pp.40-51

Meyers91 S. Meyers; "Difficulties in Integrating Multiview Development Systems", IEEE Software, January 1991

Mi92 P. Mi and W. Scacchi; "Process Integration in CASE Environments", IEEE Software, Março 1992

$O_2$93 O_2 Technology; The O_2 User Manual, Version 4.3, Released Julho 1993, Versailles Cedex, França

Penedo92 M.H. Penedo, A. Karrer, C.Shu; "A Survey of Software Engineering Environment Architectural Approaches", Technical Report Series, TRW, Arcadia-TRW-90-004-R1, Julho 1992

Russo89 V. Russo and R.H. Campbell; "Virtual Memory and Backing Storage Management in Multiprocessor Operating Systems using Class Hierarquical Design", Proceedings of OOPSLA'89 SIGPLAN Notes, Vol. 24, No. 10, New Orleans, Louisiania, Setembro 1989

Rocha90 A.R.C. da Rocha; T.C. Aguiar; J.M. Souza; "TABA: a heuristic workstation for software development"; COMPEURO'90; Tel Aviv, Israel, maio 1990

Ross85 D.T. Ross; "Aplications and Extensions of SADT", IEEE-Computer, Abril 1985

Thomas92 I. Thomas and B.A. Nejmeh; "Definitions of Tool Integration for Environments", IEEE Software Março 1992

Travassos94 Travassos, G.H. O Modelo de Integração de Ferramentas da Estação TABA . Tese de Doutorado, COPPE/UFRJ, março 1994.

Wirfs-Brock90 R.J. Wirfs-Brock, R.E. Johnson; "Surveying Current Research in Object-Oriented Design", Communications of the ACM, Vol. 33, No. 9, Setembro 1990

Yang93 Y.Yang, J.Welsh and W. Allison; "Supporting Multiple Tool Integration Paradigms within a Single Environment", IEEE CASE'93 - Proceedings Sixth International Workshop on Computer-Aided Software Engineering, Singapore, Julho 19-23, 1993

Zweig90 J. Zweig and R. Johnson; "Conduits: A communication abstraction in C++", USENIX C++ Conference, 1990