O Uso de "Frameworks" como Arquiteturas Reutilizáveis na Construção de Ambientes de Desenvolvimento de Software

Alberto Mello de Cima

Cláudia Maria Lima Werner

Guilherme H. Travassos

UFRJ - COPPE/Programa de Engenharia de Sistemas

Centro de Tecnologia - Bloco H - Sala H319

Cidade Universitária - Ilha do Fundão

21949-900 Rio de Janeiro RJ

e-mail: werner@cos.ufrj.br

Resumo

Os "frameworks" constituem uma emergente tecnologia com grande potencial para aplicações nas áreas de análise de domínio e reutilização de software. Neste artigo, tal potencial é analisado, ao levantar-se questões relacionadas ao projeto de "frameworks" no contexto de Ambientes de Desenvolvimento de Software.

Abstract

"Frameworks" constitute a new and powerful technology for applications in areas such as domain analysis and software reuse. In this paper, the modeling power of frameworks is analysed, discussing issues related to their design in the context of software development environments.

[Keywords]: "frameworks", ambientes de desenvolvimento de software, reutilização de software, análise orientada a objeto

1. Introdução

A reutilização traz como principais motivações o aumento da produtividade e da qualidade do software [TRAC88]. Este aumento da qualidade é uma consequência do reuso de componentes que já foram documentados, muito bem testados e aprovados, tanto pela equipe desenvolvedora quanto pelo usuário nas reais condições de uso. Já o aumento da produtividade é decorrente da redução do tempo de desenvolvimento necessário, uma vez que já existem partes do sistema "prontas", da redução da quantidade de documentação a ser criada e de testes a serem realizados e, ainda, da necessidade reduzida de modificações no sistema devido ao aumento da qualidade deste.

Este aumento de produtividade tem, também, uma consequência que incentiva bastante a aplicação das tecnologias de reutilização: a redução dos custos do software. Outra forma de redução de custos que motiva a reutilização decorre do reuso de arquiteturas genéricas (denominadas "frameworks"), que formalizam em um modelo orientado a objeto as informações levantadas nas atividades da "análise de domínio", na qual trabalham profissionais altamente especializados e muito mais caros que o restante da mão-de-obra necessária ao desenvolvimento do software.

Basicamente, a análise de domínio é um processo de identificação e organização de conhecimento sobre uma determinada classe de problemas (o domínio do problema) para suportar a descrição e solução dos mesmos [PRIE91]. Os "frameworks" são construídos, durante este processo e podem ser encarados como modelos formais de um domínio de problema. Esta formalização do conhecimento aumenta a confiabilidade do resultado final da análise e facilita o gerenciamento de informações de alto nível para reutilizações futuras. Segundo Rebecca Wirfs-Brock e Ralph Johnson [WIRF90], um "framework" pode ser considerado como um projeto ou arquitetura de alto nível, consistindo de classes que são especialmente projetadas para serem refinadas e usadas em grupo.

Baseado no trabalho desenvolvido em [TRAV94], este artigo discute o uso de "frameworks" dentro do domínio dos Ambientes de Desenvolvimento de Software (ADS), visando-se a construção de modelos formais para um ADS genérico com um conjunto mínimo de funcionalidades que permita a sua instanciação.

O artigo está organizado em cinco seções, além desta introdutória. Na segunda seção, é feita uma apresentação mais detalhada da tecnologia de "frameworks" e da nomenclatura utilizada no projeto dos mesmos. Na terceira seção, são identificados os requisitos do domínio de Ambientes de Desenvolvimento de Software e da Estação TABA. Na quarta seção, é feito um estudo sobre o projeto de "frameworks" para ADSs. Finalmente, na quinta seção, são apresentadas as conclusões sobre o trabalho desenvolvido.

2. A Tecnologia dos "Frameworks"

A formalização e conceituação de "frameworks" foi realizada no trabalho de Deutsch [DEUT89] que destacou a importância da reutilização da interface dos "frameworks" e da especificação de seus componentes.

O projeto de um "framework" é por si só uma atividade de pesquisa [WIRF90]. O projetista deve entender as decisões de projeto e organizá-las em um conjunto de classes abstratas (classes de alto nível que, a princípio, não permitem a instanciação de objetos) e concretas (classes passíveis de instanciação), juntamente com seus relacionamentos, que representam o comportamento básico para um subsistema. Logo, o projetista desenvolve uma teoria do domínio do problema e a expressa segundo um modelo orientado a objetos.

O "framework" é projetado para ser refinado, ou especializado. Por isso, um grande problema a ser enfrentado na criação de um "framework" é que seu projetista deve procurar soluções que cubram inteiramente um domínio de problema e implementá-las de uma forma extensível e reutilizável, ou seja, este projetista deve ser capaz de perceber a aproximação do ponto de equilíbrio entre a generalidade e a especialidade do "framework" para o domínio desejado.

Outro problema inerente a criação de um "framework" é de natureza técnica, pois a maioria das linguagens de programação orientadas a objeto não dão suporte direto à criação de classes abstratas e utilização de subsistemas [WIRF90]. É neste contexto que surge a força e a importância de uma linguagem/ambiente como Eiffel [ISE90], que oferece mecanismos para a implementação desses conceitos.

A linguagem/ambiente Eiffel foi criada por Bertrand Meyer, em 1985, e desenvolvida pela "Iteractive Software Engineering Inc.- ISE". Atualmente, Eiffel é considerada uma das linguagens orientadas a objeto mais completas, uma vez que encerra facilidades para implementação de todos os conceitos envolvidos no paradigma.

As facilidades disponíveis em Eiffel que encorajam o projeto de "frameworks" são os mecanismos de herança (simples e múltipla), a possibilidade de construção de classes genéricas e abstratas e ainda a filosofia de contratos, que permite a modelagem do comportamento das classes, através da definição de pré-condições, pós-condições e condições invariantes para seus métodos [ISE90].

As classes genéricas e abstratas têm conceitos muito parecidos que podem ser, algumas vezes, confundidos. As classes abstratas definem o comportamento que é compartilhado por algumas classes, ou seja, fornece as características reutilizáveis para as suas sub-classes, servindo como uma especificação mínima para estas [WIRF90], enquanto que as genéricas não se preocupam com a definição do comportamento a ser compartilhado pelas respectivas sub-classes, mas com a possível reutilização, por estas, de elementos e características comuns já especificadas (e implementadas).

Normalmente, tanto as classes abstratas quanto as genéricas são criadas para serem especializadas através dos mecanismos de herança, não podendo, na maioria das vezes, ser instanciadas.

3. Ambientes de Desenvolvimento de Software e a Estação TABA

O conceito de Ambientes para Engenharia de Software surgiu nos anos 70, buscando combinar técnicas, métodos e ferramentas com o objetivo de prover um meio através do qual o Engenheiro de Software pudesse ter um apoio ao construir produtos de software [PENE88]. Esta forma de tratar o processo de desenvolvimento, como se entende hoje, envolve o suporte a atividades individuais e ao trabalho em grupo, ao gerenciamento do projeto, ao aumento da qualidade geral dos produtos, ao aumento da produtividade, permitindo ao Engenheiro de Software acompanhar o trabalho e medir, através de informações obtidas ao longo do desenvolvimento, a evolução dos trabalhos.

Inicialmente estes ambientes se concentravam, apenas, nas fases de codificação e projeto detalhado, não possuindo suporte às fases iniciais do ciclo de desenvolvimento [ROTH75], [DOLO78], [GOLD83]. No final dos anos 70, viu-se uma evolução no entendimento do processo de desenvolvimento de software. O foco se moveu da fase de codificação para as fases de especificação e projeto e começou-se a entender que desenvolver software era uma ciência e não uma arte e que carecia de processos mais sistemáticos e disciplinados de trabalho [SMIT90].

Com a popularização dos computadores de uso pessoal, na metade dos anos 80, viu-se a introdução dos ambientes automatizados e de ferramentas CASE ("Computer Aided

Software Engineering") que visavam tornar mais prático e efetivo o uso dos métodos para o desenvolvimento de software [CHIK88], [MOSL92], [VESS92]. Atualmente, estamos num estágio onde, após ter-se verificado que ferramentas isoladas podem oferecer apenas soluções parciais, o que se deseja é utilizar ferramentas de apoio ao desenvolvimento de software ao longo de todo o seu processo de desenvolvimento [SMIT90]. Surge, então, o conceito de Ambientes de Desenvolvimento de Software (ADS) como o entendemos hoje:

> *"um sistema que fornece suporte a qualquer atividade executada por engenheiros de software* " [SAIT89]

> "- *um ciclo de vida que define as etapas do processo de desenvolvimento e as atividades a serem realizadas em cada etapa;*
>
> - *um conjunto de métodos, usados para organizar o pensamento e o trabalho do usuário ao longo do processo de desenvolvimento, e;*
>
> - *um conjunto de ferramentas que automatizam os métodos*" (Projeto TABA, (ROCH87))

A evolução das ferramentas CASE e dos ambientes de desenvolvimento de software mostrou que não é qualquer ferramenta ou ambiente que é realmente útil ao usuário desenvolvedor de software. Dentre as características [TRAV94] que tornam um ADS útil, temos:

- **ser customizável**: o ambiente deve permitir adaptações que o tornem mais adequado a um determinado projeto e às preferências de seus usuários.
- **oferecer suporte à construção de ferramentas.**
- **reutilizabilidade de componentes internos**: os componentes que determinam a estrutura do ambiente devem poder ser reaproveitados em outros trabalhos e atividades.
- **fornecer apoio para o controle de versões e gerenciamento total de configuração.**
- **poder ser amplamente aplicado**: o ambiente deve poder ser utilizado no desenvolvimento de um vasto domínio de aplicações e, ainda, suportar o desenvolvimento de software tanto em pequena como em larga escala.
- **possuir uma interface gráfica consistente:** a interação do usuário com o ambiente deve ser feita de forma consistente, tendo-se uma apresentação uniforme e utilizar-se de recursos gráficos, preferencialmente, os mais avançados possíveis.
- **possuir uma interface interna consistente** de forma a permitir que uma ferramenta possa se comunicar perfeitamente com outra, possibilitando a passagem de informações entre elas, e permitindo também que ferramentas possam ser facilmente introduzidas e/ou substituídas no ambiente.
- **oferecer suporte baseado em conhecimento**, sobre diferentes domínios de aplicação.
- **oferecer suporte a todas as atividades do processo de desenvolvimento**: o ambiente deve apoiar no desenvolvimento do produto ao longo de todo o processo de desenvolvimento, e não apenas em etapas específicas.

- **ser extensível**, isto é, o ambiente deve ser aberto, permitindo que novas ferramentas lhe sejam incorporadas na medida em que forem necessárias. Assim sendo, o ambiente não deve possuir uma coleção fixa de ferramentas e sim permitir um enriquecimento progressivo a partir da inclusão de novas ferramentas ao longo do tempo.

O estágio atual de evolução para ADSs envolve o conceito de meta-ambientes, que são ambientes capazes de gerar, por meio de configuração, outros mais específicos. No sentido de construir um ADS configurável para Engenharia de Software, surgiu na COPPE/UFRJ o Projeto TABA [ROCH87], [ROCH90]. A motivação para o projeto é prover desenvolvedores de software com ambientes de desenvolvimento 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 [AGUI92], foi realizada e está operacional em ambiente Sun.

- **Ambiente para Construção de Ferramentas**

Os componentes disponíveis na Estação e, portanto considerados por XAMÃ, foram, em algum momento, incorporados à mesma. Em algumas situações, entretanto, pode ocorrer que a especificação do 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[1]. 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.

Em [TRAV94], que define o modelo de integração de ferramentas da Estação TABA, foi tratada a questão do uso de "frameworks" em Ambientes de Desenvolvimento de Software. Esta abordagem é discutida na próxima seção dentro do contexto do projeto de "frameworks" e de sua evolução no domínio de Ambientes de Desenvolvimento de Software.

4. O Projeto e o Uso de "Frameworks" no contexto de ADSs

Segundo R.E. Johnson [JOHN88], um projeto de software é usualmente descrito em função de seus componentes e do modo como estes iteragem. Por isso, no "framework", cada elemento principal do domínio do software é representado por uma classe (abstrata ou concreta), ou por um atributo, e a iteração entre classes é feita através de mensagens, objetivando-se a construção de uma arquitetura genérica que possibilite a reutilização numa granularidade maior que a de classes independentes.

Com base no contexto de ADS, descrito na seção 3, podem ser identificados, como exemplo inicial, elementos principais do domínio, tais como: o próprio ambiente, o conjunto de ferramentas a este integradas, uma interface padrão com o usuário, o assistente especialista, os métodos e o ciclo de vida de desenvolvimento do software.

Cada um destes elementos pode ser descrito através de classes abstratas, que incorporam o comportamento genérico do domínio. Este comportamento genérico é representado pela existência das próprias classes e através de uma definição inicial para os atributos e métodos destas classes, onde:

---

[1] Neste contexto o meta-usuário é o Engenheiro de Software

- Cada atributo pode ser concreto ou ser representado por uma classe abstrata. No segundo caso, o atributo deve, na especificação, tornar-se um objeto cuja classe seja herdeira da classe abstrata inicial, afim de conservar o comportamento do domínio.

- Quanto aos métodos, a implementação de seus respectivos algoritmos pode ser realizada somente na especificação da classe, caso represente um comportamento específico. Entretanto, os métodos podem ter definidos os tipos de seus parâmetros as pré e pós-condições, que especificam os limites dos valores destes parâmetros, ou estados do objeto, para o início e final da execução de seu algoritmo, e ainda procedimentos para o tratamento de casos onde tais condições não são satisfeitas.

Um exemplo de "framework" para ADSs, que representa os elementos de domínio já identificados, assim como seus principais atributos e serviços, é apresentado no modelo orientado a objeto da figura 1 (acima da linha tracejada). Tal modelo utiliza a notação proposta por P.Coad e E. Yourdon [COAD91].

Todas as classes que compõem o "framework" da figura 1 são abstratas. Exemplos de comportamentos do domínio dos ADSs representados por estas são:

- O modelo genérico de interface com o usuário que permite a uniformidade desta para todas as ferramentas do ambiente e para o próprio ambiente.
- A padronização das atividades inerentes ao desenvolvimento de software, representadas pelos métodos da classe *Frame Ambientes*. Como o modo de realização destas atividades depende do paradigma e dos métodos a serem utilizados pelo ambiente concreto, os métodos de *Frame Ambientes* não são implementados no nível do "framework". A mesma observação serve aos métodos da classe *Frame Ferramenta*.
- A possibilidade de integração de um assistente especialista ao ambiente. Este assistente deve ser, no entanto, especificado mais tarde, se assim desejar o desenvolvedor do ambiente concreto.
- Um modelo para construção de ferramentas no ambiente, como o proposto em [TRAV94]. Este modelo é representado pelas classes *Frame Ferramenta*, *Ferramenta Interna* e *Ferramenta Externa*.

Para se criar um ambiente real a partir do "framework" apresentado, deve ser cumprido um processo transitório de especialização deste último. Segundo R.Wirfs-Brock [WIRF90], este processo pode ser realizado através de duas formas básicas:

- Definição ou especificação de classes e subclasses necessárias e/ou;
- Configuração do conjunto de objetos existentes, através do fornecimento de parâmetros a cada uma das conexões entre estes.

As duas formas de especialização de "frameworks" são discutidas com mais detalhes nas subseções 4.1 e 4.2.

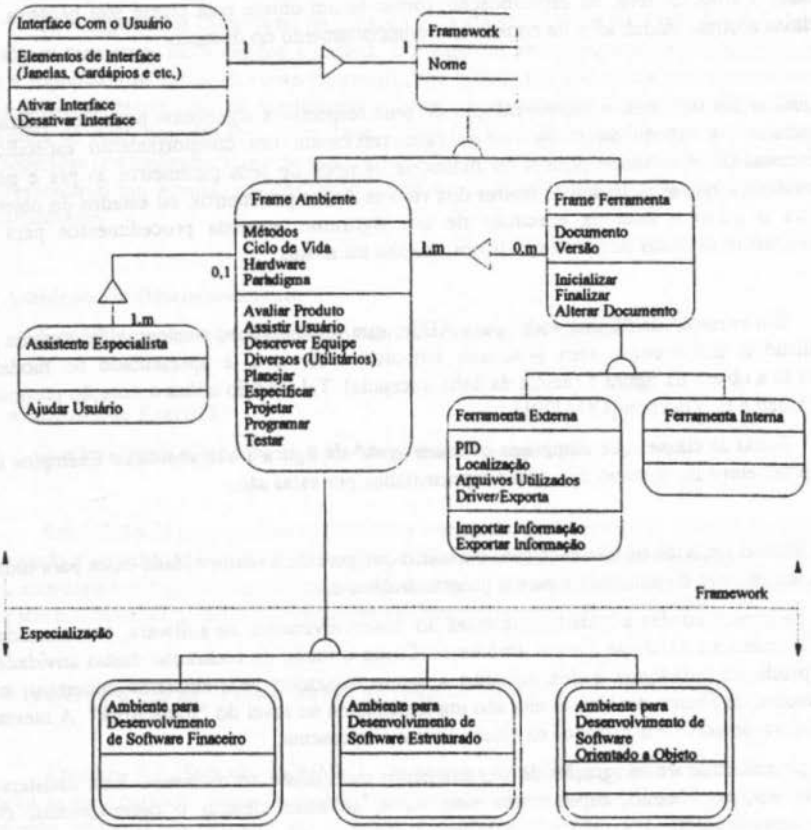

Figura 1 - Exemplo de "framework" para Ambiente de Desenvolvimento de Software e possíveis especializações

4.1 Especificação por definição das classes e subclasses necessárias

Nesta primeira forma de especialização, o comportamento específico da aplicação é inserido na arquitetura genérica, somando-se métodos às subclasses de uma ou mais classes do "framework". Cada método somado a uma subclasse deve estar de acordo com as convenções da respectiva superclasse. Os "frameworks" que fazem uso deste processo de especialização obrigam quem os utiliza a conhecer os seus dispositivos internos, sendo portanto chamados de "framework caixa-branca" [JOHN88].

O "framework" apresentado na figura 1 é do tipo "caixa-branca". Alguns exemplos simplificados de especializações para este "framework" são apresentados na própria figura 1,

abaixo da linha tracejada. Entretanto, uma especialização real deve conter subclasses que especializem também as demais classes abstratas, herdando destas o comportamento genérico ao domínio dos ADS e integrando-se na construção do ambiente real.

Embora seja bastante flexível quanto a sua construção, o "framework caixa-branca" requer a criação de muitas subclasses na sua especialização. Apesar de simples, a quantidade de subclasses a serem criadas pode tornar difícil a compreensão do projeto, dificultando qualquer tipo de manutenção no futuro [JOHN88].

4.2 Especificação pela configuração do conjunto de objetos existentes

Nesta segunda forma de especialização, o "framework" recebe um conjunto de parâmetros que representa o comportamento específico da aplicação. Este fornecimento de parâmetros é feito por protocolo através da interface de cada classe do "framework" e, por isso, esta forma de especialização requer que seja conhecida apenas esta interface. Consequentemente, o tipo de "framework", assim especializado, passou a ser conhecido como "framework caixa-preta".

Um exemplo de "framework caixa-preta" para ADS é apresentado na figura 2. O meta-ambiente da estação TABA utiliza um "framework" deste tipo no processo de instanciação de um ADS, que é realizado a partir da determinação de parâmetros tais como a plataforma, o ciclo de vida, os métodos e o paradigma utilizados. De posse de tais parâmetros, o "framework" concretiza o ADS desejado, utilizando as informações armazenadas e organizadas (segundo uma rede semântica) na classe&objeto *Conhecimento* [TRAV94].

Segundo R. Johnson [JOHN88], todo "framework caixa-branca" evolui idealmente para um "caixa-preta". Entretanto, é difícil afirmar como esta evolução ocorrerá, ou se ela realmente ocorrerá. No contexto de ADSs, verifica-se que realmente esta evolução ocorre. O "framework caixa-branca" deve ser encarado como um estágio natural de evolução de um sistema, embora em algumas situações este seja um produto final.

Tanto o "framework caixa-branca" como o "caixa-preta" favorecem o processo de reutilização. Entretanto, R.Johnson afirma que o último facilita mais a reutilização que o primeiro, pois exige que apenas a sua interface seja compreendida, podendo ser portanto encarado como um grande componente que encapsula o comportamento genérico de um domínio. Esta visão é de grande interesse para aqueles desenvolvedores de software que desejam apenas reutilizar o "framework" dentro das possibilidades absorvidas por este. Por outro lado, os "frameworks caixa-branca" são mais flexíveis e permitem sua utilização na especificação de aplicações até então desconhecidas dentro do domínio.

Uma boa estratégia para o projeto de um "framework" é iniciá-lo com a abordagem de "caixa-branca", pois esta, devido a sua informalidade interna, torna os "frameworks" mais fáceis de serem contruídos. A medida que o "framework" é projetado e usado, o acúmulo de conhecimento sobre o domínio alvo e aplicações específicas permite sua evolução para a abordagem de "caixa-preta" [JOHN88].

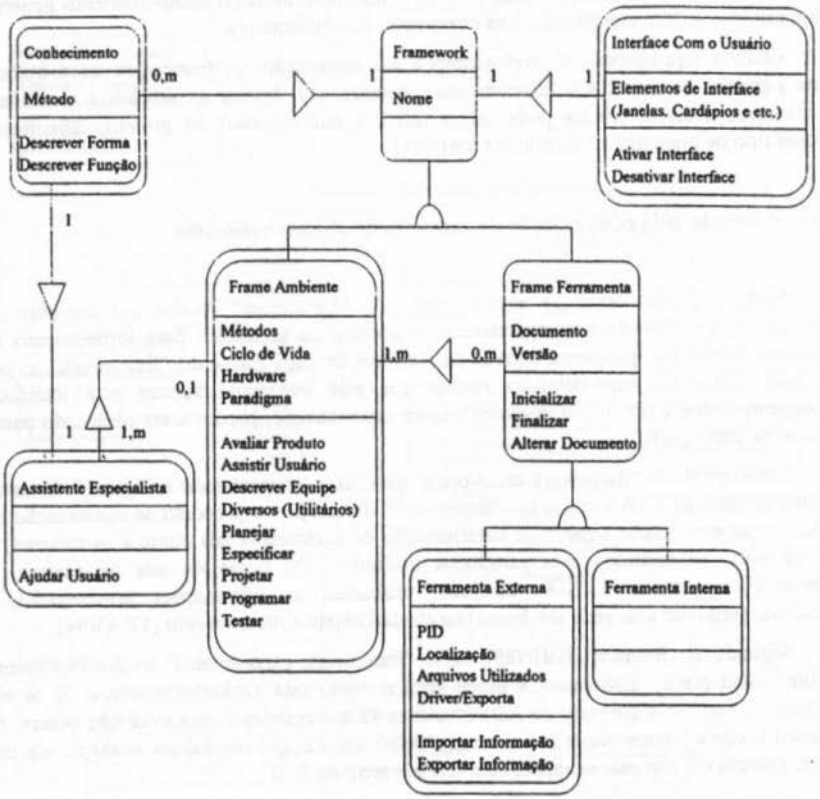

Figura 2 - "Framework caixa-preta", baseado no utilizado pelo meta-ambiente da estação TABA [TRAV94].

5. Conclusão

O projeto de um "framework" não é uma tarefa tão objetiva quanto o desenvolvimento de uma aplicação especifica. Algumas vezes, na busca da generalidade, são criados atributos, serviços e até mesmo classes que, em determinadas especializações ou instanciações do "framework", podem não ser necessários.

Contudo, as classes (e suas partes) que compõem um "framework" não devem ser avaliadas individualmente, mas como um todo que pode trazer benefícios ao projeto de uma aplicação específica que pertence ao domínio para o qual o "framework" foi projetado. O "framework" é, antes de tudo, um repositório de conhecimento sobre um domínio de aplicação, modelado segundo o paradigma da orientação a objeto, e tal conhecimento ao ser reutilizado traz em si:

- elementos já testados e, portanto, confiáveis.
- conhecimento adquirido em aplicações anteriores, que é incluído no "framework".

Assim, através dos "frameworks", o paradigma da orientação a objetos apoia a reutilização nas fases iniciais do desenvolvimento de um software, transformando o modelo orientado a objeto numa espécie de arquitetura de domínio [CIMA94]. A representação deste modelo é influenciada pela escolha de um método específico de orientação a objetos.

No projeto do "framework" apresentado neste artigo foi utilizado o método proposto por Coad e Yourdon [COAD91]. Este método tem como principal vantagem a sua simplicidade, pois permite uma visão geral do sistema projetado em um único diagrama. Porém, a representação de subsistemas e de comportamento genérico depende de adaptações convenientes.

Existem métodos de orientação a objetos que suportam naturalmente a representação de tais características, como por exemplo, as propostas de Booch [BOOC94] e de Shlaer e Mellor [SHLA92]. Estes dois métodos representam um projeto orientado a objetos sob diversas perspectivas, separarando a representação de herança, cooperação e comportamento em diagramas diferentes, acabando por dificultar uma visão única do sistema. Porém, estes métodos não só permitem, como enfatizam, o uso do conceito de subsistemas no desenvolvimento de software orientado a objetos, e fornecem notações para a representação de comportamento genérico. No caso do método de Shlaer e Mellor, a representação de comportamentos concretos e abstratos recebe ênfase especial através do uso de diagramas de estado como base na modelagem do ciclo de vida de objetos.

Bibliografia

[AGUI92] Aguiar, T.C. ; 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.

[BOOC94] Booch, G.; "Object-Oriented Analysis and Design with Applications"; Benjamin/Cummings, Redwood City, CA; 1994.

[CHIK88] Chikofsky, E.; Rubenstein, B.L.; "CASE: Reliability Engineering for Information Systems", Computer-Aided Software Engineering; (ed.) Chikofsky, E.; IEEE Computer Society Press, 1993.

[CIMA94] Cima, A.M ; Werner, C.M.L.; Castro, A.A.C.; "The Design of Object-oriented Software with Domain Architecture Reuse", Third International Conference on Software Reuse, Novembro 1994 (aceito para publicação).

[COAD91] Coad, P.; Yourdon, E.; "Object-Oriented Analysis"; 2nd. Edition; Prentice Hall, 1991.

[DEUT89] Deutsch, L.P.; "Design Reuse and Frameworks in the Smalltalk-80 System; Software Reusability"; Vol.II; (eds.) T.J.Biggerstaff and A.J.Perlis; ACM Press; 1989.

[DOLO78] Dolotta, T.A. , Haight, R.C. and Mashey, J.R. ; "UNIX Time-Sharing System: The Programmer's Workbench", Bell Systems Journal, vol. 57, no. 6, Jul-Ago 1978

[GOLD83] Goldberg, A.; "Smalltalk-80: The Interactive Programming Environment", Reading Mass.; Addison-Wesley, 1983

[ISE90] ISE - Interative Software Engineering Inc.; "Eiffel: The User Guide Version 2.3", Appendix A; An Introduction to Eiffel, Outubro de 1990.

[JOHN88] Johnson, R.E.; Foote, B.; "Designing Reusable Classes"; Journal of Object-Oriented Programming, 1 (2); Jun/Jul 1988.

[MOSL92] Mosley, V.; "How to Assess Tools Efficiently and Quantitatively", Computer-Aided Software Engineering, (ed.) Chikofsky, E.; IEEE Computer Society Press, 1993.

[PENE88] Penedo, M.H. and Riddle, W.E.; "Guest Editors' Introduction Software Engineering Environment Architectures", IEEE Transactions on Software Engineering , vol. 14, no 6, Jun 1988

[PRIE91] Prieto-Diaz, R.; Arango, G. "Domain Analysis Concepts and Research Directions"; Domain Analysis and Software Systems Modeling; (ed.) R. Prieto-Diaz and G. Arango, IEEE Computer Society Press Tutorial, 1991.

[ROCH87] Rocha, A.R.C.; Aguiar, T.C. e Blaschek, J.R.S.; Ambientes para Desenvolvimento de Software: Definição de Termos; Relatório Técnico do Programa de Engenharia de Sistemas e Computação - ES 137/87, COPPE/UFRJ, 1987

[ROCH90] Rocha, A.R.C.; Aguiar, T.C. ; Souza, J.M. ; "TABA: a heuristic workstation for software development"; COMPEURO'90; Tel Aviv, Israel, maio 1990

[ROTH75] Rothkind, M.; "The Source Code Control System"; IEEE Transactions on Software Engineering, vol. 1, no.14, Dez, 1975

[SAIT89] Saito, N.; "The Software Engineering Environment", Japanese Perspectives on Software Engineering, Matsumoto,Y.; Ohno,Y.(ed.); Addison-Wesley, 1989.

[SHLA92] Shlaer, S. e Mellor, S.J.; "Object Lifecycles - Modeling the World in States" ; Yourdon Press, 1992.

[SMIT90] Smith, D. ; Oman, P. ; "Software Tools in Context", IEEE Software, Maio 1990

[TRAC88] Tracz, W.; "Software Reuse: Motivators and Inhibitors"; Software Reuse Emerging Technology; (ed.) Will Tracz; 1988.

[TRAV94] Travassos, G.H.; O Modelo de Integração de Ferramentas da Estação TABA, Tese de Doutorado, COPPE/UFRJ, Programa de Engenharia de Sistemas de Computação, Março de 1994.

[WIRF90] Wirfs-Brock, R.J. e Johnson, R.E.; "Surveying Current Research in Object-Oriented Design", Communications of the ACM, 33 (9); Setembro - 1990.

[VESS92] Vessey, I. et alli; "Evaluation of Vendors Products: CASE Tools as Methodology Companions ", Communications of the ACM, vol. 35, no. 4, abril 1992.