

UNIVERSIDADE FEDERAL DE SANTA CATARINA
CENTRO TECNOLÓGICO
DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA

Reengenharia do framework OCEAN

Ademir Coelho

Florianópolis

2007

Ademir Coelho

Reengenharia do framework OCEAN

Trabalho de conclusão de curso apresentado como parte dos requisitos para obtenção do grau de Bacharel em Sistemas de Informação na Universidade Federal de Santa Catarina.

Orientador:

Ricardo Pereira e Silva, Dr.

UNIVERSIDADE FEDERAL DE SANTA CATARINA
CENTRO TECNOLÓGICO
DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA

Florianópolis

2007

Ademir Coelho

REENGENHARIA DO FRAMEWORK OCEAN

Trabalho de conclusão de curso apresentado como parte dos requisitos para obtenção do grau de Bacharel em Sistemas de Informação na Universidade Federal de Santa Catarina.

Florianópolis, 26 de fevereiro de 2007.

Ricardo Pereira e Silva, Dr.
Orientador
Universidade Federal de Santa Catarina

Vitório Bruno Mazzola, Dr.
Banca Examinadora
Universidade Federal de Santa Catarina

Patrícia Vilain, Dra.
Banca Examinadora
Universidade Federal de Santa Catarina

*Dedico este trabalho
à minha filha,
Alice.*

Agradecimentos

Agradeço a Deus por esse trabalho, que lembra de mim mesmo quando eu não lembro dele.

Ao professor Ricardo, pela paciência e empenho em me orientar. Pessoa esta que vou levar para o resto da vida como exemplo de orientador, professor e principalmente como exemplo de amigo.

A banca, professora Patrícia e professor Mazzola, por colaborarem com este trabalho.

À minha família pela compreensão e por acreditar em mim, mais do que eu mesmo acreditei.

Aos meus amigos pelo apoio, em especial ao Thiago, João e o Fernando.

E a todos que de alguma forma me ajudaram na conclusão deste trabalho.

*Não tenho um caminho novo.
O que eu tenho é um jeito novo de caminhar.*
Thiago de Melo

Resumo

Com a crescente produção de softwares sentiu-se a necessidade de se fazer softwares complexos em um curto espaço de tempo. Frameworks são alguns desses artefatos que possibilitam tais feitos, porém existem obstáculos relacionados a frameworks, que são a complexidade de desenvolvimento e uso.

O framework OCEAN foi concebido por Ricardo Pereira e Silva para obtenção de grau de doutor. Sua estrutura suporta a construção de ambientes de desenvolvimento que manuseiam especificações de projeto. Funções como a verificação de consistência e a geração de código se dão a partir de uma especificação de projeto do artefato utilizando UML.

Este trabalho visa uma reengenharia do framework OCEAN para uma melhor estruturação do mesmo, assim como, para gerar uma versão do framework em Java.

Abstract

With the software production increasing it was realized the necessity of make complex softwares in a short time. Frameworks are one of them artefacts that make it possible, but there are some obstacles related with frameworks, that are the complexity of development and use.

The OCEAN framework was developed by Ricardo Pereira e Silva to obtain the graduation of Doctor. Its structure supports the development environments building that handle design specifications. Functions as the consistence verification and code generation are obtained through a design specification of the artefact using UML.

This paper aims at the OCEAN framework core reengineer to improve their structure, as well as to produce a Java framework version.

Sumário

Lista de Figuras	p. 11
Lista de abreviaturas e siglas	p. 13
1 Introdução	p. 14
1.1 Motivação	p. 15
1.2 Tema	p. 15
2 Frameworks orientados a Objetos	p. 17
2.1 Conceitos básicos sobre frameworks	p. 18
2.2 Vantagens	p. 21
2.2.1 Generalidade	p. 21
2.2.2 Modularidade	p. 22
2.2.3 Reusabilidade	p. 22
2.2.4 Extensibilidade	p. 22
2.2.5 Robustez	p. 23
2.2.6 Protocolo	p. 23
2.3 Problemas	p. 23
2.3.1 Complexidade de desenvolvimento	p. 23
2.3.2 Complexidade de uso	p. 25
2.4 Classificação	p. 27
2.4.1 Quanto ao uso	p. 27
2.4.2 Quanto à finalidade	p. 28

2.5	Ciclo de vida	p. 29
3	Framework OCEAN	p. 30
3.1	Requisitos desejáveis a um ambiente de desenvolvimento de software	p. 31
3.2	Dependências	p. 32
3.3	Flexibilidade no framework OCEAN	p. 33
3.4	Extensibilidade no framework OCEAN	p. 34
3.4.1	Extensão em relação ao ambiente	p. 34
3.4.2	Extensão em relação à especificação	p. 36
3.4.3	Extensão em relação às ferramentas	p. 40
4	Ambiente SEA	p. 42
4.1	Estrutura do ambiente	p. 42
4.2	Requisitos desejáveis a um ambiente de desenvolvimento e uso de frameworks e componentes	p. 43
4.3	Tratamento de frameworks no ambiente SEA	p. 44
4.3.1	Auxílio à descrição de aplicações	p. 44
4.3.2	Auxílio à descrição de classes	p. 45
4.3.3	Auxílio à descrição de métodos	p. 45
4.4	Especificações no ambiente SEA	p. 46
4.4.1	Especificação OO	p. 46
4.4.2	<i>Cookbook</i> ativo - Suporte ao uso de frameworks	p. 52
4.4.3	Especificação de interface de componente	p. 53
5	Reengenharia do Framework OCEAN	p. 54
5.1	Engenharia reversa	p. 55
5.2	Reengenharia	p. 56
5.3	Engenharia reversa no framework OCEAN	p. 58

5.4	Reimplementação do framework OCEAN	p. 61
5.4.1	Ferramentas de conversão	p. 61
5.4.2	Conversão de SmallTalk para Java	p. 64
5.4.3	Validação do código gerado	p. 68
5.4.4	Experimentos	p. 68
5.5	Resultados	p. 69
5.5.1	Execução primitiva do ambiente	p. 69
5.5.2	Suporte à criação de diagrama de classes texto	p. 69
5.5.3	Protótipo do ambiente SEA	p. 70
5.6	Adaptações	p. 72
5.6.1	Uso de coleções no OCEAN	p. 72
5.6.2	Alteração da estrutura	p. 73
5.7	Melhorias	p. 76
5.7.1	Uso do padrão de projeto <i>Observer</i>	p. 77
5.7.2	Suporte a criação de elementos gráficos	p. 77
5.7.3	Suporte ao armazenamento de especificações	p. 78
6	Conclusão	p. 81
6.1	Realizações	p. 81
6.2	Avaliação	p. 82
6.3	Trabalhos futuros	p. 82
6.4	Considerações finais	p. 84
	Referências Bibliográficas	p. 85

Lista de Figuras

2.1	Exemplo hipotético de um framework de talheres	p. 18
2.2	Exemplo do princípio de Hollywood	p. 20
2.3	Aplicação desenvolvida totalmente	p. 20
2.4	Aplicação desenvolvida reutilizando classes de biblioteca	p. 20
2.5	Aplicação desenvolvida reutilizando um framework	p. 21
2.6	O ciclo de vida do desenvolvimento de frameworks	p. 29
3.1	Superclasses do framework OCEAN que definem a estrutura de uma especificação	p. 35
3.2	Tipos de ferramentas de ambiente sob o framework OCEAN	p. 40
4.1	Estrutura do ambiente SEA	p. 43
4.2	Diagrama de casos de uso do framework FraG	p. 48
4.3	Diagrama de atividades do framework FraG	p. 48
4.4	Parte de um diagrama de classes do framework FraG	p. 49
4.5	Diagrama de transição de estados da classe <i>GameUserInterface</i> do framework FraG	p. 49
4.6	Parte de um diagrama de seqüência do framework FraG que refina o caso de uso “inicialização”	p. 50
4.7	Diagrama de corpo de método do método <i>controlActivity</i> da classe <i>ClickController</i> do framework FraG	p. 50
5.1	Visualizações de software no ciclo de desenvolvimento	p. 55
5.2	Categorias da engenharia reversa e suas visões	p. 56
5.3	Relacionamentos no ciclo de desenvolvimento de software	p. 57
5.4	Matriz de decisão do que fazer com o sistema antigo	p. 58

5.5	Etapas para entendimento global do framework OCEAN	p. 60
5.6	Diagrama de fluxo de dados criado estendendo o ambiente SEA em SmallTalk	p. 60
5.7	Método em SmallTalk	p. 62
5.8	Método convertido com o Bistrô	p. 63
5.9	Método convertido com o St2J	p. 63
5.10	Método desejável em Java	p. 64
5.11	Processo para evitar que os erros de uma classe interfiram em outra	p. 65
5.12	Ambiente DCSEA para validação de diagrama de classe	p. 70
5.13	Diagrama de classe, de estado e de método no ambiente SEA em Java	p. 71
5.14	Diagrama de caso de uso, de atividade e de seqüência no ambiente SEA em Java	p. 72
5.15	Estrutura de classe dos conceitos de método, atributo, parâmetro de método e variável de método em SmallTalk	p. 75
5.16	Estrutura de classe dos conceitos de método, atributo, parâmetro de método e variável de método em Java	p. 75
5.17	Estrutura de classes do metamodelo do framework OCEAN em SmallTalk	p. 76
5.18	Estrutura de classes do metamodelo do framework OCEAN em Java	p. 76
5.19	Estrutura de classes que definem o novo mecanismo de armazenamento	p. 79

Lista de abreviaturas e siglas

API	Application Programming Interface (Interface para Programação de Aplicativos)
JVM	Java Virtual Machine (Máquina Virtual Java)
MVC	Model-View-Controller (paradigma para o desenvolvimento de interfaces gráficas)
OO	Object-Oriented (Orientado a Objetos)
OOAD	Object-Oriented Analysis and Design (Análise e Projeto Orientados a Objetos)
UML	Unified Modelling Language (Linguagem de Modelagem Unificada)
XMI	XML Metadata Interchange (XML para Intercâmbio de Metadados)
XML	eXtensible Markup Language (Linguagem eXtensível de Marcação)

1 *Introdução*

O ramo da engenharia civil se atenta em técnicas que produzam construções confiáveis, de qualidade e em um curto espaço de tempo. Da mesma forma a engenharia do software tem as mesmas preocupações, produzir softwares confiáveis, de qualidade e em um curto espaço tempo (FAIRLEY, 1986 apud SILVA, 2000). Para produzir artefatos com as características citadas é necessário, assim que possível, reusar o que já foi feito (JOHNSON, 1997). Considerando o exemplo de um conjunto habitacional, é bem mais vantajoso reutilizar o projeto de outra habitação do que criar um projeto para cada habitação.

A reusabilidade vem sendo uma das principais metas dos engenheiros de software (KNAB-BEN; ROBERT, 2002). No entanto, não é fácil reusar software, e em muitos casos, os esforços de reuso resultam em artefatos de uso específico, com replicação de algumas partes do código. Ou seja, normalmente os esforços se concentram na forma mais primitiva de reuso, a reusabilidade de código (SILVA, 2000).

As abordagens de frameworks orientado a objetos e desenvolvimento baseado em componentes promovem o reuso em um nível de granularidade elevado, não somente de código, mas também de projeto (JOHNSON, 1997). Porém a utilização dessas abordagens é desfavorecida pela complexidade de desenvolvimento e uso (JOHNSON; FOOTE, 1988). Nesse contexto de reuso, principalmente de projeto, é que se apresenta o framework OCEAN e o ambiente de desenvolvimento SEA, produzidos em 2000 por Ricardo Pereira e Silva durante o seu doutorado¹.

O framework OCEAN dá suporte à construção de ambientes de desenvolvimento de software que sejam manuseados por especificações de projeto. O framework possui as funcionalidades genéricas de construção e edição de especificações. Além disso, dá suporte ao desenvolvimento de artefatos para a manipulação de especificações a serem vinculados a um ambiente.

O ambiente SEA foi desenvolvido estendendo a estrutura do framework OCEAN e dá su-

¹Neste trabalho, quando se fala sobre o framework OCEAN e o ambiente SEA usou-se a tese de doutorado (SILVA, 2000) como subsídio, por ser a principal e mais completa fonte de referência sobre estes artefatos. Esta nota se justifica para que o texto não possua excessivas citações à tese de doutorado gerada por Ricardo Pereira e Silva.

porte ao desenvolvimento e uso de frameworks e componentes. Para utilizar este ambiente no desenvolvimento de um framework, componente ou uma aplicação, é necessário que se construa uma especificação de projeto do artefato utilizando UML e, de forma automatizada, verificar a consistência e convertê-la em código.

1.1 Motivação

O framework OCEAN e o ambiente de desenvolvimento SEA foram ambos desenvolvidos em SmallTalk através do ambiente VisualWorks versão 2. Além de ser uma versão antiga, lançada em 1994, é uma linguagem interpretada e pouco difundida hoje em dia. Com isso, a difusão, extensão e eficiência do framework OCEAN e do ambiente SEA são dificultadas, no comparativo com outras linguagens mais atuais, como Java por exemplo.

O desenvolvimento baseado em frameworks oferece diversas vantagens em relação aos métodos de desenvolvimento tradicionais, entretanto existem alguns empecilhos que tornam a sua utilização dificultosa. Os principais problemas são a complexidade de desenvolvimento e a complexidade de uso. Estes problemas não são diferentes no âmbito do framework OCEAN, porém há fatores agravantes destes problemas, dificultando assim a sua extensibilidade e o seu uso.

O Framework OCEAN foi implementado utilizando o framework HotDraw para suporte ao desenvolvimento de editores gráficos, que também foi implementado em SmallTalk. Atualmente o framework OCEAN apresenta estas características relacionadas à linguagem utilizada: linguagem pouco difundida, execução dependente do ambiente de desenvolvimento, e o framework HotDraw não possui compatibilidade com versões atuais da linguagem.

1.2 Tema

Este trabalho visa a reengenharia do framework OCEAN, tendo como resultado esperado uma nova versão em Java do framework. O projeto do framework será melhorado, trazendo novas abstrações e modificando as já existentes, e na medida de possível, utilizando padrões de projeto para esta tarefa.

Tem-se como objetivo a conversão e a validação de todas as partes inerentes ao framework, essas partes incluem as definições genéricas de: gerente de ambiente, gerente de armazenamento, gerente de ferramentas, gerente de navegação, gerente de referência, repositório de elementos de especificação e a estrutura de documentos. A validação da reengenharia de um

framework pressupõe que o mesmo seja totalmente testado. O framework OCEAN não compreende um artefato executável, por isso, para a validação da reengenharia será também necessária a conversão de partes externas ao framework.

Para a validação do presente trabalho será convertido e validado o ambiente SEA, assim como a estrutura e todos os elementos de uma especificação orientada a objetos. Será usado o ambiente SEA, por ser a aplicação que usa a totalidade dos recursos do framework OCEAN, e a especificação OO, por ser o tipo de especificação mais usual e a que possui o maior número de modelos e conceitos das especificações previstas no ambiente SEA. Desta forma, a reengenharia do framework OCEAN será validada através do uso de uma especificação OO no ambiente SEA em Java, e para tanto, far-se-á uso da interface gráfica projetada para o ambiente SEA, que é tratada em outro trabalho².

²A interface gráfica do ambiente SEA é tratada no trabalho de conclusão de curso de Thiago Machado (MACHADO, 2007).

2 *Frameworks orientados a Objetos*

Apesar dos avanços computacionais, o projeto e a implementação de aplicações complexas são custosos e suscetíveis a erros. Uma grande parcela do custo e esforço de desenvolvimento dessas aplicações derivam do constante redescobrimto e reinvenção de conceitos e artefatos de software¹ (FAYAD; SCHMIDT, 1997). Uma alternativa a esta situação é o reuso de aplicações existentes.

Softwares podem abranger várias maneiras de reutilização. Podendo estas atingir níveis mais baixos de granularidade², como a reutilização de código, ou níveis mais altos de granularidade, como a de análise e projeto. A reutilização de código é a utilização de rotinas de código já desenvolvidas, enquanto a reutilização de projeto refere-se ao uso de uma estrutura arquitetônica de outra aplicação previamente construída, sem necessariamente ter a mesma implementação (SILVA, 1996).

A forma mais elementar de reuso de software é o reuso de código. Para materializar este tipo de reuso são criadas bibliotecas que implementam a rotina a ser reusada, como as bibliotecas em Java para criação de arquivo.

O reuso de projeto é uma das mais importantes formas de reuso. Segundo Silva (SILVA, 2000), verificam-se dois objetivos distintos na reutilização no nível das especificações de análise e projeto:

- Utilizando um projeto de uma outra aplicação para desenvolver a aplicação em questão. Utilizado no projeto de reengenharia.
- Utilizando uma especificação de projeto de um conjunto de aplicações para desenvolver uma aplicação de mesmo domínio. Utilizado para a abordagem de frameworks orientados a objetos.

¹Artefatos de software podem ser considerados como qualquer resultante de um processo de construção de software, tais como aplicações, frameworks ou componentes.

²Nível de detalhe empregado.

2.1 Conceitos básicos sobre frameworks

Frameworks resultam da necessidade de reuso do maior número possível de trechos³ de aplicações já existentes. São estruturas, propositalmente inacabadas, para construção de artefatos de software, que podem conter diferentes módulos de funcionalidades específicas, aumentando a possibilidade desses artefatos serem produzidos mais rapidamente (SILVA, 2000), graças ao reuso empregado. A figura 2.1 ilustra um exemplo hipotético de um framework para talheres e as aplicações suportadas por ele.

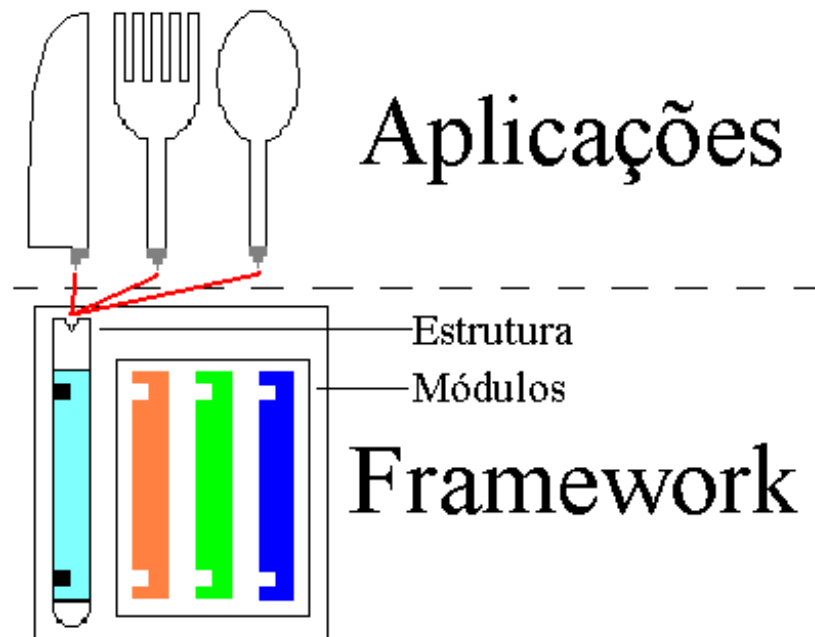


Figura 2.1: Exemplo hipotético de um framework de talheres.

Um framework generaliza um domínio, uma vez que este deve suportar um conjunto específico de aplicações. Para atender ao domínio tratado, frameworks fornecem o reuso de projeto e código. Possibilitando mais que um mero reuso de rotina ou de arquitetura, mas sim de soluções. Por esta razão frameworks são artefatos com alto nível de granularidade (JOHNSON; FOOTE, 1988).

No caso ideal, um framework deve conter todas as abstrações do domínio tratado, e uma aplicação gerada sob o framework não deveria conter abstrações que não estão presentes neste domínio. Bastando ao usuário a tarefa de completar as informações específicas de sua aplicação. Geralmente é inviável o framework prover todas as abstrações, e por essa razão o framework tende a absorver abstrações durante o seu ciclo de vida, caracterizando assim uma evolução

³Entende-se trecho como sendo parte do produto de qualquer fase de desenvolvimento de software (análise, projeto ou código).

iterativa.

Frameworks orientados a objetos⁴ compartilham um conjunto de características com técnicas de reuso em geral, e mais especificadamente, com técnicas de reuso orientado a objetos. Utilizam o paradigma de orientação a objetos para desenvolver a descrição do domínio proposto, consistindo em uma classe abstrata para cada componente principal (JOHNSON, 1997).

Johnson e Foote (JOHNSON; FOOTE, 1988) propõem que: “um framework é um conjunto de classes que incorporam um projeto abstrato para a solução de uma família de problemas correlacionados”. Wirfs-Brock et al. (WIRFS-BROCK et al., 1990 apud SILVA, 1996) define como: “um esqueleto de implementação de uma aplicação ou de um subsistema de aplicação, em um domínio particular. É composto de classes abstratas e concretas e provê um modelo de interação ou colaboração entre as instâncias definidas pelo framework”. E em outra publicação Johnson (JOHNSON, 1997) diz que: “um framework é projeto reusável de tudo ou de parte do sistema que é representado por uma coleção de classes abstratas e a maneira com que as instâncias interagem”, seguindo por outra definição: “um framework é um esqueleto de uma aplicação que pode ser adaptado pelo desenvolvedor”, e diz que a primeira definição descreve a estrutura do framework enquanto a segunda descreve o seu propósito.

Além de um framework possuir o modelo de relacionamento entre as instâncias das classes definidas, uma de suas características é a inversão do fluxo de controle, conhecido com princípio de Hollywood⁵. A responsabilidade de saber quais instâncias serão chamadas é do framework, e não da aplicação. Desta forma, as classes da aplicação esperam ser chamadas pelo framework durante o tempo de execução. A figura 2.2 mostra essa inversão do fluxo de controle.

Diferente de uma biblioteca de classes, frameworks não provêm somente classes com rotinas específicas, mas também o esqueleto básico da aplicação do domínio tratado. Um esqueleto de um framework não é passivo, como uma biblioteca de classe, mas tem seu próprio caminho de execução de código, chamando o código definido pelo usuário. Por serem artefatos isolados, o uso de bibliotecas promove o reuso de rotinas mas deixa a cargo do usuário a tarefa de estabelecer as interligações da aplicação (SILVA, 2000). Por outro lado, o uso de frameworks promove o reuso de arquitetura e rotinas, bastando ao usuário a tarefa de completar a especificidade desejada à aplicação. As figuras 2.3, 2.4 e 2.5 ilustram esta diferença (a parte sombreada representa classes e associações que são reutilizadas).

O aprendizado sobre frameworks é mais dispendioso do que o de bibliotecas, porque não se pode ver as classes como elementos isolados ao tentar aprender sobre elas (JOHNSON, 1997).

⁴A partir deste ponto a expressão *framework* refere-se a *framework orientado a objeto*.

⁵“Não ligue para nós, nós ligaremos para você”.

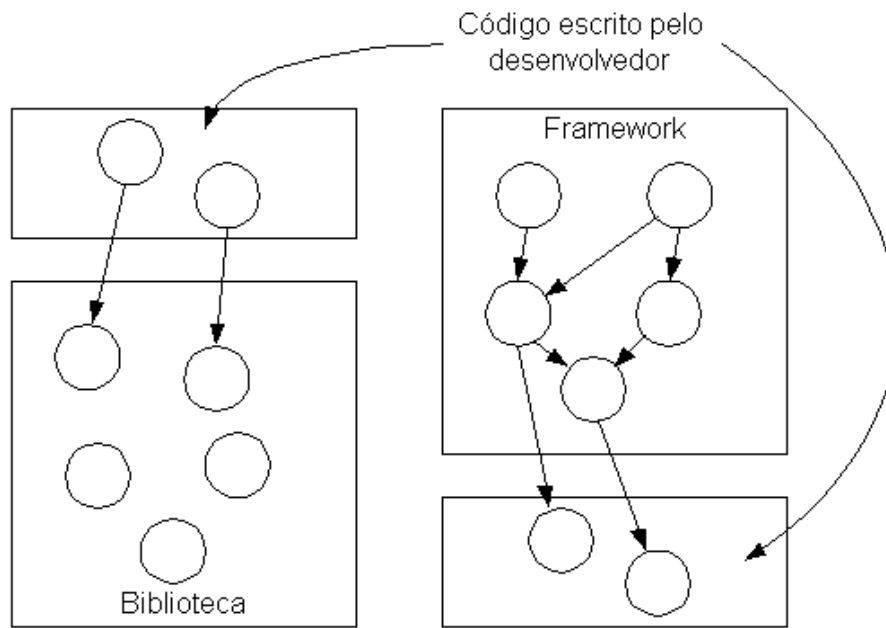


Figura 2.2: Exemplo do princípio de Hollywood (SAUVÉ, 2005).

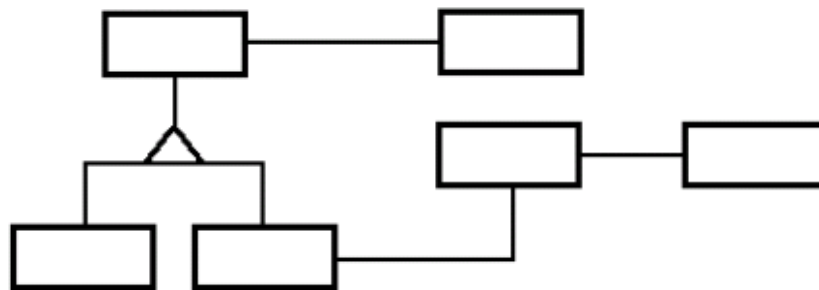


Figura 2.3: Aplicação desenvolvida totalmente (SILVA, 2000).

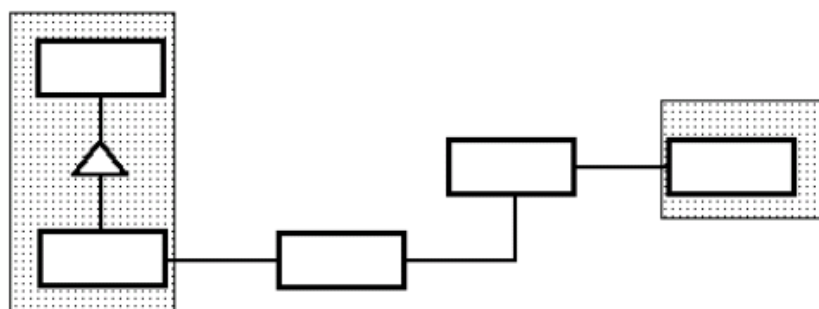


Figura 2.4: Aplicação desenvolvida reutilizando classes de biblioteca (SILVA, 2000).

Devido ao alto nível de granularidade do seu reuso, frameworks são mais difíceis de projetar que meras classes abstratas ou bibliotecas de classe.

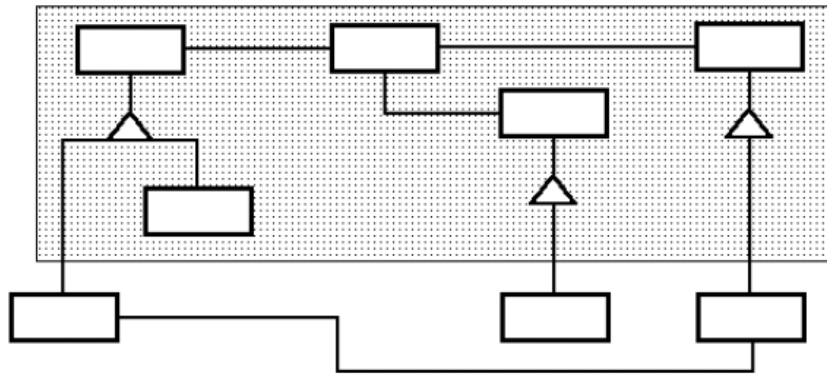


Figura 2.5: Aplicação desenvolvida reutilizando um framework (SILVA, 2000).

2.2 Vantagens

Dependendo do caso, o desenvolvimento ou uso de frameworks não é proveitoso, para tanto é necessário ter motivo(s) satisfatório(s). No caso de desenvolvimento quando o número de aplicações que serão desenvolvidas é reduzido e a complexidade envolvida nas aplicações é baixa, é questionável a real necessidade de abstrair um domínio. Semelhante situação ocorre quanto ao uso de um framework, quando o número de aplicações a serem desenvolvidas é reduzido, e o tempo para se aprender a usar o framework é maior que o tempo para se construir a uma aplicação independente.

Entretanto, quando a avaliação favorece o desenvolvimento ou uso de framework, este promove diversas vantagens:

2.2.1 Generalidade

A generalidade é a principal característica que se busca quando se deseja construir um framework. Uma vez desprovido de generalidade, este não pode ser caracterizado como um framework, uma vez que servirá somente para a construção de uma única aplicação, fugindo da definição que prevê que um framework é utilizado para um domínio de aplicações.

Buscar generalidade consiste em identificar estruturas em comum nas aplicações analisadas, e na elucidação de estruturas semelhantes, de forma a obter uma estrutura de classes que cubra o domínio tratado (SILVA; PRICE, 1998).

A generalidade de um domínio em um framework ajuda o usuário no esclarecimento de abstrações úteis à sua aplicação, mas que não foram identificados no esboço de sua aplicação.

2.2.2 Modularidade

Frameworks possuem modularidade incluindo detalhes de implementação em interfaces estáveis (FAYAD; SCHMIDT, 1997), de forma que as mudanças do framework não afetarão a aplicação⁶. Por conseqüência, a flexibilidade⁷ de um framework está amplamente relacionada a sua modularidade, pois quanto mais modular for o framework, mais fácil será para alterar as suas funcionalidades.

A modularidade de um framework é possível graças às características suportadas pelo paradigma de orientação a objetos. Definições de classes provêm a descentralização da informação. Polimorfismo reduz o número de procedimentos, e conseqüentemente, o tamanho do programa que tem que ser entendido pelo mantenedor. Herança permite a construção de uma nova versão do programa sem que esta afete a antiga (JOHNSON; FOOTE, 1988).

A modularidade facilita a manutenção e o entendimento do framework e das aplicações existentes. Além de ajudar a melhorar a qualidade de ambos, diminuindo o impacto de mudanças no projeto ou na implementação.

2.2.3 Reusabilidade

Uma aplicação desenvolvida sobre o framework reusa não somente código, mas também o projeto. Este reuso em alta granularidade, assim como no caso da modularidade, é derivado das características do paradigma de orientação a objetos. Processos que usam polimorfismo são mais fáceis de reusar do que processos que não o usam, pois o uso de polimorfismo faz com que a faixa de argumentos tratada seja mais ampla. O uso de herança permite a construção de subclasses que reusem todas as definições⁸ acessíveis das classes da hierarquia.

O reuso provido pelo framework gera melhoras substanciais na produtividade, assim como na qualidade, performance, robustez e na interoperabilidade das aplicações geradas (FAYAD; SCHMIDT, 1997).

2.2.4 Extensibilidade

Extensibilidade é a capacidade de um framework aumentar as suas funcionalidades, ou seja, refere-se à capacidade do framework se adaptar a manutenção. A construção de frameworks

⁶Desde que respeitando a estrutura, interface e o protocolo do framework.

⁷Capacidade do framework alterar as suas funcionalidades (SILVA; PRICE, 1998).

⁸Tais como atributos e métodos.

provê a preocupação com a extensibilidade, uma vez que estes possuem ciclo de evolução iterativo e tendem a agregar ou alterar conceitos devido à construção de novas aplicações.

A extensibilidade de um framework permite que o seu domínio seja aumentado ou refinado mais facilmente através da adição ou edição de métodos *hook*⁹ e *hot spots*¹⁰. Entretanto, para isto é necessário que o framework use amplamente os princípios de orientação a objetos (SILVA; PRICE, 1998), tais como herança, composição e polimorfismo.

2.2.5 Robustez

É a habilidade de um software manter as funcionalidades mesmo com mudanças na estrutura interna ou externa. As aplicações geradas sobre o framework são mais confiáveis e robustas, uma vez que usam código e estrutura do framework, cuja estrutura e código já foram testados e depurados. E quanto mais aplicações o framework suporta, mais robusto tende a ficar, e conseqüentemente também tornando mais robustas as aplicações suportadas.

2.2.6 Protocolo

Frameworks “chamam” e não são “chamados”, ou seja, contém um protocolo próprio (princípio de Hollywood), deixando o desenvolvedor livre da tarefa de esquematizar quais e quando as instâncias devem ser chamadas, e permitido que este se preocupe somente com as especificidades da aplicação.

2.3 Problemas

Apesar de apresentar diversas vantagens, frameworks têm como principais problemas a complexidade de desenvolvimento e uso.

2.3.1 Complexidade de desenvolvimento

Complexidade de desenvolvimento está relacionado ao esforço necessário para produzir uma abstração que generalize um domínio para diferentes aplicações.

⁹“Métodos *hook* formam, em conjunto com métodos *template*, um ponto de flexibilidade. Métodos *hook* podem ser redefinidos pela aplicação e são utilizados pelos métodos *templates*, que implementam de forma genérica um algoritmo” (KNABBEN; ROBERT, 2002).

¹⁰“*Hot spots* são as partes da estrutura de classes de um framework que devem ser mantidas flexíveis, para possibilitar sua adaptação a diferentes aplicações do domínio” (SILVA; PRICE, 1998).

Metodologias de desenvolvimento

Uma metodologia de desenvolvimento de frameworks é composta de procedimentos para captar as características de um domínio, e para desenvolver uma abstração que generalize este domínio. As principais metodologias para o desenvolvimento de frameworks são: *projeto dirigido por exemplo*, *projeto dirigido por hot spot* e a metodologia *Taligent*.

A metodologia de *projeto dirigido por exemplo* se caracteriza pela produção de uma generalização através do aprendizado de aplicações semelhantes de forma *bottom-up* (JOHNSON, 1993 apud SILVA, 1996).

A metodologia *projeto dirigido por hot spot* se caracteriza pelo processo cíclico na identificação de partes flexíveis na estrutura de classes do framework (PREE, 1995 apud SILVA, 1996).

E a metodologia *Taligent* se caracteriza pela produção de diversos frameworks menores direcionados aos aspectos específicos do problema, que em conjunto darão origem às aplicações (TALIGENT, 1995 apud SILVA, 1996).

Nenhuma das três metodologias de desenvolvimento adota técnicas de modelagem para a descrição do framework, carecendo da descrição estática e dinâmica, tanto ao nível de classes isoladas, como ao nível de framework (SILVA, 1996). Segundo Silva (SILVA, 2000), as metodologias existentes, em sua maioria, não tratam modelagem de alto nível, voltando-se basicamente a geração de código. Considera-se que as metodologias para frameworks são incompletas, e portanto o desenvolvimento das descrições de frameworks é sustentado na sua maior parte pela experiência do desenvolvedor. O ideal seria que essas técnicas pudessem proporcionar no mínimo as seguintes descrições úteis no desenvolvimento do framework:

- diagrama de classes mostrando as classes e os métodos que devem e que podem ser estendidos pelo usuário no desenvolvimento da aplicação;
- diagrama de seqüência descrevendo a interação entre as instâncias, destacando o papel das classes na construção das aplicações.

Além da ineficiência em descrever o framework, outro problema é que as metodologias para desenvolvimento não provêm a existência de mecanismo de descrições de decisões. Um exemplo de descrição de decisão é a documentação da razão pela qual um determinado conceito foi agregado à abstração do framework. Isso faz com que o desenvolvimento, e principalmente a manutenção, sejam prejudicados pela falta dessas informações. Quando esse mecanismo existe, a maioria das vezes é controle de versão ou comentários de código.

Validação

A validação de um framework é diferente da validação de uma aplicação. Em uma aplicação, é possível executá-la e verificar a coerência das funcionalidades com os requisitos. Em frameworks isto não é possível, uma vez que um framework é uma “aplicação inacabada”, necessitando construir aplicações que o usem para fazer essa verificação de coerência entre as funcionalidades e os requisitos.

A validação de um framework é custosa, pois depende de elementos externos à ele, que também podem conter erros. Não se pode considerar que o framework foi validado se este não foi executado usando aplicações exemplo.

Os exemplos são concretos e facilitam o entendimento do framework como um todo. Eles resolvem um problema particular, e a execução deles possibilita aprender o fluxo de controle inserido no framework. Eles demonstram como é o comportamento do framework e como os programadores o usam. Idealmente, um framework deve ter um conjunto de exemplos que vão do trivial ao avançado, cobrindo toda a amplitude das características do domínio e todas as funcionalidades do framework (JOHNSON, 1997).

Podemos exemplificar isso citando o FraG¹¹. Caso seja testado usando apenas jogos de dois jogadores e que não usem elementos diferenciados¹² como o xadrez, o jogo da velha ou o jogo de damas, o framework certamente não será bem validado e não se adaptará a jogos que fujam deste âmbito, como por exemplo, jogo de ludo ou banco imobiliário.

2.3.2 Complexidade de uso

Complexidade de uso diz respeito ao esforço para aprender a desenvolver aplicações a partir do framework. O esforço necessário para aprender a usar um framework é considerável. Entretanto, depois da fase de aprendizado é possível começar o desenvolvimento de aplicações imediatamente (KNABBEN; ROBERT, 2002).

Documentação

Johnson (JOHNSON, 1997) afirma que a documentação do framework deve explicar:

- qual o propósito do framework;

¹¹Framework para jogos de tabuleiro (SILVA; PRICE, 1997).

¹²Ex: dado, cartas e conta de jogadores.

- como o framework funciona;
- como usar o framework.

Geralmente a complexidade de uso está relacionada à falta de documentação do domínio, do framework ou de uso do framework. Documentação do domínio se refere à explanação do domínio tratado pelo framework. Documentação do framework se refere à descrição de detalhes de implementação do framework. E a documentação de uso faz referência a um manual de como produzir aplicações a partir do framework.

Após a etapa de análise do domínio, é possível como resultado um documento que descreva detalhadamente o domínio tratado sem referências ao framework que será produzido. Este documento seria de grande valia a desenvolvedores que queiram criar uma aplicação usando o framework, mas conhecem pouco sobre o domínio tratado, e facilitaria para o desenvolvedor extrair tudo que o framework pode oferecer. No entanto, raramente este documento é anexado aos frameworks.

Normalmente a documentação de implementação do framework é somente código¹³. Na maioria das vezes, quando existe, se restringe ao diagrama de classes do framework e a uma descrição rápida das classes e métodos, não abrangendo o entendimento do framework como um todo e nem detalhes de implementação. A escassez desse tipo de documentação pode dificultar o desenvolvimento de aplicações mais complexas.

Uma das maneiras mais intuitivas de se aprender a usar um framework é a utilização de um *cookbook*¹⁴. *Cookbooks* não ajudam usuários avançados, mas eles são de grande importância para iniciantes (JOHNSON, 1997).

Exemplos

Johnson (JOHNSON, 1997) afirma que “a melhor maneira de aprender sobre o alcance da aplicabilidade dos frameworks é analisando os seus exemplos”. Normalmente os exemplos de um framework se limitam a geração de uma aplicação mínima, não explorando a totalidade de recursos disponíveis pelo mesmo. O esforço do aprendizado sobre as particularidades fica a cargo do desenvolvedor.

¹³No caso de um framework de código aberto, onde o usuário tem acesso ao código.

¹⁴Livro de receitas, contém passos a serem seguidos para produzir uma aplicação ou parte dela (SILVA, 2000).

2.4 Classificação

Frameworks podem ser classificados em 7 categorias: quanto à extensibilidade (*extensibility*), quanto ao escopo (*scope*), quanto ao acesso (*approach*), quanto à padronização (*standardization*), quanto à granularidade (*granularity*), quanto à licença (*license*) e quanto ao formato (*format*) (KRAJNC; HERIEKO, 2003). Neste trabalho, a atenção é voltada à classificação quanto ao uso (extensibilidade) e quanto à finalidade (escopo).

2.4.1 Quanto ao uso

A classificação quanto ao uso se refere à maneira com que o usuário terá que desenvolver a aplicação a partir do framework. Esta classificação evidencia um espectro de níveis de uso do framework, e não uma divisão propriamente dita (JOHNSON, 1997).

Caixa branca

Os frameworks do tipo caixa-branca são dirigidos à arquitetura, e o reuso é proporcionado pelo uso de herança. O comportamento específico é definido adicionando métodos nas subclasses de uma ou várias classes. Cada método adicionado à subclasse deve respeitar as convenções internas das superclasses.

Estes tipos de framework são mais poderosos quando utilizados por usuários avançados, entretanto para a produção de uma aplicação é necessário entender sobre a implementação do framework (JOHNSON, 1997). Um exemplo de framework caixa branca é o MVC¹⁵ do Small-Talk.

Caixa preta

Os frameworks do tipo caixa-preta são dirigidos a dados, e o reuso é proporcionado pelo uso de composição. As especificidades são definidas conectando componentes, sem a necessidade de se entender a implementação do framework.

Frameworks caixa preta são mais fáceis para aprender a usar, entretanto são menos flexíveis (JOHNSON; FOOTE, 1988). Muitos frameworks de interface gráfica possuem essa forma de uso, onde cabe ao usuário a tarefa de escolher quais componentes irão compor a aplicação de interface.

¹⁵Model/View/Controller.

Caixa cinza

Os frameworks do tipo caixa-cinza são uma combinação das possibilidades anteriores. Possuem partes prontas e permitem que sejam adicionadas novas funcionalidades a partir da criação de classes. Não somente fornecem subclasses concretas, mas permitem a criação de novas subclasses.

As vantagens e as desvantagens desse tipo de framework estão relacionadas com a distância do framework no espectro, ou seja, se são mais parecidos com frameworks caixa branca ou caixa preta.

2.4.2 Quanto à finalidade

A classificação quanto à finalidade refere-se ao propósito do uso do framework, podendo ser classificada em: framework de infra-estrutura de sistema, framework de integração *middleware* ou framework de aplicação empresarial (FAYAD; SCHMIDT, 1997).

Frameworks de infra-estrutura de sistema

São usados para suportar vários tipos de aplicações, independentes de seus domínios. Facilitam o desenvolvimento da infra-estrutura de sistemas, como por exemplo, frameworks para acessar recursos dos sistemas operacionais.

Frameworks de integração *middleware*

São usados para integrar aplicações e componentes distribuídos. Eles ajudam os desenvolvedores na tarefa de modularizar, reutilizar e estender a infra-estrutura de softwares em um ambiente distribuído.

Frameworks de aplicação empresarial

Estão voltados a domínios de aplicação, sendo os mais comumente encontrados, e são fundamentais para atividades de negócios das empresas.

2.5 Ciclo de vida

O ciclo de vida do desenvolvimento dos frameworks é diferente dos ciclos de vida tradicionais para o desenvolvimento de aplicações comuns, isto porque um framework nunca é um artefato isolado, mas sua existência está sempre relacionada à existência de outros artefatos (SILVA, 2000).

Frameworks podem conter conceitos que não são usados em todas as aplicações do domínio tratado. Porém, podem agregar novos conceitos previstos em uma nova aplicação, cabendo ao desenvolvedor refinar o framework para que este possa ser o mais abrangente possível.

A figura 2.6 mostra as etapas de desenvolvimento de um framework: análise do domínio, modelagem do framework, implementação do framework e aplicações sob o framework. De forma que o desenvolvimento de aplicações resulta no refinamento do framework, possibilitando o desenvolvimento de novas aplicações.

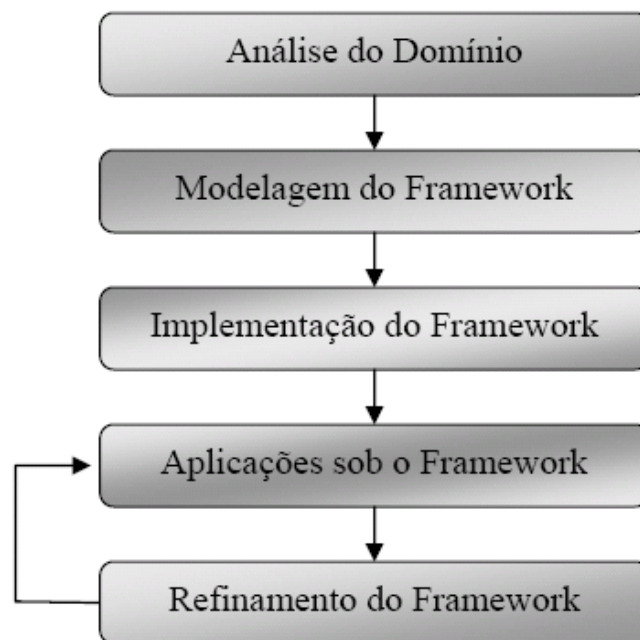


Figura 2.6: O ciclo de vida do desenvolvimento de frameworks. (KNABBEN; ROBERT, 2002).

3 *Framework OCEAN*

O OCEAN é um framework cujo seu domínio atende a produção de diferentes ambientes de desenvolvimento de software baseado em notação de alto nível. Ele possibilita que tais ambientes manipulem diferentes estruturas de especificação de projeto e apresentem diferentes funcionalidades. Cada tipo de especificação no framework OCEAN define quais os tipos de modelos¹ válidos para aquela especificação, e cada modelo define quais os tipos de conceitos² tratados por cada modelo.

Tanto o framework OCEAN quanto o ambiente SEA foram produzidos por Ricardo Pereira e Silva (SILVA, 2000). O framework OCEAN foi usado para a construção do ambiente SEA e teve como motivação a necessidade de construir um ambiente de software que suportasse o desenvolvimento e o uso de frameworks e componentes, e que fosse dotado de flexibilidade. Portanto o desenvolvimento do framework foi norteado pela busca da flexibilidade, que foi caracterizada como requisito funcional, para permitir que o ambiente se molde às necessidades encontradas durante o seu ciclo de vida.

A análise dos ambientes existentes quando comparada com os requisitos desejáveis ao framework possibilitou generalizar o domínio destes ambientes, identificando requisitos de flexibilidade ao framework OCEAN. Desta forma o OCEAN consegue dar suporte tanto ao desenvolvimento de ambientes relacionados a frameworks e componentes, como o desenvolvimento de ambientes em geral.

O OCEAN é um framework do tipo caixa cinza, pois fornece funcionalidades através de subclasses concretas e permite que novas funcionalidades sejam adicionadas a partir da criação de novas subclasses. Um ambiente mínimo é produzido através da criação de uma subclasse de *EnvironmentManager*³ e do reuso de classes previstas no framework para a criação do ambiente

¹Um modelo de uma especificação, como um diagrama de classes por exemplo, possui uma estrutura de informações (as classes, atributos, métodos, etc) e uma ou várias representações visuais desta estrutura (representação gráfica, textual, etc). Modelos podem ser vistos como um invólucro de um conjunto específico de informações.

²Um conceito representa uma unidade de informação do domínio tratado. Por exemplo, no caso da modelagem baseada no paradigma de orientação a objetos são tipos de conceitos: classe, atributo, herança, caso de uso, etc.

³Classe que define as características específicas do ambiente.

SEA. No entanto, pode-se criar classes para a extensão⁴ de abstrações⁵ genéricas definidas no framework. Em uma subclasse de EnvironmentManager são definidas as seguintes especificidades de um ambiente:

- quais os tipos de especificações tratadas;
- qual o mecanismo de visualização e edição para cada modelo e conceito;
- qual o mecanismo de armazenamento de especificações;
- quais as ferramentas utilizáveis para manipular especificações.

3.1 Requisitos desejáveis a um ambiente de desenvolvimento de software

Espera-se que ambientes de desenvolvimento de software disponham de processos automatizados para as várias etapas do ciclo de vida de um software. Silva (OLIVEIRA; MARTIN; ODELL, 1996, 1995 apud SILVA, 2000) estabelece que o desejável é que estes ambientes possuam as seguintes características:

Interface gráfica dos modelos: A disponibilidade de interfaces gráficas para edição e visualização de modelos inseridos em uma especificação pode facilitar a tarefa do desenvolvedor além de estimulá-lo, uma vez que o resultado do seu trabalho é visível através de elementos gráficos.

Repositório centralizado para os dados: Consiste em mecanismo que concentre todas as informações de uma especificação de forma não redundante, viabilizando o compartilhamento com diferentes mecanismos, além de garantir a coerência e a integridade. Além de compartilhar espera-se que este repositório dê suporte às funcionalidades de verificação de consistência e geração de código.

Verificação de consistência: Quando existe uma grande quantidade de informação, a verificação manual de consistência pode se tornar uma tarefa complicada. Mecanismos de verificação de consistência de especificações são fundamentais para suportar o desenvolvimento de softwares, já que a verificação manual além de cansativa pode ser ineficiente.

⁴Refere-se a extensão por herança.

⁵Tais como especificação, modelo, conceito, mecanismo de armazenamento ...

Suporte a prototipação: A simulação de protótipos da interface do sistema é importante para a verificação de cobertura dos requisitos durante a etapa de projeto.

Referência cruzada: A possibilidade de comparar informações é fundamental para o desenvolvimento de especificações e facilita o entendimento de trechos de uma especificação. Possibilita ainda que o usuário navegue de forma mais intuitiva pela especificação.

Geração de código: É a capacidade de expressar as informações do repositório em uma linguagem de programação. Quanto mais código um ambiente puder gerar, melhor será o desenvolvimento, já que a possibilidade de inconsistência com o projeto diminui a medida que cresce o código gerado automaticamente.

Geração de documentos: A obrigatoriedade da utilização do ambiente para a visualização de uma especificação, além de ser restritivo pode não ser usual. Portanto é desejável que o ambiente suporte a geração de relatórios externos, ainda que semi-automatizada.

Suporte a reutilização: A reutilização possibilita a geração de softwares confiáveis em menos tempo. É desejável que o ambiente possibilite o reuso em vários níveis de granularidade, abrangendo tanto o reuso de código (como classes de uma biblioteca) quanto o reuso de projeto (como padrões de projeto).

3.2 Dependências

Alterações podem ser dificultadas pelas dependências do framework OCEAN, que se distinguem em 3 tipos: de linguagem, de visualização e de editores gráficos. Assim como qualquer software desenvolvido sobre uma linguagem, o OCEAN depende da linguagem que foi construída, o SmallTalk. A dependência do mecanismo de visualização é evidenciada pelo reuso do framework MVC no OCEAN.

Um dos diferenciais para a escolha do SmallTalk como linguagem é que o framework HotDraw⁶, indispensável para a construção de editores gráficos no framework OCEAN, está implementado nesta linguagem. Porém o uso do framework HotDraw é a dependência do OCEAN com conseqüências mais drásticas, tais como: consumo exagerado de memória, lentidão e incompatibilidade com versões mais atuais da linguagem.

⁶Framework para editores gráficos usado no OCEAN para representação visual de modelos.

3.3 Flexibilidade no framework OCEAN

O framework OCEAN suporta vários tipos de especificações de projeto, entretanto a sua flexibilidade não está restrita a este aspecto. As definições das funcionalidades previstas aos ambientes desenvolvidos sobre ele também são flexíveis. Os ambientes produzidos sob o framework OCEAN podem atuar sobre as especificações reusando estas funcionalidades da seguinte forma:

Funcionalidades supridas pelo framework OCEAN: são as funcionalidades gerais, podendo ser aplicadas a diversos ambientes. Algumas delas são totalmente definidas no framework, como as funcionalidades de navegação. Outras são definidas de forma genérica e precisam ser completadas em subclasses concretas de classes redefiníveis do OCEAN, como as funcionalidades referentes à criação de conceitos e modelos não previstos no framework, onde somente devem ser definidos os tipos de conceitos e modelos tratados.

Funcionalidades supridas através de editores de modelo: são as funcionalidades que só podem ser aplicadas por cada editor de modelo específico. Um editor pode ter uma ou várias capacidades, tais como criar, alterar ou remover conceitos em modelos de uma especificação. No entanto estas funcionalidades estão restritas aos tipos de conceitos definidos no modelo tratado e as funcionalidades de edição que o framework OCEAN fornece.

Funcionalidades supridas através de ferramentas: são as funcionalidades supridas para as ações de edição, análise ou transformação de conceitos, modelos ou especificações, diferindo dos editores de modelos por possuírem rotina automatizada, ou semi-automatizada, para a tarefa proposta. Assim como ocorre com os editores, estas funcionalidades reutilizam funcionalidades de edição supridas pelo framework OCEAN.

O uso de ferramentas não exige a alteração do restante da estrutura de documentos do ambiente⁷, e por este motivo constituem o meio de inserção de funcionalidade mais flexível dentre as possibilidades oferecidas pelo framework OCEAN. Além da flexibilidade, ferramentas são constituídas de rotinas automatizadas ou semi-automatizadas de edição semântica, e por esse motivo são consideradas poderosas formas de edição e podem ser usadas para *ações de edições complexas*.

⁷Exceto a alteração da especificação onde a ferramenta é declarada, pois no framework OCEAN é a especificação que determina quais ferramentas que podem atuar sobre ela.

3.4 Extensibilidade no framework OCEAN

Segundo Silva, o suporte a extensibilidade no framework OCEAN é dividida de 3 formas (SILVA, 2000):

- Suporte à Criação de Estruturas de Documentos;
- Suporte à Edição Semântica de Especificações;
- Suporte a Composição de Ações de Edição Complexas através de Ferramentas.

Neste trabalho será descrita a extensibilidade do framework OCEAN em relação ao seu foco, que podem estar relacionadas:

- ao ambiente - diz respeito à extensão de funcionalidades que o ambiente terá⁸.
- às especificações tratadas - diz respeito à extensão de uma especificação, definindo quais características específicas que essa especificação terá⁹.
- às ferramentas - diz respeito à extensão de estruturas funcionais de manipulação de especificações¹⁰.

3.4.1 Extensão em relação ao ambiente

Especificações de projeto de artefatos do software são documentos estruturados, que descrevem visões distintas de projeto. O framework OCEAN suporta a definição de diferentes estruturas de documentos, e para isso, provê uma definição genérica de estrutura de documento e também funcionalidades genéricas para visualização e armazenamento de dados.

Estrutura de documentos

A estrutura de uma especificação no OCEAN pode ser vista na figura 3.1, entretanto observa-se que esta não define somente a estrutura de uma especificação, mas também a estrutura de documentos em geral. Documentos genéricos são representados por *OCEANDocument*, especificações são representadas por *Specification* e elementos de especificação são representados por *SpecificationElement*, que podem ser modelos ou conceitos representados respectivamente

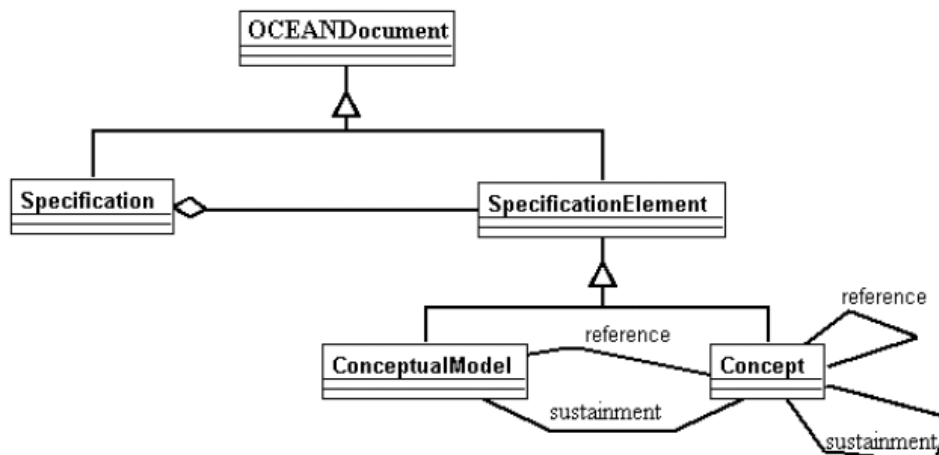


Figura 3.1: Superclasses do framework OCEAN que definem a estrutura de uma especificação (SILVA, 2000).

por *ConceptualModel* e *Concept*.

Desta forma define-se uma especificação como sendo uma entidade que agrega elementos de especificação, que podem ser modelos ou conceitos, e registra as associações de sustentação e referência¹¹ entre pares desses elementos.

A construção de uma especificação a partir da estrutura de documentos do framework consiste em definir um conjunto de tipos de modelo (subclasses concretas de *ConceptualModel*), um conjunto de tipos de conceito (subclasses concretas de *Concept*) e uma rede de associações entre os elementos destes conjuntos. A estrutura de um tipo de modelo é definida em função dos tipos de conceitos que são referenciados. Uma especificação OO do ambiente SEA, por exemplo, utiliza diagrama de classes (um tipo de modelo), que referencia classes, mas que não pode referenciar estados (isto é, instâncias destes tipos de conceito).

Visualização

A visualização de elementos de especificação resume-se basicamente ao gerente de referências¹², que relaciona cada elemento de especificação a um mecanismo de visualização, possibilitando o desacoplamento entre elementos de especificação e seus mecanismos de visualização.

⁸Ex: Criação de um ambiente que só armazena especificações em banco de dados.

⁹Ex: Criação de uma especificação que contenha somente digramas de caso de uso e diagramas de classe sem métodos e atributos.

¹⁰Ex: Ferramenta de partição de diagramas extensos.

¹¹Os tipos de associações do framework OCEAN serão vistas mais adiante.

¹²Instância de *ReferenceManager* ou de uma subclasse desta classe.

Um gerenciador de ambiente¹³ agrega um gerente de referências, que mantém um relacionamento *default* entre os elementos de especificação previstos no framework e os seus mecanismos de visualização. A criação de uma subclasse de *ReferenceManager* e o seu registro na classe *EnvironmentManager* possibilita a alteração dos relacionamentos *default* ou a inclusão de novos relacionamentos entre elementos de especificação não previstos no framework, e seus respectivos mecanismos de visualização.

Navegação

O mecanismo de memorização dos documentos visualizados é implementado na classe *Path*, cuja instância agrega células que referenciam os elementos de especificação visualizados ao longo do uso, assim como a especificação que o referencia.

Armazenamento

O mecanismo de armazenamento é representado pela classe *StoreManager*. Ela define um protocolo de armazenamento e recuperação de especificações, possibilitando que diversos mecanismos de armazenamento sejam implementados, como banco de dados por exemplo, sem que gere efeitos colaterais no restante do ambiente.

3.4.2 Extensão em relação à especificação

Desenvolver uma especificação consiste em criar, modificar ou remover elementos de especificação, isto é, modelos e conceitos. Basicamente a alteração de um modelo consiste em incluir e remover conceitos. A modificação de um conceito consiste na alteração de suas informações, podendo acarretar na inclusão e remoção de conceitos envolvidos em associações de sustentação ou referência.

O framework OCEAN suporta edição semântica de especificações onde modificações sobre algum elemento de especificação refletem em modificações em sua representação visual. A edição semântica faz com que as alterações sejam procedidas de maneira coerente, possibilitando o estabelecimento de restrições às ações de edição.

¹³Instancia de subclasse de *EnvironmentManager*.

Mecanismos de interligação de elementos de especificação

Na modelagem do tratamento de especificações verificou-se a necessidade de adotar mecanismos de ligação entre os elementos de especificação (conceitos e modelos) de forma que este mecanismo pudesse ser usado em diferentes contextos de modelagem. Por exemplo, para estabelecer a ligação entre um caso de uso, presente em um diagrama de casos de uso, e um diagrama de seqüência que o refine, ou para estabelecer uma ligação que defina que a existência de um diagrama de transição de estados está condicionada à existência da classe por ele modelado. O framework OCEAN oferece três mecanismos para realizar ligações entre elementos de especificação: *links*, associações de *sustentação* e associações de *referência*.

Um *link* é um tipo de conceito que pode ser associado a qualquer elemento de especificação para apontar outros documentos¹⁴. Os *links* podem ser usados para criar caminhos de navegação ou para estabelecer ligações semânticas entre elementos de especificação.

Um *link* usado para criação de caminho de navegação possibilita que um elemento de especificação seja referenciado. A remoção de *link* de navegação não implicará em uma incoerência na especificação, já que este serve exclusivamente para facilitar a navegação. Um exemplo seria a criação de *link* de navegação associado a uma classe que aponte para o diagrama de transição de estados de sua superclasse. O framework suporta a criação de *links* voltados para a definição de caminhos de navegação de forma manual ou automática.

Links usados para estabelecer ligações semânticas entre elementos de especificação são muito semelhantes a *links* de navegação, entretanto a sua ligação estabelece relações que podem afetar a coerência da especificação. Um exemplo de estabelecimento de ligação semântica através de *link* é a ligação entre um caso de uso e o diagrama de seqüência que o refine. Um *link* estabelece uma associação mantendo baixo acoplamento, de forma que, não afete diretamente a estrutura dos elementos envolvidos. A remoção de um dos elementos da especificação envolvidos na associação do *link*, não altera a estrutura do outro elemento envolvido, no entanto pode fazer com que a especificação se torne incompleta.

Uma associação de *sustentação* no framework OCEAN é a ligação semântica entre dois elementos de especificação distintos, de forma que um deles assume o papel de elemento sustentador e o outro de elemento sustentado. A existência de um elemento sustentado depende da existência do elemento sustentador, a remoção do elemento sustentador implica na remoção do elemento sustentado. A diminuição do acoplamento promovida por este tipo de mecanismo possibilita um maior reuso dos elementos de especificação. É o caso, por exemplo, da estrutura

¹⁴Pode apontar para uma especificação ou para um elemento de especificação, que pode ou não estar na mesma especificação do *link*.

de uma classe¹⁵ no ambiente SEA, onde métodos e atributos são associados às classes através de associações de sustentação, e por este motivo esta não prevê referência a eles. Desta forma, a entidade classe pode servir tanto para representar uma classe com métodos e atributos como para representar uma classe somente com atributos.

A associação de *referência* entre elementos de especificação é usada quando parte da definição de um elemento de especificação, o elemento referenciador, está dependente da referência a outro elemento de especificação, o elemento referenciado. Se o elemento referenciado for removido, o elemento referenciador não será removido, porém a sua estrutura ficará incompleta. Um exemplo é o caso da referência de uma mensagem (de diagrama de seqüência) a um método.

As associações de *sustentação e referência* entre elementos¹⁶ são registradas na estrutura de uma especificação através das tabelas de duplas, as chamadas tabelas de sustentação e tabela de referência.

Criação de modelos e conceitos

A criação de um elemento de especificação só é bem sucedida se o elemento a ser criado passar em todos os testes de verificação, que são os testes de duplicidade e teste de conflito.

O teste de duplicidade verifica se o elemento de especificação que está sendo criado apresenta coincidência de características com algum elemento já existente, como por exemplo, a criação de uma classe com o mesmo nome de uma classe existente.

O teste de conflito verifica se o elemento criado produz alguma inconsistência semântica na especificação, como por exemplo, a criação de uma herança cíclica.

Remoção de conceitos e modelos

A remoção de um modelo faz com que ele deixe de existir na especificação corrente. A remoção de um conceito pode fazer com que este deixe de existir na especificação, quando é referenciado por apenas um modelo, ou simplesmente deixe de ser referenciado por um modelo, quando existem vários modelos que possuam referência ao conceito a ser removido.

¹⁵Subclasse de *Concept* que define o conceito classe, para ser usado no diagrama de classes, por exemplo.

¹⁶Entre elementos de uma mesma especificação.

Alteração de conceitos

A alteração de um conceito pode acarretar na alteração de um elemento de especificação. O protocolo de mudança do framework OCEAN, implementado sobre o padrão de projeto *Observer*, permite que todos os elementos¹⁷ que referenciam o conceito tratado possam fazer alguma ação de alteração ou atualização, podendo propagar a notificação de mudança para outros elementos.

Para evitar que uma atualização cíclica¹⁸ ocorra, o protocolo de atualização do framework OCEAN prevê um mecanismo de parada, de forma que o elemento que já foi atualizado não receba a notificação novamente.

Transferência e fusão de conceitos

A funcionalidade de transferência consiste na troca do elemento sustentador de um conceito, um exemplo disto é transferência de um atributo de uma classe para outra. A fusão de dois conceitos de mesmo tipo faz com que um deles seja removido da especificação e transfere associações de sustentação e referência do conceito removido para o segundo conceito envolvido na fusão. O framework OCEAN prevê conjunto de regras que possibilita que estas funcionalidades sejam implementadas de maneira semântica.

Cópia e colagem de conceitos

A funcionalidade de cópia e colagem permite reusar conceitos existentes para a criação de novos conceitos, como por exemplo a criação de um novo método através da cópia de um método de uma classe existente e sua colagem em outra classe. O procedimento de cópia faz uso do mesmo mecanismo utilizado para registrar o caminho de navegação para gerar a área de transferência. Esta funcionalidade consiste em realizar verificações sobre a possibilidade da função ser executada. Em caso afirmativo é feita uma cópia do conceito alocando-o na área de transferência, e fazendo com que este passe a referenciar o conceito onde será colado. A área de transferência também pode referenciar um conjunto de conceitos, desde que sejam de uma mesma especificação.

¹⁷Refere-se a elementos de especificação ou a elementos de gráficos.

¹⁸Atualização cíclica é quando um elemento recebe uma notificação de mudança, procede a ação de atualização e acaba recebendo outra notificação para realizar a mesma ação de atualização.

3.4.3 Extensão em relação às ferramentas

Ferramentas no contexto do framework OCEAN são estruturas funcionais que reutilizam processos de edição semântica fornecidos pelo framework. Um ambiente construído sob o framework OCEAN agrega um gerente de ferramentas¹⁹. Estas ferramentas passam a ser acessíveis através do menu, podendo ou não ser selecionadas em um contexto específico²⁰ durante a operação do ambiente, além de poderem executar uma tarefa auxiliar durante a atuação de outra ferramenta.

O framework fornece a definição genérica de uma ferramenta através da classe *OceanTool*, que deve ser especializada para a geração de ferramentas específicas. Ferramentas são associadas a um ou mais tipos de especificação e podem estar associadas a um ou mais tipos de modelo. A figura 3.2 mostra os tipos de ferramentas, possuindo a seguinte classificação:

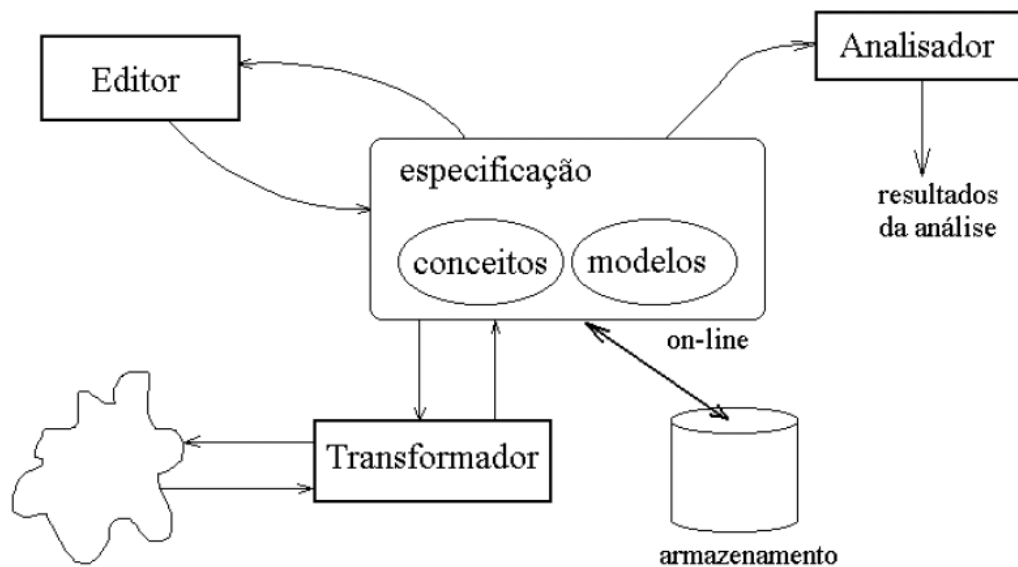


Figura 3.2: Tipos de ferramentas de ambiente sob o framework OCEAN (SILVA, 2000).

- Ferramentas de edição - tem a capacidade de ler a alterar uma especificação. Ajudam o usuário na tarefa de edição da especificação. Ex: Ferramenta para geração de métodos de acesso para atributos de uma classe.
- Ferramentas de análise - são ferramentas que lêem uma especificação, sem alterá-la, e geram descrições de características desta especificação. Estas descrições podem auxiliar

¹⁹Instância da classe *ToolManager* ou de uma subclasse dela.

²⁰Ex: Uma ferramenta de verificação de consistência de diagrama de classe só pode ser chamada se um diagrama de classes estiver aberto. Porém algumas ferramentas são projetadas para ser chamadas em qualquer contexto, como as de verificação de consistência da especificação ou geração de código para uma determinada linguagem.

no desenvolvimento, assim como podem ser somente de cunho estatístico ou descritivo.

Ex: Ferramenta para verificação de consistência de um modelo específico.

- Ferramenta de transformação - fazem a ligação entre as especificações e elementos externos. Tem a capacidade de importar ou exportar especificações. Ex: Ferramenta para geração de código em uma determinada linguagem.

4 *Ambiente SEA*

O ambiente SEA foi desenvolvido com o intuito de estender as classes do framework OCEAN e prevê o desenvolvimento e uso de artefatos de software reutilizáveis.

Para promover o reuso no desenvolvimento de software, o ambiente SEA possibilita a utilização integrada de abordagens de desenvolvimento orientado a componentes¹ e desenvolvimento baseado em frameworks, incluindo o uso de padrões de projeto. O ambiente SEA suporta o desenvolvimento de frameworks, componentes e aplicações como estruturas baseadas no paradigma de orientação a objetos.

O desenvolvimento de software² no ambiente SEA resume-se em produzir uma especificação de projeto deste artefato utilizando UML³, podendo converter essa especificação em código.

4.1 **Estrutura do ambiente**

A estrutura do ambiente SEA é suportada pelo framework OCEAN, pelo framework HotDraw e pelas bibliotecas de classe do SmallTalk. O ambiente possui um repositório de especificações que possibilita o compartilhamento com diversas ferramentas. O gerente de ambiente é responsável pela interface com o usuário e possibilita o uso de ferramentas para manipular as especificações contidas no seu repositório. O gerente de armazenamento é responsável pelo intercâmbio entre o repositório de especificações e os dispositivos de armazenamento, possibilitando que especificações sejam gravadas ou carregadas. A figura 4.1 mostra a estrutura do ambiente SEA.

¹Para suportar a construção de arquiteturas de componentes, por exemplo.

²Frameworks, componentes e aplicações.

³UML é utilizado com certas modificações.

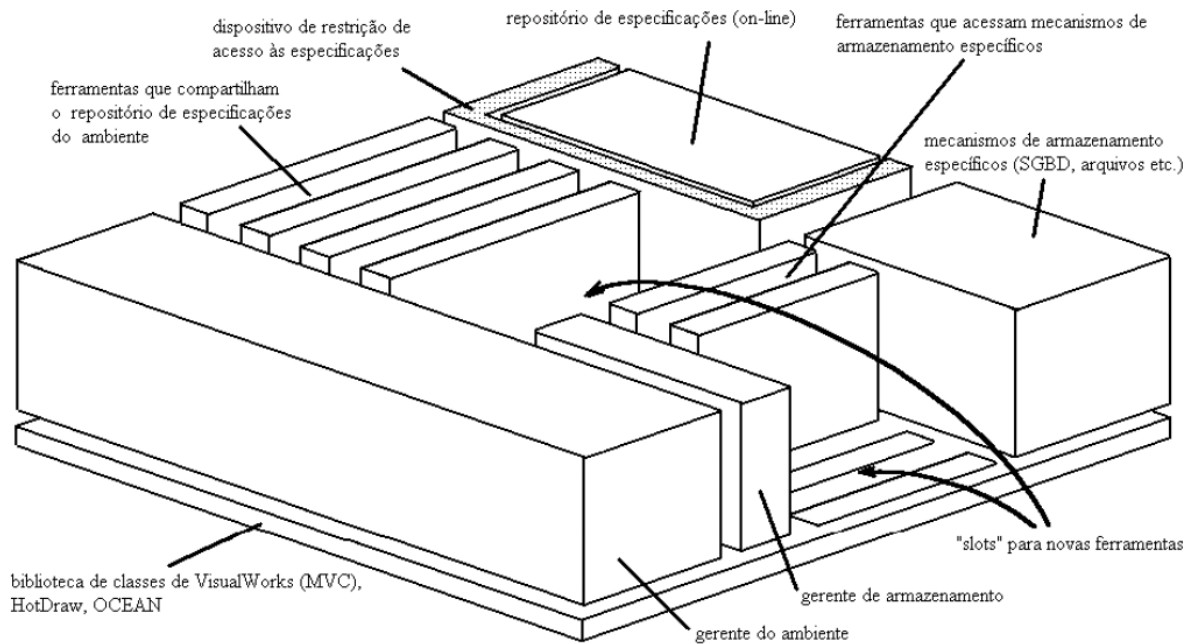


Figura 4.1: Estrutura do ambiente SEA (SILVA, 2000).

4.2 Requisitos desejáveis a um ambiente de desenvolvimento e uso de frameworks e componentes

Um ambiente que suporte as abordagens de frameworks orientados a objetos e desenvolvimento baseado em componentes deve suprir os requisitos específicos de cada uma destas abordagens, e possibilitar sua aplicação conjunta. Frameworks e componentes são artefatos que podem reutilizar outros frameworks, outros componentes e padrões de projeto. Desta forma é possível generalizar que artefatos de software (frameworks, componentes e aplicações) podem ser construídos reutilizando frameworks, componentes e padrões de projetos, simultaneamente ou não.

Silva destaca que um ambiente que suporte o desenvolvimento e uso de frameworks e componentes deve considerar os seguintes requisitos (SILVA, 2000):

Registro da flexibilidade de um framework: as informações que registram a flexibilidade de um framework são úteis para aprender a usar o framework, bem como para alterá-los;

Suporte à produção de artefatos a partir do framework: um ambiente que suporte o uso de frameworks deve possibilitar que um artefato de software seja desenvolvido estendendo a estrutura do framework, podendo dispor de recursos que ajudem neste desenvolvimento;

Suporte ao refinamento do framework a partir de suas aplicações: deve suportar que o fra-

mework agregue partes dos artefatos produzidos a partir dele, permitindo que, ao longo do seu uso, o framework incorpore informações do seu domínio a partir de suas aplicações;

Suporte à alteração do framework a partir de aplicações do domínio tratado: o suporte à manutenção deve possibilitar o reuso de estruturas de artefatos externos que atendam o domínio tratado para produzir e alterar frameworks;

Suporte ao desenvolvimento de componentes: o suporte à produção de componentes deve prever o desenvolvimento de interfaces e o de estruturas internas de componentes como atividades distintas, possibilitando que uma mesma especificação de interface possa ser usada por mais de um componente;

Suporte à interconexão de componentes: deve ser possível especificar um artefato de software a partir da interconexão de componentes através de seus canais de comunicação;

Suporte ao uso simultâneos de frameworks e componentes: deve ser possível reutilizar frameworks e componentes de maneira simultânea para a produção de artefatos de software;

Suporte ao uso de padrões em especificações: um ambiente voltado para o reuso deve possibilitar que partes da especificação sejam feitas de uma maneira padrão, promovendo o reuso de experiência de projeto;

Suporte ao uso de padrões arquitetônicos: deve ser possível especificar o estilo arquitetônico de um artefato desenvolvido através da interconexão de padrões arquitetônicos existentes, promovendo o reuso de arquiteturas na fase de projeto.

4.3 Tratamento de frameworks no ambiente SEA

A abordagem de frameworks prevê requisitos de modelagem que em geral não são atendidos por metodologias OOAD em geral. Os requisitos podem ser focados na especificação de aplicações sob o framework, sobre as classes ou sobre os métodos do framework.

4.3.1 Auxílio à descrição de aplicações

O uso de frameworks está relacionado à criação de novos artefatos de software (framework, aplicações, componentes), e para que isto seja possível é necessário relacionar a especificação do artefato de software com a especificação do framework que o gerou.

A descrição de uma aplicação gerada a partir de um framework não deve ter somente um atalho à especificação do framework, mas sim uma ligação semântica. Essa ligação semântica deve possibilitar que o ambiente funcione como um compilador, que tenha a capacidade de determinar a coerência entre a especificação da aplicação e a do framework. Uma especificação no ambiente SEA pode apontar para uma ou mais “especificações-mãe”.

4.3.2 Auxílio à descrição de classes

Frameworks mantêm partes flexíveis, essa flexibilidade reside nas classes que podem ou devem ser redefiníveis através da criação de subclasses. Para o registro explícito da flexibilidade de classes de um framework, foi introduzido no ambiente SEA, além da classificação de concreta e abstrata, as propriedades de redefinibilidade e essencialidade, descritas abaixo:

Redefinibilidade de classe: estabelece as classes que podem ou não gerar subclasses nos artefatos de software desenvolvidos sob o framework. Uma classe sinalizada como redefinível equivale a registrar explicitamente que esta classe pode ser usada para a geração de novas subclasses. Apenas é prevista a criação de subclasses de classes redefiníveis.

Essencialidade de classe: sinaliza se uma classe é essencial ou não para o uso do framework. Apenas classes redefiníveis podem ser classificadas como essenciais. Registrar uma classe como essencial corresponde que todo artefato de software produzido a partir do framework deve utilizar esta classe (ou uma subclasse dela).

4.3.3 Auxílio à descrição de métodos

O ambiente prevê a explicitação da classificação dos métodos como *abstrato*, *template* ou *base*. Método abstrato é um método que possui apenas a assinatura na classe abstrata, e que precisa ser sobrescrito nas subclasses para definição do seu corpo. Método template é um método que é definido em termos de outros métodos, é um método que tem como função chamar outros métodos, estabelecendo um algoritmo através da ordem lógica dos métodos chamados (*hook*). Fornecem uma estrutura de algoritmo que permite a flexibilidade através dos métodos hook chamados. Métodos base são métodos que possuem uma implementação “default” e não precisam ser alterados, mas entretanto é possível sobrescrevê-los.

4.4 Especificações no ambiente SEA

As especificações do ambiente SEA usam a estrutura de especificações manipuláveis pelo framework OCEAN. Esta estrutura suporta a criação de especificações distintas, através da definição de subclasses concretas. A abordagem do framework OCEAN prevê que uma especificação agrega elementos de especificação, que podem ser modelos ou conceitos, e um elemento de especificação pode estar associado a outros elementos de especificação.

A primeira versão do ambiente SEA suporta 3 tipos de especificação:

- Especificação OO, que pode descrever um framework, uma aplicação, um componente ou um componente flexível;
- Especificação de interface de componente;
- *Cookbook* ativo, para orientar o uso de um framework previamente desenvolvido.

4.4.1 Especificação OO

Uma especificação OO consiste em uma estrutura de classes que pode corresponder à especificação de projeto de um framework, de uma aplicação, de um componente ou de um componente flexível. No contexto do ambiente SEA uma especificação OO é utilizada para designar uma especificação baseada no paradigma de orientação a objetos e produzida a partir do uso de notação de alto nível, no caso a UML.

Adaptações

Segundo Silva (SILVA, 2000) as “notações de metodologia OOAD em geral, incluída a notação de UML, não são totalmente adequadas à modelagem de frameworks por não possibilitarem a modelagem desta flexibilidade e nem a ligação semântica entre especificações distintas”. Isto acarreta em uma especificação incompleta de frameworks, e não possibilita registrar a ligação entre a especificação de um artefato de software e a especificação do framework que a originou.

Nem todos os elementos sintáticos previstos nas técnicas de UML foram incorporados aos editores do ambiente SEA, o que resulta em uma diminuição da expressividade original. Entretanto, foram introduzidas as seguintes extensões no ambiente SEA para (SILVA, 2000):

- representar conceitos do domínio de frameworks, não representáveis nas técnicas de UML;
- agregar recursos de modelagem a UML, como a possibilidade de estabelecimento de restrições semântica aos elementos da especificação;
- possibilitar a descrição do algoritmo dos métodos na especificação de projeto;
- expressar a ligação semântica entre elementos de uma especificação e entre especificações distintas;
- possibilitar que especificações sejam tratadas como hiperdocumentos, de modo que os elementos de uma especificação possuam *links* associados que apontem para outros documentos.

Diagramas previstos para especificação OO

O ambiente SEA não utiliza todas as técnicas de modelagem previstas em UML⁴, se atendo a cinco destas técnicas, e a mais uma técnica adicional para a descrição do algoritmo dos métodos, que não é prevista em UML.

Nas descrições das técnicas a seguir, são apresentados como exemplo, modelos da especificação do framework FraG⁵:

Diagrama de casos de uso: é a descrição que delimita as situações de processamento para o artefato de software em questão, registrando os eventos tratados por cada entidade que acessa o sistema. O diagrama de casos de uso é composto por casos de uso, atores e relações que interligam atores e casos de uso. A figura 4.2 apresenta o diagrama de casos de uso do framework FraG.

Diagrama de atividades: é composto por casos de uso (que correspondem às atividades) e transições entre casos de uso. No ambiente SEA este diagrama permite estabelecer restrições quanto às seqüências de situação de processamento do sistema modelado. Uma atividade pode ser classificada como inicial (representada por um contorno grosso) quando esta representa o primeiro processamento do sistema, todas as outras atividades devem ser alcançáveis a partir da atividade inicial. A figura 4.3 apresenta o diagrama de atividades do framework FraG.

⁴As técnicas de UML usadas na especificação OO do ambiente SEA se baseiam na versão 1.4 da UML, sendo que o diagrama de estados é a única técnica que se difere da notação original.

⁵Framework para jogos de tabuleiro também desenvolvido por Ricardo Perreira e Silva.

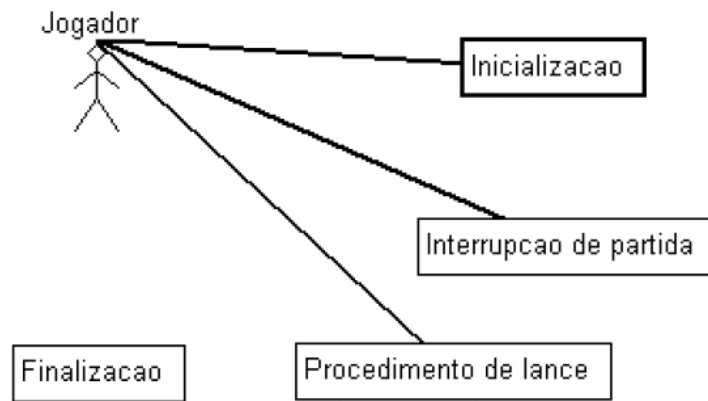


Figura 4.2: Diagrama de casos de uso do framework FraG (SILVA, 2000).

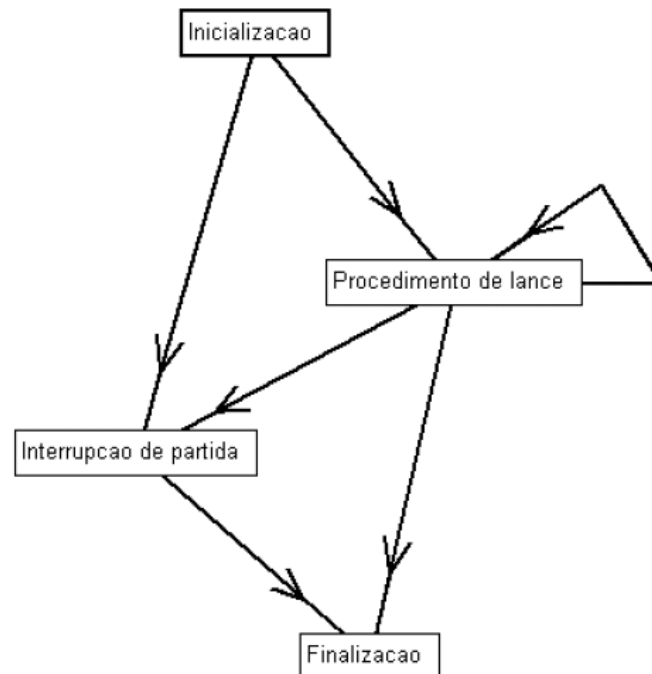


Figura 4.3: Diagrama de atividades do framework FraG (SILVA, 2000).

Diagrama de classes: ilustra as entidades do domínio tratado e seus relacionamentos. No ambiente SEA o diagrama de classes é composto pelos conceitos de classe, com seus atributos e métodos, associação binária, agregação e herança. O diagrama de classes no ambiente prevê representação visual da classificação de conceitos externos, da classificação de classe quanto à redefinibilidade e essencialidade, e classificação de métodos em abstrato, base ou template, além de prever a diferenciação entre classes abstratas e classes concretas. A figura 4.4 mostra parte de um diagrama de classes do framework FraG.

Diagrama de transição de estados: é composto por estados e as transições entre eles e é as-

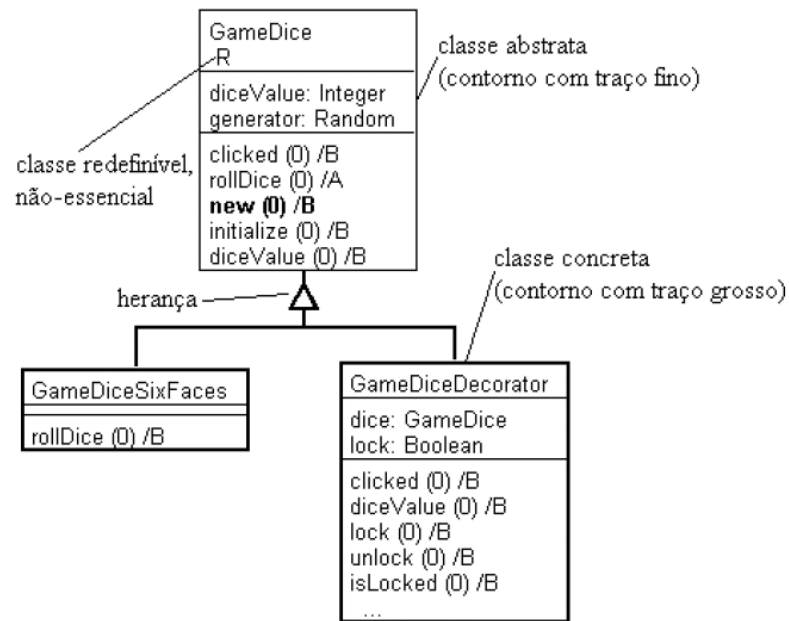


Figura 4.4: Parte de um diagrama de classes do framework FraG (SILVA, 2000).

sociado a uma classe. Difere-se da notação original da UML por permitir que os estados sejam definidos em termos dos valores assumidos pelos atributos da classe modelada. No ambiente SEA é possível refinar um estado em um diagrama separado. A figura 4.5 mostra o diagrama de transição de estados da classe *GameUserInterface* do framework FraG.

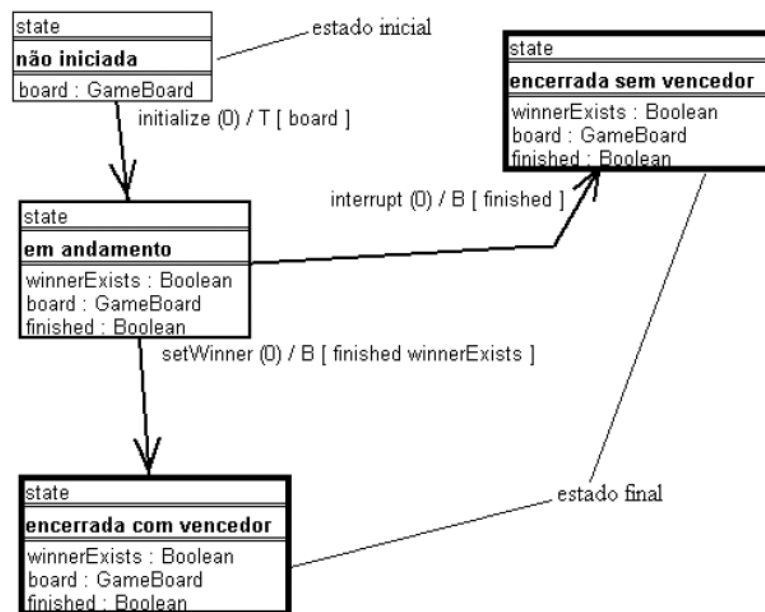


Figura 4.5: Diagrama de transição de estados da classe *GameUserInterface* do framework FraG (SILVA, 2000).

Diagrama de seqüência: é composto de mensagens e elementos que trocam mensagens. Mostra a interação de mensagens dentro de um caso de uso. No ambiente SEA uma mensagem pode ter um condicionador de ocorrência associado, e a classificação dos métodos e classes possui uma representação visual correspondente. A figura 4.6 mostra parte de um diagrama de seqüência do framework FraG.

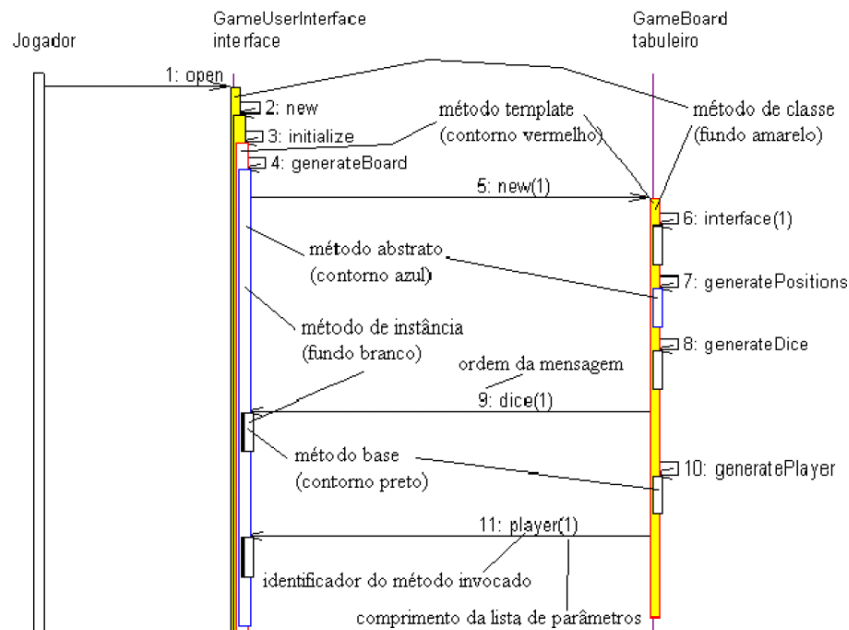


Figura 4.6: Parte de um diagrama de seqüência do framework FraG que refina o caso de uso “inicialização” (SILVA, 2000).

Diagrama de corpo de método: não previsto em UML, dispõe de comandos para a descrição dos algoritmos dos métodos. Esses comandos podem ser convertidos para diferentes linguagens de programação orientadas a objeto. A figura 4.7 mostra o diagrama de corpo de método do método *controlActivity* da classe *ClickController* do framework FraG.

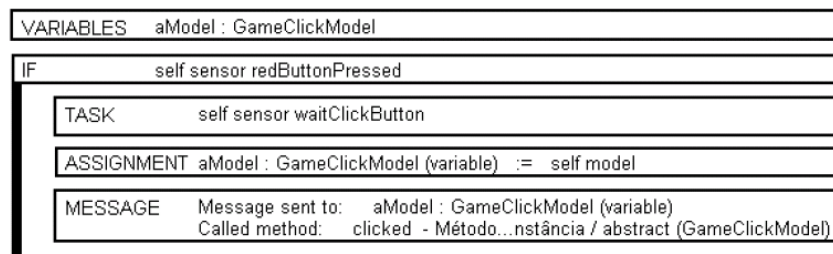


Figura 4.7: Diagrama de corpo de método do método *controlActivity* da classe *ClickController* do framework FraG (SILVA, 2000).

Funcionalidades e ferramentas no ambiente SEA

Para suportar a produção de especificações OO, o ambiente SEA apresenta um conjunto de funcionalidades, que podem estar incorporadas às ferramentas. A seguir é apresentada a descrição destas funcionalidades:

Consistência de especificações OO: uma especificação OO no ambiente SEA apresenta semântica definida e uma gramática de atributos. Isto possibilita a verificação de consistência de especificações OO, bem como a conversão de uma especificação em outras linguagens. Essa consistência de uma especificação OO é garantida pelo seguinte conjunto de recursos do ambiente (SILVA, 2000):

- ações de edição bloqueadas - toda ação de edição que envolva criação ou modificação de elemento de especificação é submetida à avaliação de conflito, que pode impedir a sua consumação;
- ações de edição impossibilitadas - certas ações não são possíveis de realizar em função das interfaces de edição do ambiente SEA;
- o ambiente apresenta sete ferramentas de análise para verificação de consistência de especificações OO, uma ferramenta de análise para cada um dos seis tipos de modelo e uma ferramenta de análise do conjunto da especificação.

Alteração de relações semânticas: uma especificação contém conceitos e as relações semânticas registradas nas tabelas de sustentação e referência, segundo a abordagem de definição de estruturas de especificação adotada no framework OCEAN. A transferência e a fusão de conceitos são ações de edição semântica disponibilizadas no ambiente SEA, restritas ao contexto de modelos, que alteram essas relações semânticas entre conceitos.

Reuso de conceitos por cópia e colagem: cópia e colagem são um dos recursos de edição semântica do ambiente SEA que usa área de transferência do ambiente, possibilitando o reuso de estruturas de conceito já criadas.

Criação automática de métodos de acesso aos atributos: o ambiente SEA possui ferramentas para criação automática de métodos de acesso aos atributos, capazes de criar toda a estrutura de um método, isto é, assinatura e corpo.

Suporte à composição do diagrama de corpo de métodos: o ambiente SEA possui uma ferramenta que varre os diagramas de seqüência e os diagramas de transição de estados e

gera comandos *external*, que fornece apoio à construção de diagramas de corpo de método. A ferramenta produz automaticamente esses comandos no diagrama de corpo de método sob edição. Esses comandos registram a participação do método que está sendo modelado naqueles diagramas, e que podem ser usados para facilitar a construção dos diagramas de corpo de método.

Suporte para alteração de frameworks: o ambiente SEA possui uma ferramenta de transferência de partes de especificação para o framework. Esta ferramenta automatiza o procedimento de alterar o framework para suportar novas aplicações, cuja principal vantagem é evitar a introdução de erros no procedimento desta transferência, que envolve um grande número de ações de edição.

Suporte a padrões de projeto: padrões de projeto são estruturas de classes que correspondem a soluções para problemas de projeto. O desenvolvimento no ambiente SEA corresponde à criação de uma especificação de projeto para posterior geração de código. O ambiente SEA possui uma ferramenta de inserção semi-automática de padrões, que permite selecionar uma especificação de padrão de projeto na biblioteca de padrões do ambiente, e inseri-la em uma especificação em desenvolvimento. Além disto, o SEA também suporta o desenvolvimento de novas especificações de padrões de projeto, e sua inclusão na biblioteca do ambiente.

Geração de código: o ambiente SEA dispõe de uma ferramenta com a capacidade de gerar código SmallTalk executável. Entretanto é necessário que a especificação contenha todas as informações necessárias à geração do código, e que estas informações estejam consistentes.

4.4.2 *Cookbook* ativo - Suporte ao uso de frameworks

O ambiente SEA possibilita a criação de *cookbooks* ativos, que dão suporte ao uso de frameworks através do fornecimento de instruções que auxiliam na tarefa de construir uma aplicação a partir do framework.

Cookbooks ativos são mais que meros manuais, são hiperdocumentos que possuem links ativos que permitem ações de edição automatizada. Entretanto eles são incapazes de descrever todas as possibilidades de criação de artefatos de software a partir de um framework (SCHEIDT, 2003).

A adoção da abordagem de *cookbook* ativo no ambiente SEA foi possibilitada pelo estabelecimento de alguns requisitos para um mecanismo de suporte ao uso de frameworks: dire-

cionar as ações dos usuários de frameworks, reduzir o esforço para entender o projeto de um framework, liberar o usuário de atividades de baixo nível e verificar a satisfação dos requisitos para a geração de aplicações a partir de um framework.

4.4.3 Especificação de interface de componente

A especificação estrutural de uma interface de componente relaciona os métodos fornecidos, os métodos requeridos e a associação a cada canal da interface. A estrutura de uma interface no ambiente SEA é produzida relacionando assinaturas de métodos e canais, e definindo a ligação entre eles. Outra questão que deve ser tratada em uma especificação de interface de componente é a sua descrição comportamental da interface, que está relacionada com a existência ou inexistência de restrições à ordem de invocação dos métodos.

5 *Reengenharia do Framework OCEAN*

Este capítulo descreve a reengenharia do framework OCEAN, focando nas partes inerentes ao framework e compreendendo a conversão para Java, a validação, as adaptações e as melhorias propostas por esta reengenharia. Porém o framework OCEAN não é um artefato executável, e por isso é necessário usar uma aplicação para validar a reengenharia. Para isto, será convertido também o ambiente SEA, que é ambiente mais completo produzido a partir do framework, assim como a estrutura e todos os elementos de uma especificação orientada a objetos, que é a especificação mais usual e abrangente. Desta forma, esta reengenharia é caracterizada pela conversão do framework OCEAN, do ambiente SEA e da especificação OO¹ para uma linguagem de programação mais atual, motivada pela necessidade de modificar o framework para que o seu processo de manutenção seja favorecido.

A conversão de linguagem de programação de um artefato de software não é algo trivial, isto porque envolve três fatores bem distintos: a linguagem de programação de origem, o artefato de software e a linguagem de programação de destino. Além da complexidade envolvida na associação destes três fatores, existe a complexidade isolada de cada fator. Neste trabalho, a linguagem de programação origem é o SmallTalk, o artefato de software é framework OCEAN, e a linguagem de programação de destino é Java.

Quando um software começa a ser utilizado, ele entra em estado contínuo de mudança. Essas mudanças geralmente ocorrem sem que haja preocupação com a arquitetura geral do sistema, acarretando em uma estrutura gradativamente “depreciada”, de forma que, cada mudança necessária ao sistema tenda a se tornar cada vez mais cara. Quando o sistema não é de fácil manutenção, e este é de grande importância, ele deve ser reconstruído (JACOBSON; LINDSTRÖM, 1991).

¹Originalmente implementados na linguagem de programação SmallTalk.

5.1 Engenharia reversa

O processo de quando se parte dos requisitos, avançando nas fases do processo tradicional de desenvolvimento até que se chegue no software é o que denominados de “engenharia progressiva”. Como o próprio nome diz, a engenharia reversa percorre o caminho inverso ao da engenharia progressiva, aonde se parte do software, ou do seu código, até chegar a um nível de abstração mais alto, podendo ou não chegar até o nível de abstração dos requisitos do sistema. O principal objetivo da engenharia reversa é o entendimento do comportamento e da estrutura de um sistema, ou de parte dele (PIEKARSKI; QUINÁIA, 2000).

O resultado da engenharia reversa pode possuir várias visões, que representam diferentes níveis de abstração. Essas visões podem ser, começando do nível mais baixo de abstração e indo para o mais alto: visão em nível implementacional, visão em nível estrutural; visão em nível funcional ou visão em nível do domínio. A figura 5.1 ilustra essas visões no ciclo de desenvolvimento de software.

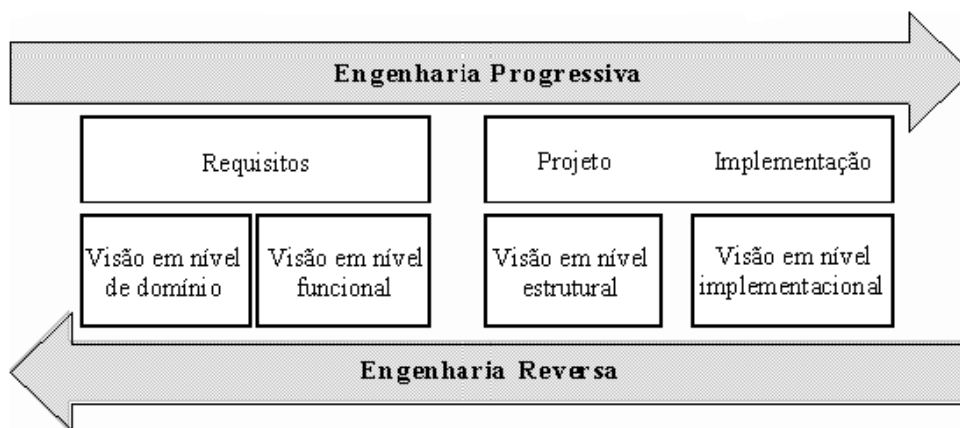


Figura 5.1: Visualizações de software no ciclo de desenvolvimento (COSTA, 1997 apud PIEKARSKI; QUINÁIA, 2000).

Além das visões, é possível categorizar o resultado da engenharia reversa de acordo com o entendimento obtido (PIEKARSKI; QUINÁIA, 2000):

Visualização de código: tem como intenção gerar uma outra forma de representar o código, tornando mais fácil o seu entendimento. Visa recuperar a documentação do sistema.

Entendimento de programa: tem como objetivo entender o sistema como um todo, utilizando para isso uma combinação de código, documentação existente, experiências pessoais e conhecimentos gerais sobre o problema e o domínio de aplicação. Visa a recuperação do projeto.

A figura 5.2 mostra o relacionamento dessas categorias com visões da engenharia reversa.

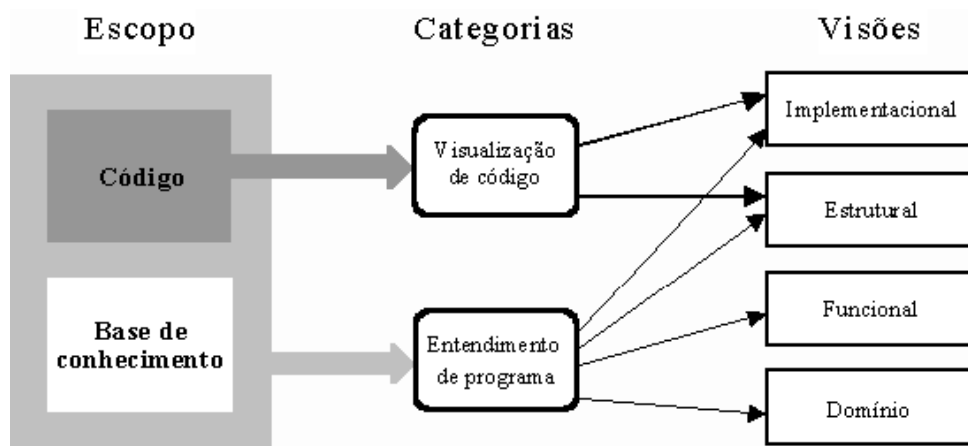


Figura 5.2: Categorias da engenharia reversa e suas visões (PIEKARSKI; QUINÁIA, 2000).

O principal propósito da engenharia reversa é incrementar o conhecimento sobre todo o sistema tanto para manutenção quanto para estendê-lo, e para que isto seja possível é necessário que a engenharia reversa supra os seguintes objetivos (CHIKOFFSKY; CROSS, 1990):

- diminuir a complexidade;
- gerar visões alternativas;
- recuperar informações perdidas;
- detecte efeitos colaterais;
- sintetizar abstrações;
- facilitar o reuso.

5.2 Reengenharia

Um software tende a sofrer alteração com o passar do tempo, ou para atender novos requisitos, ou para aperfeiçoar os requisitos existentes, ou para melhorar o desempenho ou para consertar erros que não foram pegos na fase de desenvolvimento (FONTANETTE; PRADO; OLIVEIRA, 2004). Com o passar do tempo a alteração no software tende a ficar mais custosa, em função das alterações anteriores ou da obsolescência da tecnologia em que o software foi desenvolvido.

Para Chikofsky e Cross (CHIKOFSKY; CROSS, 1990) e Jacobson e Lindström (JACOBSON; LINDSTRÖM, 1991), reengenharia é análise e alteração de um artefato de software com a intenção de reimplimentá-lo de uma nova forma, que inclui um processo de engenharia reversa seguido de uma engenharia progressiva. Jacobson e Lindström afirmam ainda que além da engenharia reversa e da engenharia progressiva, a “fórmula” da reengenharia conta com um “ Δ ”, onde este pode ser de dois tipos:

- mudança de funcionalidade - altera ou adiciona alguma funcionalidade, alterando parcialmente o objetivo principal do sistema;
- mudança da técnica de implementação - altera a forma de implementação do sistema.

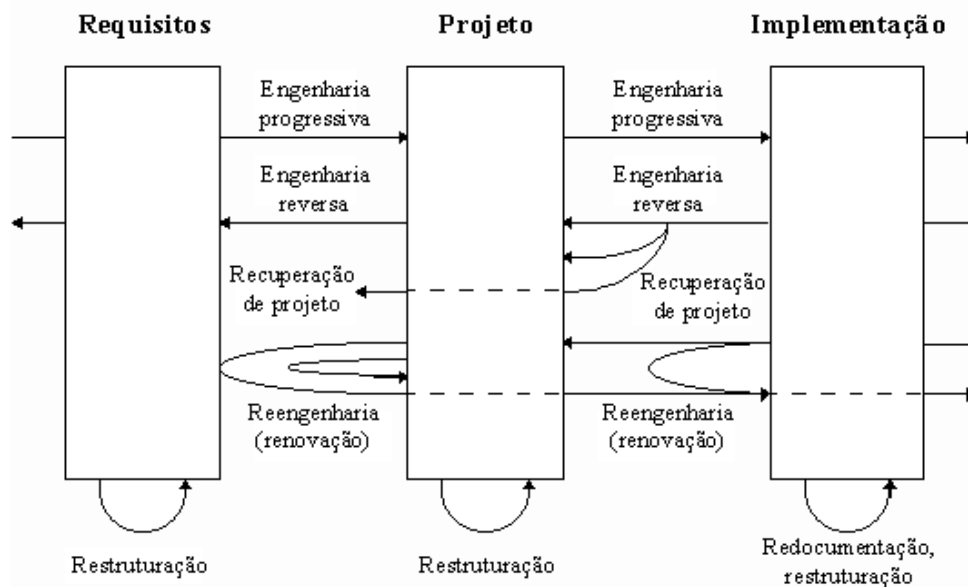


Figura 5.3: Relacionamentos no ciclo de desenvolvimento de software (CHIKOFSKY; CROSS, 1990 apud PIEKARSKI; QUINÁIA, 2000).

A reengenharia favorece o uso do software, uma vez que este tende a ser tornar mais atual, entretanto o maior ganho está relacionado à manutenção. O ganho no esforço e no custo de uma manutenção devem ser significativos após a reengenharia, caso não sejam, a reengenharia pode ter sido inútil. Para Jacobson e Lindström (JACOBSON; LINDSTRÖM, 1991) a reengenharia do sistema depende da mutabilidade e da importância do artefato de software, que pode ser visto na figura 5.4, de forma que a reengenharia só é compensatória quando a mutabilidade é baixa e a importância é alta.

O processo de uma reengenharia pode ser dividido em três partes (JACOBSON; LINDSTRÖM, 1991 apud PIEKARSKI; QUINÁIA, 2000):

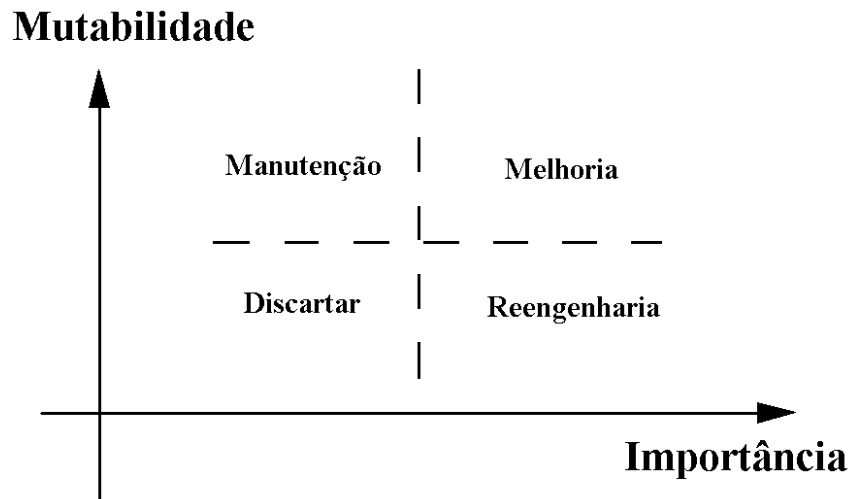


Figura 5.4: Matriz de decisão do que fazer com o sistema antigo (JACOBSON; LINDSTRÖM, 1991).

Realizar a engenharia reversa: identificar os componentes e suas relações, ou seja, aprender sobre o sistema de forma que seja possível gerar uma descrição mais abstrata;

Decidir sobre alterações na funcionalidade: discutir as funcionalidades do novo artefato de software em relação ao antigo;

Reprojetar o sistema: é a execução da engenharia progressiva com base nas informações do novo sistema. Neste passo, deve-se levar em consideração se houve mudança nas técnicas de implementação, e qual a porcentagem do sistema novo que mudou em relação ao antigo.

O processo de reengenharia deve ter, na medida do possível ferramentas que auxiliem os desenvolvedores nesta tarefa. O processo manual de reengenharia pode ser desencorajador, inviavelmente custoso, e pode gerar erros devido à complexidade do artefato de software, podendo acarretar em uma diferença de comportamento entre o novo artefato de software e o antigo.

5.3 Engenharia reversa no framework OCEAN

A reengenharia proposta para o framework OCEAN prevê que a sua fase de engenharia reversa proporcione um entendimento de todo o framework e não somente um entendimento do que os métodos fazem. Para isto é necessário o conhecimento sobre os artefatos que dependem do framework OCEAN, assim como sobre os artefatos que o framework depende. Este processo

de aprendizado necessário até que se chegue ao entendimento global do framework OCEAN é ilustrado na figura 5.5 e compreende as seguintes etapas:

- base de dados OCEAN - consiste em se aprofundar teoricamente sobre o framework OCEAN e o seu domínio; A principal fonte de informações foi a tese de doutorado originada com o framework;
- usar aplicações do OCEAN - é executar aplicações feitas sobre o framework OCEAN. A aplicação que foi usada é a mais importante que foi desenvolvida com o framework, o ambiente SEA. A utilização do SEA, efetuada no modo normal e no modo de depuração, possibilitou a criação de diversos diagramas que ajudam a entender quais as funcionalidades oferecidas pelo framework;
- conhecimento em SmallTalk - uma vez que o framework foi feito em SmallTalk é indispensável o conhecimento básico sobre esta linguagem de programação para o entendimento global do framework OCEAN. Tutoriais, como os que acompanham o ambiente VisualWorks, são muito eficientes para quem não conhece sobre a linguagem SmallTalk e deseja aprender sobre ela. Os tutoriais possibilitaram a criação de uma calculadora e de um jogo da velha, aqui chamado como JVelhaSolo.
- conhecimento em HotDraw - o conhecimento básico sobre o framework de editores gráficos usado pelo OCEAN, o HotDraw, é aconselhável para o entendimento global do framework OCEAN. Como o HotDraw tem muito pouca documentação, foi estudado o código dos editores criados com o framework OCEAN;
- estender OCEAN - criar aplicações usando o framework OCEAN possibilita uma visão diferente da simples execução de uma aplicação pronta, e por isso também é aconselhável para o seu entendimento global do framework. A partir de roteiros desenvolvidos pelo criador do framework foi possível criar um diagrama de fluxo de dados que possibilitou verificar como o framework funciona, ilustrado na figura 5.6.

Vale ressaltar que apenas seguir superficialmente as etapas de aprendizagem do framework OCEAN não faz com que uma pessoa entenda completamente o mesmo. É necessário que a verificação e o estudo de cada etapa tenha profundidade. Para que se possuísse o conhecimento sobre o framework, a passagem por cada etapa teve a preocupação de tentar cobrir todos os casos da mesma. Como exemplo, tem-se a etapa de uso do ambiente SEA, onde houve a preocupação em criar-se o maior número possível de tipos de modelos e de conceitos. A tarefa de aprender sobre um framework é complexa e desmotivadora, pois inicialmente tem-se muito esforço e pouco resultado. Com o framework OCEAN não foi diferente, porém tanto na etapa de

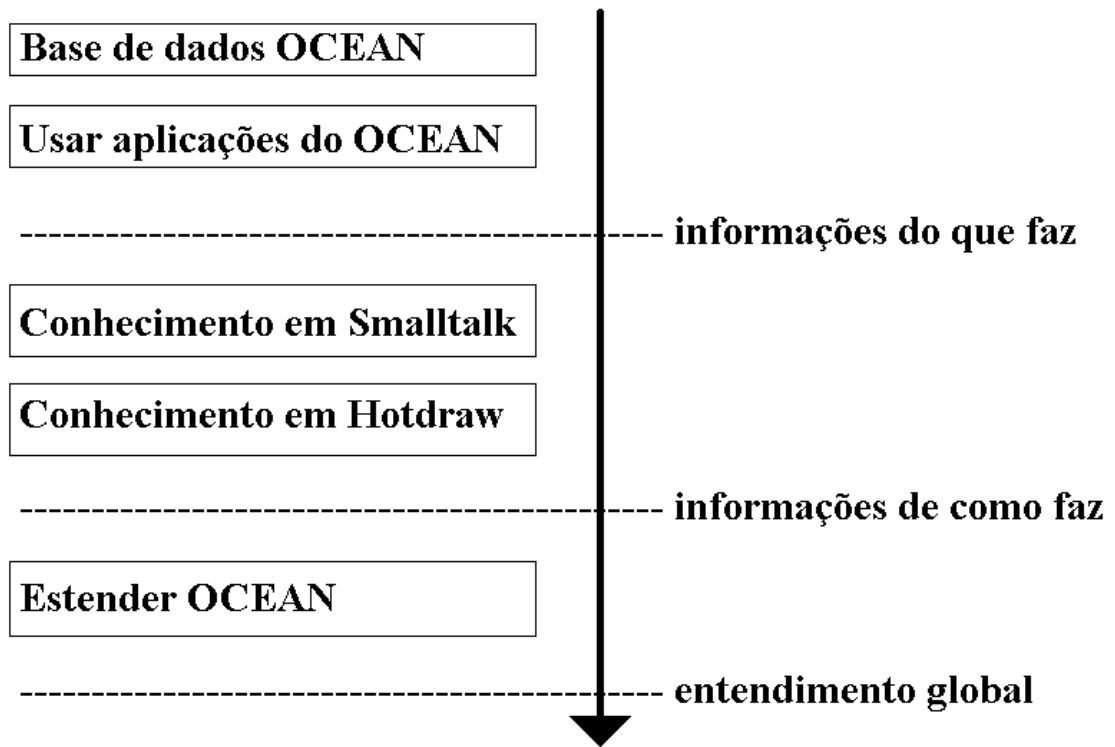


Figura 5.5: Etapas para entendimento global do framework OCEAN.

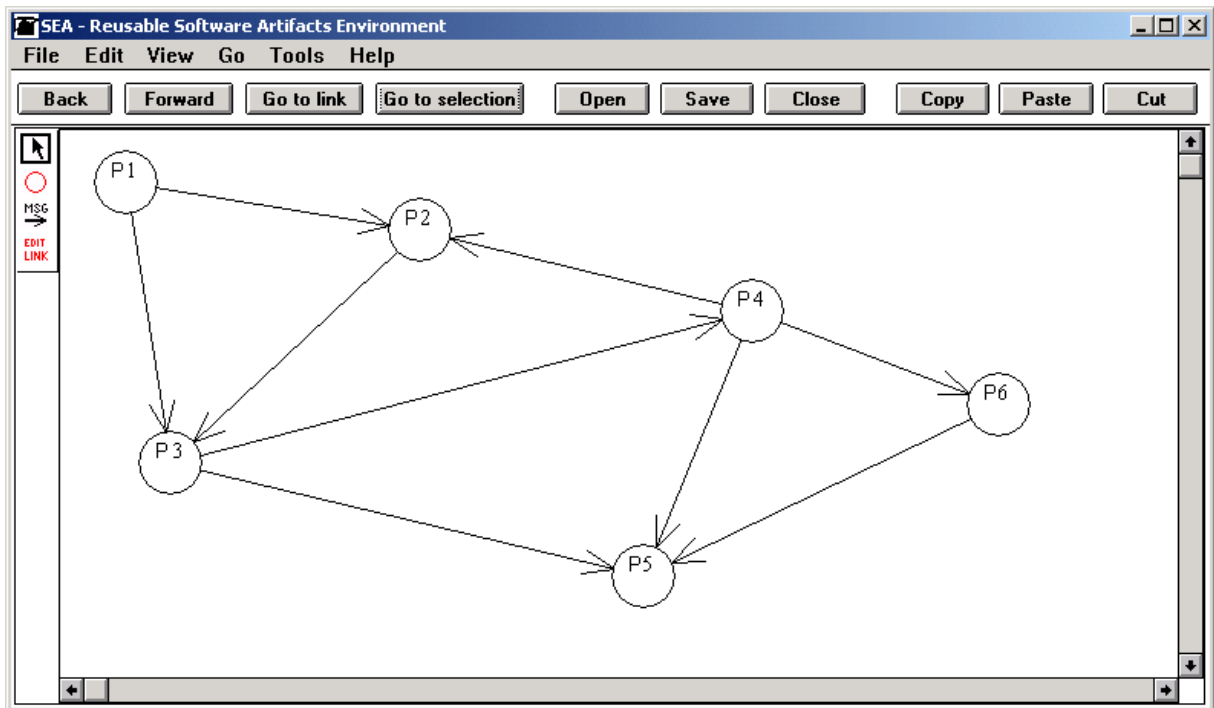


Figura 5.6: Diagrama de fluxo de dados criado estendendo o ambiente SEA em SmallTalk.

aprendizado quanto na de reimplementação teve-se o suporte do criador do framework, fazendo com que estas tarefas não fossem tão árduas.

5.4 Reimplementação do framework OCEAN

A linguagem de programação em que seria reimplementado o framework OCEAN deveria ser uma linguagem atual e difundida, de forma que alterações no framework fossem menos árduas. Além disto, era requerido que possuísse um artefato implementado nesta linguagem que possibilitasse a criação de editores gráficos, para que pudesse servir como artefato equivalente ao HotDraw em SmallTalk. Estes requisitos fizeram com que a linguagem de programação escolhida fosse Java, pois: é uma linguagem bem conhecida entre os desenvolvedores; é independente de plataforma; existe uma gama de artefatos implementados nesta linguagem e existe uma versão do HotDraw implementado em Java, o JHotDraw.

Escolhida a linguagem de programação em que o framework OCEAN seria reimplementado, foi decidido que a versão em Java deveria possuir:

- tratamento diferenciado dos objetos visuais - na versão em SmallTalk os objetos visuais dos editores são armazenados como elementos persistidos juntamente com a especificação, acarretando em especificações demasiadamente grandes, em termos de armazenamento;
- classes renomeadas com nomes mais apropriados - consiste em uma padronização dos nomes de acordo com a sua função, possibilitando uma maior compreensão das mesmas;
- otimização dos métodos - percebeu-se que muitos métodos não tinham a preocupação com a otimização, uma reestruturação nos métodos facilitaria a sua compreensão e melhoraria o tempo de execução;
- documentação - a documentação de código, que é quase inexistente na versão em SmallTalk, possibilitaria um entendimento mais rápido e favoreceria a sua manutenção.

5.4.1 Ferramentas de conversão

Uma atividade crítica neste trabalho é a conversão semi-automática do código de OCEAN e SEA, que é bastante extenso (mais de 250 classes), e para tanto é necessário o uso de uma ferramenta. Para isso estudou-se a possibilidade de construir uma nova ferramenta de conversão, mas que foi descartada em função de que esta alternativa fugiria do foco do trabalho, já que é

previsto o uso, e não a criação, de um mecanismo de conversão de código SmallTalk em código Java. Então a solução natural foi encontrar ferramentas prontas que fizessem esse trabalho inicial de conversão.

Para avaliar as ferramentas utilizadas na tarefa de conversão inicial de SmallTalk para Java foram adotados os seguintes passos:

- avaliação das informações sobre a ferramenta de conversão;
- teste de conversão com aplicações de baixa complexidade. A aplicação escolhida foi jogo da velha, produzido durante a fase de treinamento básico na linguagem SmallTalk, o JVelhaSolo;
- teste de conversão com o framework FraG e com um jogo da velha feito a partir do framework, aqui denominado como JVelhaFraG.

As ferramentas avaliadas foram o Bistrô (BOYD, 2000) e o St2J (OBJECTART, 1997). A figura 5.7 mostra um método simplificado do framework FraG em SmallTalk; a figura 5.8 mostra a conversão deste método feita pelo Bistrô; na figura 5.9, tem-se a conversão feita pelo St2J; a conversão ideal do método para Java aparece na figura 5.10.

```

alocarPeao: linha and: coluna
  (self primeiroJogador informarDaVez) ifTrue: [
    (self posicoes atPoint: (linha @ coluna))
      alocarPeao: self primeiroJogador.
    ^false
  ]
  ifFalse: [^true].

```

Figura 5.7: Método em SmallTalk.

A primeira ferramenta avaliada foi o Bistrô, que foi rejeitada por ter um foco diferente do requerido neste trabalho. O Bistrô na realidade é uma linguagem de programação que é executado sobre a linguagem Java, e através de suas ferramentas é possível transformar um código SmallTalk em um código do Bistrô. O uso do Bistrô serve como camada intermediária para executar o código produzido em SmallTalk na JMV, e a aplicação em Java, além de ter um código de difícil manutenção, será dependente do Bistrô.

A ferramenta St2J fazia conversão de maneira satisfatória, resultando em código não compilável de forma que a estrutura da lógica era mantida, deixando para o usuário a tarefa de resolver as especificidades entre as linguagens, o que será discutido mais adiante. Desta maneira

```

public Object alocarPeao_and(final Object linha,
                             final Object coluna) {
    try {
        if (primitive.booleanFrom(this.primeiroJogador()
                                  .perform("informarDaVez"))){
            (new ZeroArgumentBlock() { public Object value() {
                TesteConv.this.posicoes().perform_with("atPoint:",
                linha.perform_with("@", coluna))
                .perform_with("alocarPeao:",
                TesteConv.this.primeiroJogador());
                throw new MethodExit("TesteConv.alocarPeao_and",
                Object.primitive.literalFalse());
            }}).value();
        } else {
            (new ZeroArgumentBlock() { public Object value() {
                throw new MethodExit("TesteConv.alocarPeao_and",
                Object.primitive.literalTrue());
            }}).value();
        }
        throw new MethodExit("TesteConv.alocarPeao_and", this);
    } catch (MethodExit e) {
        return e.exitOn("TesteConv.alocarPeao_and");
    }
}

```

Figura 5.8: Método convertido com o Bistrô.

```

public <TYPE> alocarPeao_and(<TYPE> linha, <TYPE> coluna) {
    if (this.primeiroJogador().informarDaVez()) {
        this.posicoes().atPoint(linha @ coluna)
        .alocarPeao(this.primeiroJogador());
        return false;
    }
    else
        return true;
    return this;
}

```

Figura 5.9: Método convertido com o St2J.

terminou-se a procura por uma ferramenta de conversão de SmallTalk para Java, escolhendo a ferramenta St2J para esta tarefa.


```

public boolean alocarPeao_and(int linha, int coluna) {
    if (this.primeiroJogador.informarDaVez()) {
        (this.posicoes.atPoint(new Point(linha, coluna))
        .alocarPeao(this.primeiroJogador);
        return false;
    }
    else {
        return true;
    }
}

```

Figura 5.10: Método desejável em Java.

5.4.2 Conversão de SmallTalk para Java

Para a conversão de código SmallTalk em Java usou-se um repositório de soluções a problemas, possibilitando que problemas semelhantes tivessem soluções semelhantes. O processo de conversão consiste na geração automática das classes Java a partir das classes em SmallTalk², analisar cada classe, e para cada erro decorrente da diferença das linguagens implementar a solução correspondente no repositório de soluções, caso não haja uma solução correspondente no repositório esta deve ser criada e adicionada.

Um dos problemas encontrados durante a fase de conserto do código não compilável é que os erros de uma classe impediam a compilação de outra classe, tornando muito difícil distinguir quais os erros eram originados em uma classe. Para contornar este problema, foram criadas classes “interface” para as classes que se deseja compilar³, de modo que esta classe contenha a mesma relação de métodos da classe a corrigir. Este processo é ilustrado na figura 5.11, que consiste em:

1. Criar uma classe “interface” que contenha todos os métodos da classe alvo, métodos sem corpo ou com retorno nulo quando existir retorno. Fazer que a classe criada possua a mesma superclasse da classe alvo, depois modificar a superclasse da classe alvo para a nova classe que foi criada.
2. Repetir o passo anterior para todas as classes que representam uma dependência para a classe alvo, e alterar essas dependências para as classes “interface” criadas.
3. Arrumar os erros de compilação da classe alvo.

²Geração de código Java não compilável através do uso da ferramenta de conversão St2J.

³Uma classe “interface” é identificada pela terminação “Interf” em sua nomenclatura.

4. Remover a classe “interface” criada para a classe alvo e alterar todas as classes que dependem dessa classe “interface” para que dependa da classe alvo. Após isto, é necessário verificar se a modificação na classe alvo acarretou em algum erro nas classes que dependem dela, por exemplo, incompatibilidade de tipos de retorno, e caso haja erros, estes devem ser consertados.

Estes passos devem ser executados até que todas as classes “interface” sejam removidas. Este procedimento possibilita visualizar os erros inerentes a classe que se deseja consertar, e a criação de classes “interface” evita que as classes fiquem com algum vestígio indesejado, como por exemplo, um retorno nulo onde não existe este retorno.

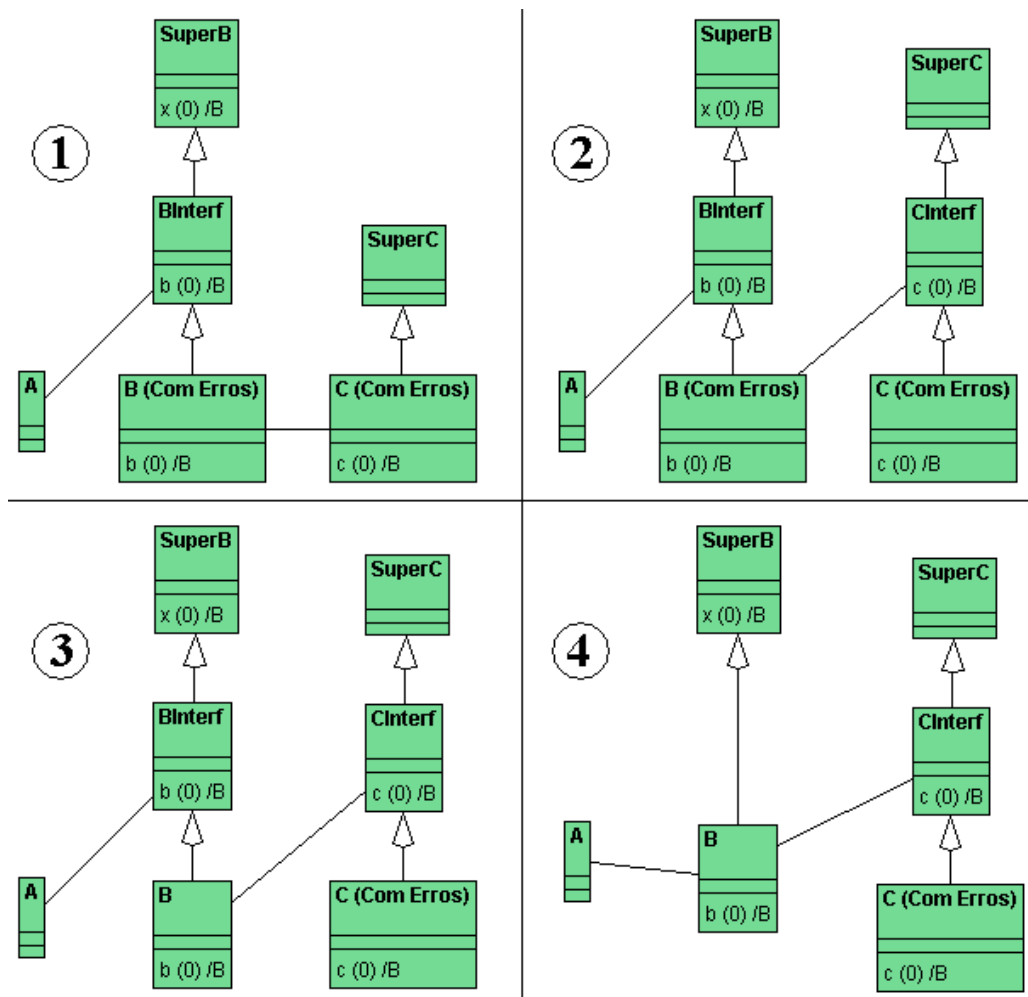


Figura 5.11: Processo para evitar que os erros de uma classe interfiram em outra.

A variação dos problemas a serem resolvidos na fase de conversão manual é alta, e ocorrem com frequência. Não serão descritos todos os problemas da conversão pois isto ficaria deveras extenso e monótono, ao invés disso, serão descritos os tipos de problema e a sua devida solução.

Tipagem de Objetos

SmallTalk não tem tipagem de variáveis e parâmetros, como ocorre em Java. Essa característica gera uma grande liberdade na programação em SmallTalk, mas dificulta a conversão para Java. Um exemplo disso é o uso de uma mesma variável para armazenar um número inteiro e para armazenar uma lista. Quando se encontrou o problema de uma mesma variável que é usada para dois tipos totalmente diferentes, esta foi quebrada em duas de modo que cada uma tenha um tipo específico.

Quando uma variável era usada para armazenar objetos de tipos diferentes, mas que tinham uma superclasse comum, com exceção da classe *Object*⁴, esta superclasse era usada para definir o tipo desta variável. Entretanto o uso de superclasses muito genéricas acarretava, muitas vezes, na necessidade de verificar-se o tipo da variável em tempo de execução, para tratar um tipo em específico.

Equivalência de rotinas nativas

A equivalência de rotinas nativas consiste em procurar recursos em Java que forneçam a mesma funcionalidade dos recursos usados em SmallTalk. Quando existe um recurso semelhante este deve ser modificado para que o comportamento do novo código seja igual ao antigo.

O uso do método *copy*⁵ em SmallTalk pode ser substituído pelo método *clone* em Java, entretanto deve-se sinalizar as classes que podem ser clonadas, implementando a interface *Cloneable*, para evitar que a chamada do método gere a exceção *CloneNotSupportedException*.

Em SmallTalk o uso de "=", para comparar o conteúdo de objetos, deve ser substituído em Java pela invocação do método *equals* passando por parâmetro o objeto que se deseja comparar. Para comparar se dois objetos são o mesmo, é utilizado o sinal "==", assim como em Java. Diferente do SmallTalk, em Java a chamada de um método de uma variável nula causa uma exceção. É importante a atenção neste ponto, pois se variável puder possuir o valor nulo então o código deve tratar esta possibilidade.

O padrão de projeto *Observer* está implementado na classe *Object* do SmallTalk, enquanto que em Java a classe que implementa este padrão é a classe *Observable*. Para resolver este problema de modo que estrutura hierárquica não se altere, as classes que usavam o padrão *Observer* passaram a usar a classe *Observable*, mas não mais por herança e sim por agregação (GERHARDT; WEGE, 1999).

⁴*Object* é superclasse de todas as classes no Java.

⁵Método da classe *Object*, que no SmallTalk também é a superclasse global.

Diferenças semânticas

As diferenças semânticas referem-se a diferenças entre as linguagens SmallTalk e Java, que necessariamente implicam em mudanças na implementação em Java.

SmallTalk tem a definição de métodos de classe e que possuem polimorfismo, em Java existe a definição de métodos estáticos em uma classe mas que não possuem polimorfismo com a sua estrutura hierárquica (GERHARDT; WEGE, 1999). A conversão automática transforma métodos de classe em métodos estáticos. Para manter o polimorfismo, foram transformados em métodos de instância todos os métodos estáticos que não geravam algum efeito colateral⁶. Nos métodos em que não foi possível empregar esta solução, optou-se por manter o método como estático, porém modificando-o para receber parâmetros que possibilitem o processamento genérico.

A manipulação da entidade classe⁷ é diferenciada nas duas linguagens. Por exemplo, para a criação de um objeto a partir de uma classe que foi passada por parâmetro: em SmallTalk simplesmente chama-se o método de classe que cria a instância, enquanto em Java é necessário usar uma API específica para isto⁸.

Em SmallTalk todos os tipos usados são objetos, enquanto em Java alguns tipos não são objetos, são os chamados *tipos primitivos*⁹ (GERHARDT; WEGE, 1999). Conseqüentemente, o uso de tipos primitivos na linguagem Java para substituir algum tipo em SmallTalk é desfavorecido, pois é necessário adotar um valor que represente o valor nulo para a variável¹⁰ e trabalhar com objetos invólucros para adicionar os tipos primitivos em coleções¹¹, uma vez que coleções aceitam somente objetos.

Em Java o índice de coleções normalmente começa no zero, enquanto em SmallTalk começa no 1. Para resolver este problema na implementação em Java tem-se como alternativas modificar o algoritmo para se adaptar as coleções, o que pode gerar efeitos colaterais, ou criar-se uma estrutura que comece em 1, como será visto mais adiante.

⁶Um exemplo de método que foi transformado em método de instância foi um método que retorna um identificador de um tipo de elemento de especificação.

⁷Refere-se a classe de maneira genérica, como entidade tratada pela linguagem que suporta o paradigma OO e não ao conceito classe presente no diagrama de classe.

⁸Neste caso é necessário usar a API de reflexão.

⁹Exemplos de tipos primitivos em Java: inteiro, real e caractere.

¹⁰Em Java, variáveis declaradas com tipos primitivos não tem valor nulo, pois tipos primitivos não representam objetos e sim valores simples. Por exemplo, ao se declarar uma variável do tipo inteiro, automaticamente ela possui o valor zero. Poderia-se usar o objeto invólucro para o tipo inteiro (instância da classe *Integer*), mas optou-se pelo uso do tipo primitivo *int* pois o seu uso é simplificado.

¹¹A versão de 1.5 de Java prevê a transformação automática de um tipo primitivo em seu objeto invólucro correspondente. Atualmente o OCEAN é executado com a versão 1.5, mas inicialmente era usada a versão 1.4, que não previa estas facilidades.

5.4.3 Validação do código gerado

A validação do código em Java baseia-se no fato que a aplicação em SmallTalk está funcionando corretamente, desta forma constata-se que os erros do código em Java foram introduzidos na fase de conversão. Para validar o código Java gerado foram usadas as seguintes regras para testar cada funcionalidade:

1. Gerar a relação de resultados esperados da funcionalidade que irá ser testada. Esta relação deve constar o estado das variáveis dos objetos envolvidos antes e depois de executada a funcionalidade.
2. Executar o código em Java, e se resultar em erros, estes devem ser consertados e este passo de ser executado novamente.
3. Comparar os resultados obtidos com os encontrados. Se forem diferentes as duas implementações, em SmallTalk e em Java, ambas são executadas paralelamente em modo de depuração e as partes que geram inconsistências devem ser consertadas. Este passo de ser executado novamente enquanto as inconsistências persistirem.
4. Validar novamente as funcionalidades que foram afetadas com as mudanças da funcionalidade testada.

Para a validação da reengenharia do framework OCEAN foi usado o ambiente SEA, criando uma especificação OO e todos os seus elementos¹². Decidiu-se converter a especificação OO, implementada na versão do SEA em SmallTalk, para que fosse possível comparar a totalidade da execução em Java com a execução em SmallTalk, possibilitando a validação de todo o código convertido de SmallTalk para Java, que inclui o framework OCEAN, o ambiente SEA e toda a estrutura da especificação OO.

5.4.4 Experimentos

O primeiro experimento foi feito com uma aplicação do jogo da velha, o JVelhaSolo¹³, e visa avaliar as características do código gerado pela ferramenta em conjunto da conversão manual. O segundo experimento foi feito com o framework FraG e com uma aplicação de jogo da velha feito a partir do framework. Este experimento visa avaliar o processo de conversão de um framework e suas aplicações, possibilitando uma comparação em menor escala com o

¹²O ambiente SEA e toda a estrutura da especificação OO também foram convertidos para Java.

¹³Aplicação do jogo da velha criado como resultado dos treinamentos básicos em SmallTalk.

framework OCEAN e o ambiente SEA. Os dois experimentos foram validados com sucesso, e possibilitaram a geração e a avaliação de todo o processo de conversão.

5.5 Resultados

O processo descrito para conversão de SmallTalk para Java possibilitou a quebra da complexidade em diversas partes, de modo que a conversão total se deu com a validação iterativa dessas partes específicas. Estas validações iterativas possibilitaram a geração de resultados gradativos, a seguir descritos.

5.5.1 Execução primitiva do ambiente

A execução primitiva do ambiente SEA é caracterizada por não possuir uma interface, onde os testes eram feitos através da adição de código que invoca diretamente os métodos para a validação das funcionalidades.

Duas funcionalidades foram testadas nesta fase, a funcionalidade de criar um ambiente e a de criar uma especificação OO sem modelos. A validação de ambas as funcionalidades neste momento tiveram a intenção de testar o protocolo e os algoritmos envolvidos nessas funcionalidades, servindo de base para testes mais aprimorados.

5.5.2 Suporte à criação de diagrama de classes texto

Esta fase tinha como objetivo a criação de diagrama de classe que possibilitasse manipular vários conceitos. Esta funcionalidade contemplava muitas combinações de teste, tornando complexa esta validação, caso os testes fossem feitos como na execução primitiva do ambiente. A solução natural foi o desenvolvimento de um ambiente, o DCSEA, que foi implementado estendendo o SEA para que alterações no novo ambiente não prejudicassem o SEA, e que contivesse uma interface gráfica, ainda que rudimentar.

O ambiente DCSEA suportava a criação de uma especificação OO de um diagrama de classe somente, fazendo essas criações de forma automática. Entretanto era possível criar e remover classes, relacionamentos de herança, composição e associação, métodos e atributos associados a uma classe, e parâmetros associados a um método. A visualização do diagrama de classe era feita de forma descritiva em uma interface texto, e a criação, remoção e edição dos conceitos era feita através de botões específicos. A figura 5.12 mostra o ambiente DCSEA.

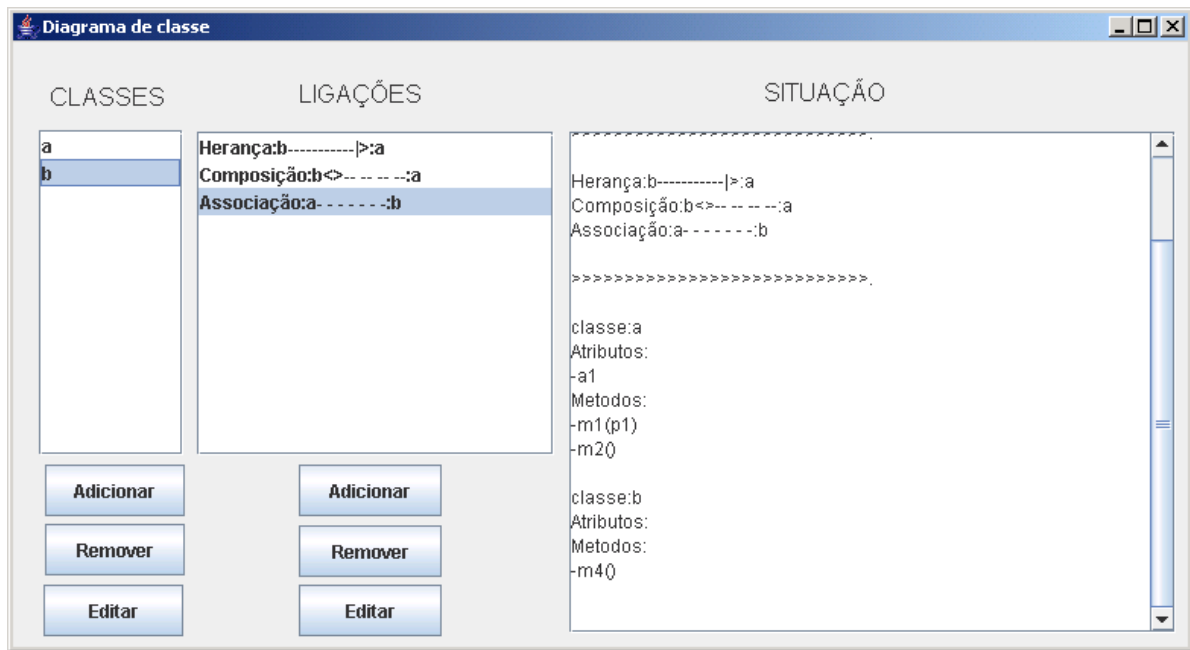


Figura 5.12: Ambiente DCSEA para validação de diagrama de classe.

Apesar deste ambiente possuir uma interface não intuitiva e limitar a quantidade e a variedade de diagramas, este foi de fundamental importância para depurar erros de conversão pois possibilitou trabalhar simultaneamente com sete tipos de conceitos diferentes¹⁴, proporcionando a validação de muitas funcionalidades.

5.5.3 Protótipo do ambiente SEA

Após a validação da criação do diagrama de classe e seus conceitos, a interface gráfica com o usuário do ambiente SEA estava pronta para se integrar a sua parte conceitual. Conseqüentemente a próxima iteração no processo de validação consistiu em realizar esta integração, que foi realizada em conjunto com a outra vertente de conversão para Java, a vertente de interface gráfica. O presente trabalho se ateve na depuração do framework, possibilitando que o trabalho responsável pela interface gráfica com o usuário do ambiente SEA efetuasse a sua validação (MACHADO, 2007).

Também de maneira paralela estava sendo desenvolvido o suporte a editores gráficos de modelos usando o JHotDraw¹⁵ a ser integrado no ambiente SEA (AMORIM, 2006). A interface proposta para o ambiente SEA possibilitava a visualização de elementos de especificação de

¹⁴Classe, herança, composição, associação, atributo, método e parâmetro.

¹⁵O suporte a editores gráficos usando o JHotDraw foi desenvolvido por João Amorim como trabalho de conclusão de curso.

maneira genérica, suportando os editores gráficos projetados, entretanto faltava ao framework OCEAN a adaptação ao framework de editores gráficos, que em SmallTalk era o HotDraw e em Java o JHotDraw. Esta adaptação foi feita de maneira semelhante ao processo de integração da interface gráfica do SEA, de forma que esse trabalho se ateu as alterações no framework OCEAN, sem se ater a aspectos de visualização.

O ambiente resultante tinha a capacidade de criar diversos modelos, entretanto somente o diagrama de classes tinha um modo de visualização¹⁶. Para suprir a deficiência de não poder editar os outros diagramas e para validar a manipulação de conceitos de outros modelos, foram projetados editores com elementos textuais¹⁷, para os diagramas de caso de uso, de seqüência e para os diagramas de corpo de método. O trabalho paralelo de criação de editores gráficos possibilitou geração de editores gráficos para os digramas de caso de uso, de atividades, de estados e de seqüência, além do diagrama de classes (AMORIM, 2006). O protótipo do ambiente SEA em Java e os diagramas da especificação OO podem ser evidenciados nas figuras 5.13 e 5.14.

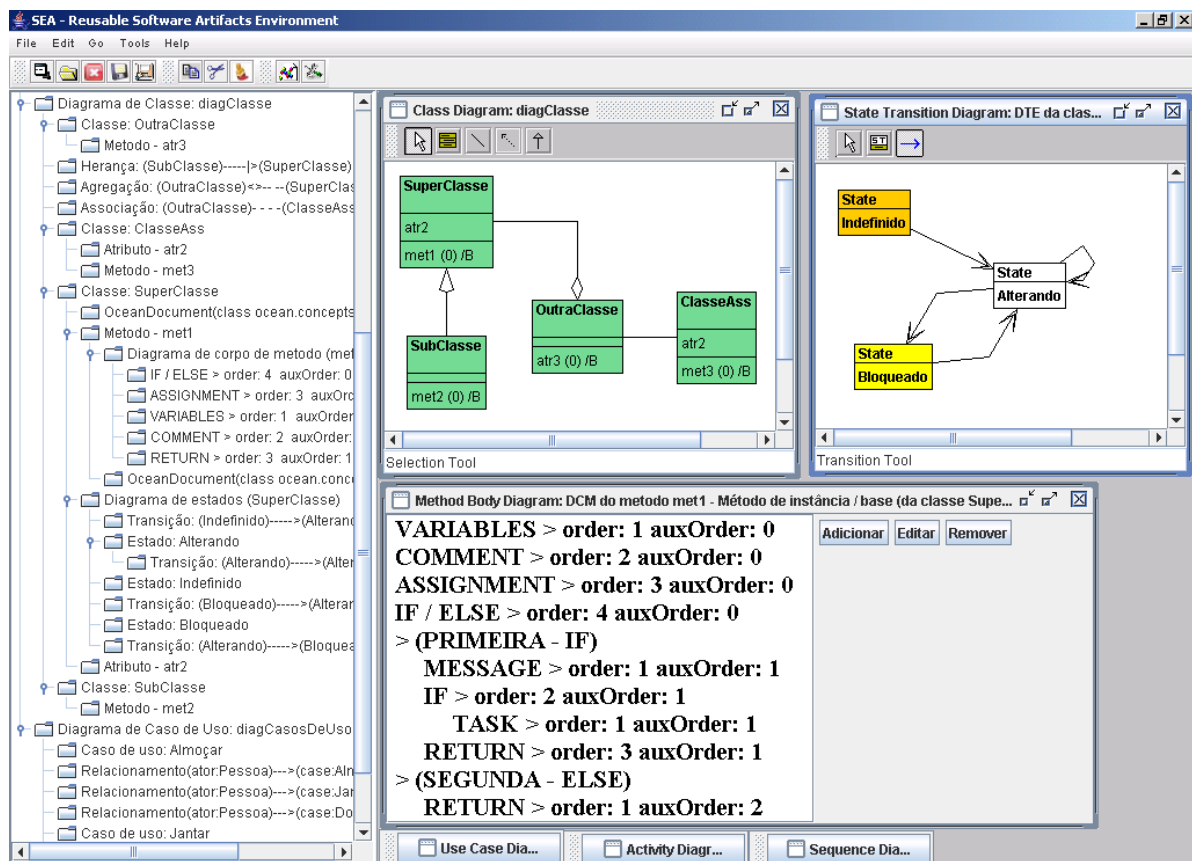


Figura 5.13: Diagrama de classe, de estado e de método no ambiente SEA em Java.

¹⁶Neste momento o modo de visualização do diagrama de classe já era com elementos gráficos, não mais em com elementos textuais.

¹⁷Semelhante ao diagrama de classe em modo texto citado na subseção anterior.

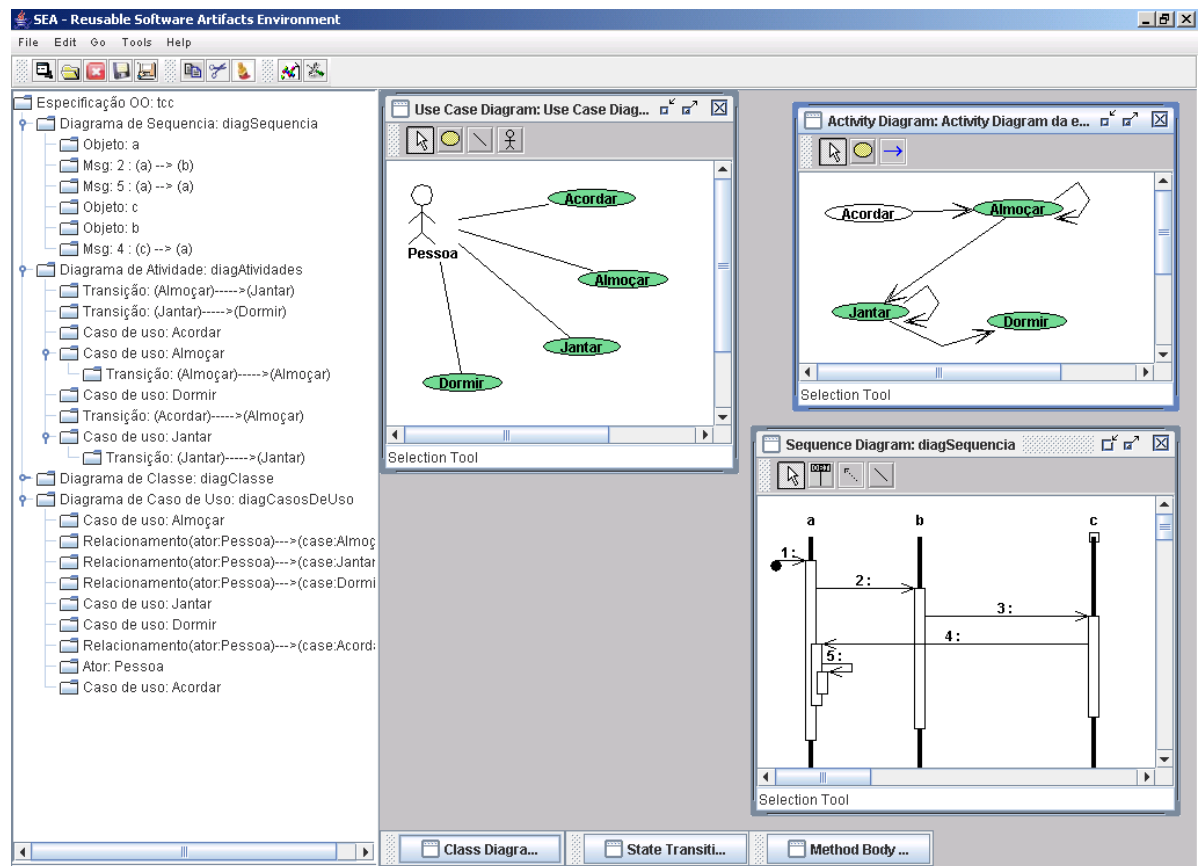


Figura 5.14: Diagrama de caso de uso, de atividade e de seqüência no ambiente SEA em Java.

Após a validação das funcionalidades de manipulação de conceitos dos diagramas de uma especificação OO, foi iniciada a validação da conversão das ferramentas embutidas no ambiente SEA, possibilitando também a validação da ferramenta de análise de diagrama de classe.

5.6 Adaptações

Para a reimplementação do framework OCEAN em Java projetou-se soluções para determinados tipos de problemas, entretanto algumas soluções necessitaram de adaptações perante o framework OCEAN.

5.6.1 Uso de coleções no OCEAN

O índice de coleções em SmallTalk começa em 1 enquanto em Java começa em 0. Para resolver este problema visualizou-se duas vertentes: ou alterava-se todos algoritmos que fazem uso de índices de uma coleção ou criava-se uma coleção que começasse no 1.

A primeira solução, a de alterar todos algoritmos que fazem uso de índice de coleções, tinha como vantagem o uso do padrão de Java, possibilitando o reuso de estruturas de coleção previstas na linguagem. A desvantagem é que essa alteração poderia gerar muitos erros, uma vez que a quantidade de código a ser alterada seria grande, podendo gerar um comportamento diferenciado do esperado, tornando a validação do framework mais custosa.

A segunda solução, de criar e usar uma coleção que começasse em 1, tem a vantagem de diminuir o código alterado durante a fase de conversão manual, diminuindo a possibilidade de erros. A desvantagem seria a adoção de uma coleção que difere do padrão de Java, dificultando a manutenção do código que usa essa estrutura.

A solução adotada é uma mescla das duas soluções, inicialmente usa-se a coleção que começa em 1 e a medida que o framework for validado altera-se para usar coleções nativas de Java. A classe que corresponde à coleção criada foi denominada de *OceanVector*, que agrega um *Vector*¹⁸ e que possui todos os métodos da classe *Vector*, sendo a classe *OceanVector* um invólucro à classe *Vector*, pois usa uma coleção que começa em 0 mas se comporta como se começasse em 1.

5.6.2 Alteração da estrutura

SmallTalk não possui tipagem de objetos, um variável pode receber qualquer objeto, fazendo que o polimorfismo do SmallTalk seja global, enquanto o polimorfismo de Java é restrito ao tipo. Em SmallTalk, a invocação de um método pode ser efetuada para qualquer objeto que possua este método e se o objeto não possuir o método haverá um erro de execução. Em Java, pode-se apenas invocar métodos presentes na classe, ou em alguma superclasse, que define o tipo da variável, caso contrário haverá um erro de compilação.

Quando se converte um código para Java e depara-se com este tipo de problema, uma solução é fazer a verificação do tipo do objeto em tempo de execução, e quando for chamar o método, forçar o tipo da variável para o tipo verificado. Entretanto a melhor solução é usar como tipo uma classe comum na hierarquia de classes dos objetos previstos, de modo que esta classe possua o método invocado naquele trecho de código. Caso não seja possível encontrar uma classe comum que resolva o problema, uma solução é a adoção de interfaces Java, outra é a alteração da estrutura de classes. Para essa necessidade de alteração no framework OCEAN, decidiu-se alterar a estrutura de classes do framework para que este suporte tais necessidades sem o auxílio de recursos específicos de linguagem, como as interfaces Java.

¹⁸Uma das estruturas de coleção previstas no Java.

Um problema constatado na estrutura do framework OCEAN estava relacionado aos conceitos sustentados pelo conceito de classe, conceitos de atributo e método, e relacionado aos conceitos sustentados pelo conceito de método, conceitos de parâmetro e de variável. Percebeu-se que os conceitos de atributo e método possuíam métodos em comum, entretanto não possuíam uma superclasse comum que definia estes métodos. Criou-se a classe abstrata *ClassPart*¹⁹ como subclasse de *TypedObject*²⁰ contendo os métodos comuns entre os conceitos de atributo e método, e estes passaram a ser subclasse da classe criada. Um efeito colateral desta mudança é que o conceito de método passou a conter a classe *TypedObject* na sua hierarquia, que antes não possuía. Verificou-se que esta alteração tinha lógica, uma vez que um método pode possuir o tipo de retorno, entretanto percebeu-se uma incoerência conceitual pois um método não é um objeto, e para resolver isto a classe *TypedObject* foi renomeada para *TypedElement*²¹. A modificação dos conceitos de parâmetro e de variável de método seguiu a mesma linha de raciocínio da alteração dos conceitos de atributo e método, onde foi criada a classe *MethodPart*²². Entretanto a modificação dos conceitos de parâmetro e de variável de método não causou nenhum efeito colateral uma vez que ambos os conceitos possuíam a mesma hierarquia de super classes. A figura 5.15 mostra a estrutura de classe dos conceitos de método, atributo, parâmetro de método e variável de método implementada em SmallTalk. A figura 5.16 mostra essa mesma estrutura implementada em Java.

Muitos métodos do framework OCEAN em SmallTalk recebiam objetos da classe *OceanDocument*²³ e da classe *SpecificationElementHolder*²⁴, porém a única classe comum entre as duas era a classe *Object*. A figura 5.17 mostra o metamodelo do framework OCEAN implementado em SmallTalk. Ao converter para Java, esses métodos teriam que ser tipados com a classe *Object*, entretanto não poderiam receber como parâmetro objetos que não fossem da classe *OceanDocument* ou da classe *SpecificationElementHolder*.

A solução foi alterar o metamodelo do framework OCEAN quebrando a classe *OceanDocument* em duas. Para isso foi criada a classe *SpecificationDocument*²⁵ estendendo a classe *OceanDocument*, e todas as classes que estendiam *OceanDocument* foram alteradas para que passassem a estender a classe *SpecificationDocument*. Esta classe contém todos os métodos da classe *OceanDocument* que não são comuns à classe *SpecificationElementHolder*, e a classe *OceanDocument* passou a conter somente os métodos que são comuns a classe *SpecificationE-*

¹⁹A classe *ClassPart* define conceitos que são parte de uma classe.

²⁰A classe *TypedObject* representa um objeto com tipo.

²¹A classe *TypedElement* representa um elemento de especificação com tipo.

²²A classe *MethodPart* define conceitos que são parte de um método.

²³Define documentos em geral no framework OCEAN.

²⁴Classe que tem a função de manter todos os elementos de especificação de um determinado tipo.

²⁵A classe *SpecificationDocument* define documentos vinculados à especificação.

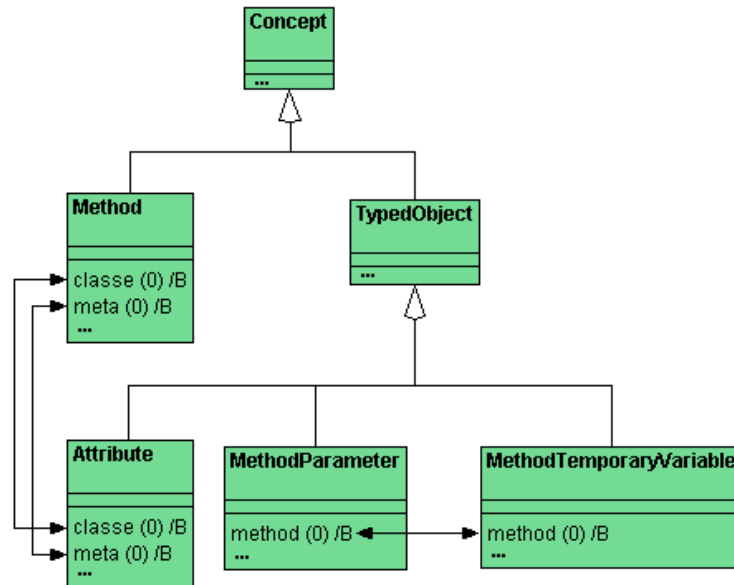


Figura 5.15: Estrutura de classe dos conceitos de método, atributo, parâmetro de método e variável de método em SmallTalk.

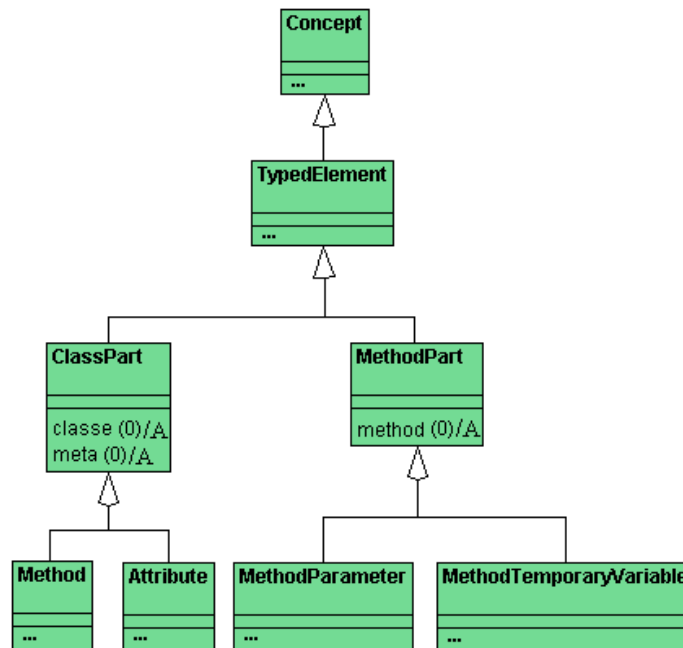


Figura 5.16: Estrutura de classe dos conceitos de método, atributo, parâmetro de método e variável de método em Java.

lementHolder, e esta foi alterada para estender a classe *OceanDocument*. A figura 5.17 mostra o metamodelo do framework OCEAN implementado em SmallTalk e a figura 5.18 mostra o metamodelo implementado em Java.

Desta forma é possível usar a classe *SpecificationDocument* para definir um tipo que se

restringe a especificações e elementos de especificações, e a classe para *OceanDocument* para restringir a documentos internos do framework OCEAN.

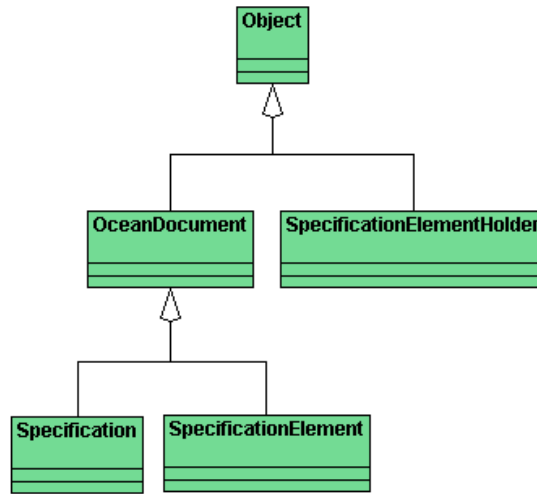


Figura 5.17: Estrutura de classes do metamodelo do framework OCEAN em SmallTalk.

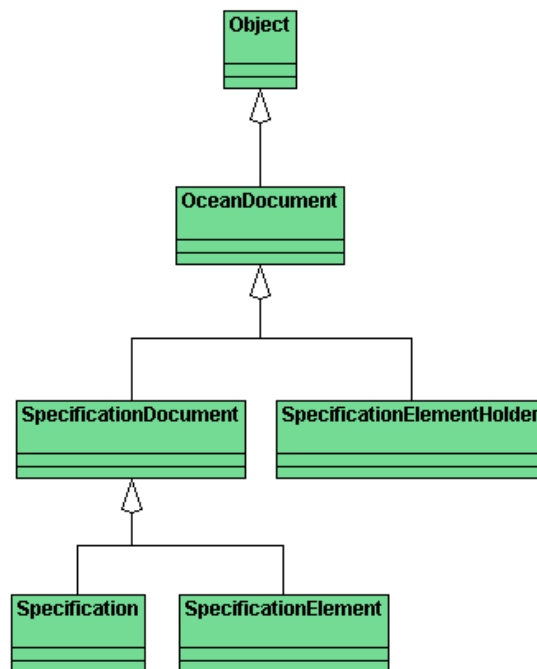


Figura 5.18: Estrutura de classes do metamodelo do framework OCEAN em Java.

5.7 Melhorias

A reengenharia do framework OCEAN prevê como objetivo principal uma reimplementação do framework em Java, entretanto somente as adaptações não bastam para que essa reimple-

mentação atinja seu objetivo de facilitar a sua manutenção. Aqui serão discutidas as melhorias que fomentam a versão em Java do framework OCEAN.

5.7.1 Uso do padrão de projeto *Observer*

A utilização do padrão de projeto *Observer* no framework OCEAN em SmallTalk era usado estritamente para propagação das notificações de atualização, através do método *redraw*. Essa propagação era feita percorrendo a lista dos observadores e chamando diretamente o método *redraw*. Ao converter para Java observou-se que não era possível percorrer a lista de observadores do objeto da classe *Observable*, e muito menos chamar diretamente o método. Além disso a forma que foi implementado o padrão *Observer* no SmallTalk restringia o uso do padrão, não permitindo o seu uso para outro fim. Optou-se em usar o padrão *Observer* de maneira genérica, possibilitando a propagação da notificação de atualização ou qualquer outro tipo de notificação. Onde era percorrida a lista de observadores para chamar o método *redraw* foi substituído pela chamada do método, passando como parâmetro um identificador, que gera a notificação para os observadores. O método das classes dos objetos que podem receber uma notificação só invoca o método *redraw* se o identificador passado for igual ao identificador da notificação de atualização.

5.7.2 Suporte a criação de elementos gráficos

No framework OCEAN é possível através de um editor gráfico de um modelo criar os conceitos daquele modelo. Porém não existe um mecanismo que partindo do modelo e seus conceitos consiga recriar a parte gráfica, onde a solução adotada para o armazenamento de especificações foi manter os elementos gráficos como elementos persistentes de uma especificação. Para a implementação em Java percebe-se que o JHotDraw não possuía um suporte satisfatório ao armazenamento de elementos gráficos²⁶ inviabilizando que a mesma solução da implementação em SmallTalk fosse adotada.

Decidiu-se em criar esse mecanismo de criação da parte gráfica a partir do modelo e dos conceitos na implementação em Java. Para esta solução alterou-se a entidade que representa um conceito, classe *Concept*, adicionando um método para ser estendido nas suas subclasses, que informa se o elemento gráfico correspondente ao conceito precisa de um ponto para ser redesenhado²⁷. No caso deste método não ser sobrescrito, o retorno é falso. Na entidade que

²⁶Por exemplo, um dos problemas de armazenamento do JHotDraw é que seus elementos usam estruturas que não podem ser serializadas, inviabilizando o armazenamento em arquivo por serialização Java.

²⁷Um exemplo de conceito que precisa de um ponto para ser reconstruído é o conceito de classe do diagrama

representa um modelo, classe *ConceptualModel*, foi adicionada uma estrutura de dados para mapear um conceito²⁸ para o ponto de localização da figura correspondente ao conceito no editor daquele modelo. Foram adicionados métodos na entidade de modelo que atualizam os pontos mapeados e que criam os elementos gráficos. O método que cria os elementos gráficos repassa essa função para a camada responsável pela intermediação com o framework de editores gráficos, o JHotDraw, passando como parâmetros os seus conceitos e o mapeamento de pontos desses conceitos. Nessa camada foi adicionado um protocolo genérico de criação de figuras, onde é necessário apenas completar-se as especificidades²⁹, entretanto é possível criar um protocolo específico³⁰. Na entidade de especificação também foram adicionados métodos para atualizar os pontos e para criar os elementos gráficos de seus modelos, onde simplesmente é repassada a tarefa para todos os modelos criados.

5.7.3 Suporte ao armazenamento de especificações

O suporte à criação de desenhos possibilitou alterar o protocolo de armazenamento para que a sua lógica se preocupasse apenas com o tratamento de especificações. Antes de salvar uma especificação, são atualizados os pontos dos conceitos de todos os modelos, e os elementos gráficos são descartados. Após o carregamento de uma especificação, os elementos gráficos antes descartados são recriados. Como as especificações armazenadas não contêm elementos gráficos, pode-se remover o framework JHotDraw para o uso de um outro artefato de software para edição de elementos visuais, de modo que, as especificações antigas poderão ser carregadas normalmente. Outra característica do mecanismo de armazenamento, não presente na versão em SmallTalk, é que além da opção *salvar* existe a possibilidade *salvar como*, onde são necessárias informações do usuário para salvar a especificação.

O framework OCEAN tem uma entidade responsável pelo armazenamento, a classe *StoreManager*, e no SmallTalk a forma de uso desta classe permite apenas um tipo de armazenamento, e a criação de outra forma de armazenamento implica na exclusão de uso das outras³¹.

de classe, ou um conceito de objeto do diagrama de seqüência, em que a posição do elemento gráfico depende do ponto do editor onde a ferramenta de criação foi clicada. Um exemplo de conceito que não precisaria de um ponto é o conceito de herança, onde o posicionamento do seu elemento gráfico depende da localização dos elementos gráficos das classes que o conceito de herança relaciona. Outro conceito que não necessita de pontos é conceito de mensagem, a posição do seu elemento gráfico depende da posição do elemento gráfico do objeto que contém a mensagem, da ordem da mensagem e de informações das mensagens com ordem menor.

²⁸Apenas conceitos que necessitam de um ponto são mapeados.

²⁹Um tipo de especificidade é dizer qual figura será criada para representar determinado tipo de conceito.

³⁰O diagrama de seqüência possui um protocolo específico, porque o protocolo genérico não satisfaz corretamente as restrições deste diagrama.

³¹O armazenamento de especificações implementado em SmallTalk usa a serialização de objetos prevista na linguagem para armazenamento em arquivo. Por exemplo, se for criado um mecanismo de armazenamento em banco de dados, não será possível carregar uma especificação armazenada em arquivo e salvá-la em banco de

Na implementação em Java criou-se a classe *SeaStoreManager*, que estende *StoreManager*, e agrega outras instâncias de *StoreManager*. A classe *SeaStoreManager* foi implementada usando os padrões de projeto *composite* e *decorator*, e desta forma possibilita que seja adicionada a funcionalidade de poder escolher diversos tipos de *StoreManager*, ou seja, diversos tipos de armazenamento. A classe criada não trata especificações para um determinado tipo de armazenamento, ela delega as funções de armazenamento para outros objetos cuja classe estende a classe *StoreManager*, possibilitando, por exemplo, que o usuário escolha carregar uma especificação de um arquivo e salvá-la em um banco de dados. A figura 5.19 mostra a classe *SeaStoreManager* envolvida na estrutura de classes responsáveis pelo armazenamento de especificações no framework OCEAN em Java.

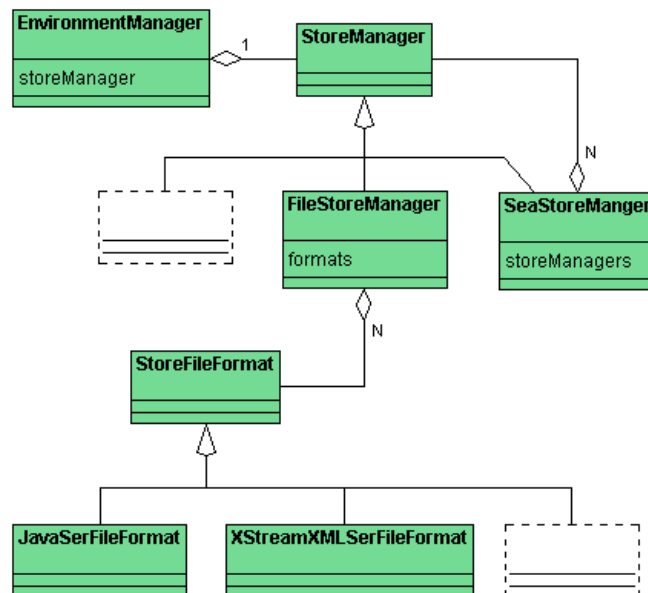


Figura 5.19: Estrutura de classes que definem o novo mecanismo de armazenamento.

Para o armazenamento propriamente dito, uma subclasse de *StoreManager* foi projetada para o armazenamento em arquivo usando a serialização de objetos do Java e outra subclasse para o armazenamento em arquivo XML, usando a serialização da biblioteca *XStream*³². Entretanto percebeu-se que as duas implementações possuíam características em comum, por exemplo, o algoritmo para perguntar onde o arquivo seria salvo. A análise destas características resultou na criação da classe *FileStoreManager*, que é responsável em armazenar especificações em arquivo de forma genérica, e que agrega uma coleção de objetos da classe *StoreFileFormat*, que generaliza as características do formato para armazenamento em arquivo. O suporte a um determinado formato é feito através da criação de uma subclasse de *StoreFileFormat* onde são

dados, ou vice-versa. Só é possível carregar e salvar especificações armazenadas em um mesmo formato.

³²Disponível em <http://xstream.codehaus.org>.

definidas as especificidades do formato desejado³³. A versão em Java do framework OCEAN prevê o suporte ao armazenamento em arquivos com a extensão “.ser”, usando a serialização de objetos Java implementado na classe *JavaSerFileFormat*, e o suporte ao armazenamento de arquivos com a extensão “.xml”, usando a serialização de objetos da biblioteca *XStream* implementado na classe *XStreamXMLSerFileFormat*. Vale ressaltar que, se a classe *EnvironmentManager* usar como gerente de armazenamento um objeto da classe *FileStoreManager*, só será permitido carregar e salvar uma especificação armazenada nos formatos de arquivo suportados pela *FileStoreManager*. A figura 5.19 mostra o suporte ao armazenamento de especificações em arquivo e os formatos criados no framework OCEAN em Java.

Esta estrutura de classe possibilita duas formas de reuso na extensão para suporte ao armazenamento de especificações: definindo um novo tipo de armazenamento, criando uma sub-classe de *StoreManager*, ou criando um novo tipo de formato de arquivo, definindo uma sub-classe de *StoreFileFormat*. A figura 5.19 ilustra estas possibilidades nas classes tracejadas.

³³As especificidades previstas na classe que define um formato são: o algoritmo para salvar e carregar um objeto daquele formato, e a extensão e a descrição do formato em que o arquivo será salvo.

6 *Conclusão*

Cada vez mais os softwares tendem a ser mais complexos e o seu ciclo de desenvolvimento mais rápido. Para isto é imprescindível o reuso de software.

6.1 **Realizações**

O framework OCEAN possibilita a aplicação de abordagens voltadas ao reuso manuseando especificações de projeto. O trabalho desenvolvido teve como objetivo a reengenharia do framework OCEAN, mas compreendeu também a conversão do ambiente SEA e de toda a estrutura da especificação OO de SmallTalk para Java, sendo necessário para isto:

- Conhecer a fundo o framework OCEAN;
- Projetar um processo de conversão;
- Usar uma ferramenta de conversão de código SmallTalk para código Java, ainda que gerando código não compilável;
- Planejar a tipagem, uma vez que Java prevê o uso de tipos para variáveis e SmallTalk não;
- Encontrar rotinas em Java que são equivalentes às usadas em SmallTalk;
- Modificar os métodos estáticos em Java, para suprir necessidades atendidas com o uso de métodos de classe em SmallTalk;
- Usar uma biblioteca em Java para criar objetos e acessar métodos a partir de uma classe genérica, em prol da diferença entre as linguagens de Java e SmallTalk neste aspecto;
- Tratar os tipos primitivos em Java, já que em SmallTalk tudo é objeto;
- Prover o suporte para a migração gradativa de coleções para o padrão Java;

- Modificar a estrutura do framework, melhorando o uso dos padrões de projeto existentes e incorporando novas abstrações;
- Prover o suporte à criação de elementos gráficos a partir de seus modelos;
- Aprimorar o mecanismo para armazenamento de especificações.

6.2 Avaliação

Com a reengenharia realizada obteve-se uma versão em Java do framework OCEAN, possibilitando a sua independência de plataforma e de ambiente de desenvolvimento. Além disso, possibilitou-se o amadurecimento da estrutura do framework OCEAN, um melhor tratamento ao uso de elementos gráficos do framework JHotDraw e o intercâmbio para especificações armazenadas em diferentes formatos.

A reengenharia de um framework é uma tarefa bastante complexa, que despence um tempo satisfatório de desenvolvimento e que pode refletir em suas limitações. Com o uso da atual versão do framework OCEAN através do ambiente SEA é possível criar totalmente uma especificação OO, entretanto não é possível verificar a sua consistência, nem gerar código para alguma linguagem de programação, além de não ser possível exportar as informações geradas para um meio independente do framework, como a impressão de modelos por exemplo. Isto acarreta na geração manual de código para alguma linguagem de programação e na dependência do framework para esta tarefa.

Outra limitação é que somente a especificação OO foi validada, limitando por exemplo, o uso de outras especificações suportadas pelo ambiente SEA. Além disso, o mecanismo de propagação de notificação de atualização não classifica o tipo de alteração ocorrida, desta forma, um elemento de especificação notifica os outros elementos que está associado, entretanto pode acontecer que alguns desses elementos não precisem se atualizar, dependendo da modificação. A estrutura do framework OCEAN continua mantendo um relacionamento de alto acoplamento com o framework de editores gráficos.

6.3 Trabalhos futuros

A versão em SmallTalk do framework OCEAN foi um resultado parcial, em relação ao conjunto de requisitos estabelecidos para um ambiente de apoio ao desenvolvimento e uso de frameworks e componentes (SILVA, 2000). A reengenharia do framework OCEAN abrangeu

grande parte dos recursos previstos na versão em SmallTalk no que tange a conversão de código para Java, porém a sua validação cobriu apenas parte destes recursos convertidos. Assim, a primeira prioridade em termos de trabalhos futuros é a continuação do desenvolvimento ora iniciado neste trabalho, seguido da evolução do ambiente SEA, visando a satisfação do conjunto de requisitos estabelecido. Isto implica na evolução dos recursos funcionais e de modelagem do framework OCEAN em Java a serem utilizados pelo ambiente. Abaixo são citados os trabalhos futuros relacionados com o seu escopo:

Validação: Envolve recursos não testados neste trabalho, tais como: criação de diferentes especificações, modelos e conceitos, além do uso das ferramentas e funcionalidades de apoio à edição¹ implementadas na versão em SmallTalk.

Reestruturação: Consiste na melhoria da estrutura do framework OCEAN. O uso do desenvolvimento em camadas possibilitaria a geração de um framework caixa branca a partir do núcleo atual do framework, de modo que este framework caixa branca não teria dependência com elementos externos, eximindo-se de qualquer acoplamento. Poderia haver uma análise de uso dos métodos que possibilitasse uma simplificação na sua quantidade, gerando uma junção de métodos semelhantes de forma que as especificidades sejam descritas pela adição de parâmetros. O código pode ser otimizado e a estrutura de coleção adotada, classe *OceanVector*, poderia ser removida. Além do uso de um sistema de exceção, possibilitando resposta imediata sem que o framework possuísse interação direta com o usuário, onde camadas mais externas poderiam decidir o que fazer com a exceção².

Documentação: Consiste na criação da documentação de projeto e código. A criação de uma ferramenta de engenharia reversa auxiliaria a criação da documentação de projeto do framework OCEAN no próprio ambiente SEA. A documentação do código consiste em criar uma explicação do que faz cada parte de um algoritmo, além da documentação sobre as classes e os métodos, possibilitando a geração da documentação no padrão Java, os *javadocs*.

Melhorias: Um recurso desejável no framework OCEAN é o suporte para geração de relatórios externos, descrevendo métricas e exportando figuras de seus modelos, sendo a impressão um recurso adicional. Outras melhorias seriam em relação às funcionalidades de apoio à edição, onde poderiam ser aprimoradas as existentes e adicionadas outras, como por exemplo, as funcionalidades de desfazer e refazer.

¹Ex: Copiar, colar e recortar.

²Ex: Mostrar no console, em um diálogo ou repassar a exceção para quem requisitou o processamento.

Atualização: Consiste na criação de uma especificação que use técnicas atuais de modelagem, como a UML versão 2.1.1. Além de fornecer suporte ao armazenamento de especificações em formatos atuais, como o XMI versão 2.1.

6.4 Considerações finais

A motivação desta reengenharia é a melhoria do framework OCEAN, tendo como obstáculos dependências que atravancam a sua atualização e a linguagem de programação em que foi implementado, o SmallTalk, que atualmente não é usual.

O trabalho realizado dispõe de um protótipo em Java do framework OCEAN, que apesar de inacabado é melhor estruturado que a versão original e usa uma linguagem de programação mais atual e difundida, e além disto, dispõe também de um protótipo em Java do ambiente SEA e de toda especificação OO, usados para a validação do framework.

Considera-se que o seguinte trabalho seja fundamental para outros trabalhos que venham a realizar manutenções ou melhorias no framework OCEAN, uma vez que possibilitou que as alterações sejam feitas de maneira menos árdua. Ajudando também, na difusão de abordagens voltadas ao reuso, podendo contribuir significativamente ao aumento da produtividade e da qualidade no desenvolvimento de software.

Referências Bibliográficas

- AMORIM, J. de J. *Integração dos frameworks JHotDraw e OCEAN para a produção de objetos visuais a partir do framework OCEAN*. 2006. Trabalho de conclusão de curso – Universidade Federal de Santa Catarina.
- BOYD, N. *Smalltalk over Java: An Introduction to Bistro*. 2000. Disponível em: <http://bistro.sourceforge.net/> Acesso em: 23 de fevereiro de 2007.
- CHIKOFFSKY, E. J.; CROSS, J. H. I. Reverse engineering and design recovery: A taxonomy. *IEEE Software*, v. 7, p. 13–17, 1990.
- FAYAD, M.; SCHMIDT, D. C. Object-oriented application frameworks. *Special Issue on Object-Oriented Application Frameworks*, v. 40, n. 10, 1997.
- FONTANETTE, V.; PRADO, A. F. do; OLIVEIRA, A. L. C. de. Uma abordagem para migração gradativa de aplicações legadas. In: *XVIII Simpósio Brasileiro de Engenharia de Software*. [S.l.: s.n.], 2004.
- GERHARDT, F.; WEGE, C. Comparison of the application architectures in smalltalk and java with respect to migration. In: *STJA 99*. [S.l.: s.n.], 1999.
- JACOBSON, I.; LINDSTRÖM, F. Re-engineering of old systems to an object-oriented architecture. In: *Conference proceedings on Object-oriented programming systems, languages, and applications*. [S.l.]: ACM Press, 1991. p. 340–350.
- JOHNSON, R. E. Components, frameworks, patterns. In: *Proceedings of the 1997 symposium on Software reusability*. [S.l.]: ACM Press, 1997. p. 10–17.
- JOHNSON, R. E.; FOOTE, B. Designing reusable classes. *Journal of Object-Oriented Programming*, v. 1, n. 2, p. 22–35, 1988.
- KNABBEN, A. L.; ROBERT, T. *MAGIC: Um framework para jogos de cartas*. 2002. Trabalho de conclusão de curso – Universidade Federal de Santa Catarina.
- KRAJNC, A.; HERIEKO, M. Classification of object-oriented frameworks. In: *EUROCON 2003. Computer as a Tool. The IEEE Region 8*. [S.l.: s.n.], 2003. v. 2, p. 57–61.
- MACHADO, T. S. *Reengenharia da interface do ambiente SEA*. 2007. Trabalho de conclusão de curso – Universidade Federal de Santa Catarina.
- OBJECTART. *St2J - Smalltalk to Java Converter*. 1997. Disponível em: <http://www.objectart-software.com/> Acesso em: 23 de fevereiro de 2007.
- PIEKARSKI, A. E. T.; QUINÁIA, M. A. Reengenharia de software: o que, por quê e como. In: *Ciências Exatas e Naturais*. [S.l.]: UNICENTRO, 2000. v. 1, n. 2.

SAUVÉ, J. *Projeto de software orientado a objeto*. 2005. Disponível em: <http://jacques.dsc.ufcg.edu.br/cursos/map/html/map2.htm> Acesso em: 03 de janeiro de 2007.

SCHEIDT, N. *Introdução de suporte gerencial baseado em workflow e CMM a um ambiente de desenvolvimento de software*. Dissertação (Mestrado) — Universidade Federal de Santa Catarina, 2003.

SILVA, R. P. e. *Avaliação de metodologias de análise e projeto orientadas a objetos voltados ao desenvolvimento de aplicações, sob a ótica de sua utilização no desenvolvimento de frameworks orientados a objetos*. 1996. Trabalho Individual I – Universidade Federal do Rio Grande do Sul.

SILVA, R. P. e. *Suporte ao desenvolvimento e uso de frameworks e componentes*. Tese (Doutorado) — Universidade Federal do Rio Grande do Sul, 2000.

SILVA, R. P. e; PRICE, R. T. O uso de técnicas de modelagem no projeto de frameworks orientados a objetos. In: *Proceedings of 26th International Conference of the Argentine Computer Science and Operational Research Society (26th JAIIO) / First Argentine Symposium on Object Orientation (ASOO'97)*. [S.l.: s.n.], 1997. p. 87–94.

SILVA, R. P. e; PRICE, R. T. A busca de generalidade, flexibilidade e extensibilidade no processo de desenvolvimento de frameworks orientados a objetos. In: *Proceedings of Workshop Iberoamericano de Engenharia de Requisitos e Ambientes de Software (IDEAS'98)*. [S.l.: s.n.], 1998. v. 2, p. 298–309.