Um Framework para Editores de Diagramas Cooperativos baseados em Anotações

Cleidson Ronald Botelho de Souza

Dissertação de Mestrado

Instituto de Computação
Universidade Estadual de Campinas

Um Framework para Editores de Diagramas Cooperativos baseados em Anotações

Cleidson Ronald Botelho de Souza[1]

Outubro de 1998

Banca Examinadora:

- Jacques Wainer
  Instituto de Computação
  Universidade Estadual de Campinas (Orientador)
- Cirano Iochpe
  Instituto de Informática
  Universidade Federal do Rio Grande do Sul
- Eliane Martins
  Instituto de Computação
  Universidade Estadual de Campinas
- Luiz Eduardo Buzato (Suplente)
  Instituto de Computação
  Universidade Estadual de Campinas

[1] Departamento de Informática - Universidade Federal do Pará
Apoio Financeiro: Coordenadoria de Aperfeiçoamento de Pessoal de Nível Superior (CAPES) e Universidade Federal do Pará - Programa de Incentivo à Capacitação de Docentes e Técnicos.

FICHA CATALOGRÁFICA ELABORADA PELA BIBLIOTECA DO IMECC DA UNICAMP

Souza, Cleidson Ronald Botelho de

So85f Um framework para editores de diagramas cooperativos baseados em anotações / Cleidson Ronald Botelho de Souza -- Campinas, [S.P. :s.n.], 1998.

Orientadores : Jacques Wainer, Cecília Mary Fisher Rubira

Dissertação (mestrado) - Universidade Estadual de Campinas, Instituto de Computação.

1. Framework (Programa de computador). 2. Modelagem. 3. Autoria. 4. Grupos de trabalho. I. Wainer, Jacques. II. Rubira, Cecília Mary Fisher. III. Universidade Estadual de Campinas. Instituto de Computação. IV. Título.

Um Framework para Editores de Diagramas Cooperativos baseados em Anotações

Este exemplar corresponde à redação final da Dissertação devidamente corrigida e defendida por Cleidson Ronald Botelho de Souza e aprovada pela Banca Examinadora.

Campinas, 12 de outubro de 1998.

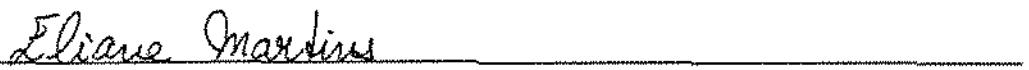
Jacques Wainer
Instituto de Computação
Universidade Estadual de Campinas
(Orientador)

Cecília Mary Fischer Rubira
Instituto de Computação
Universidade Estadual de Campinas
(Co-Orientadora)

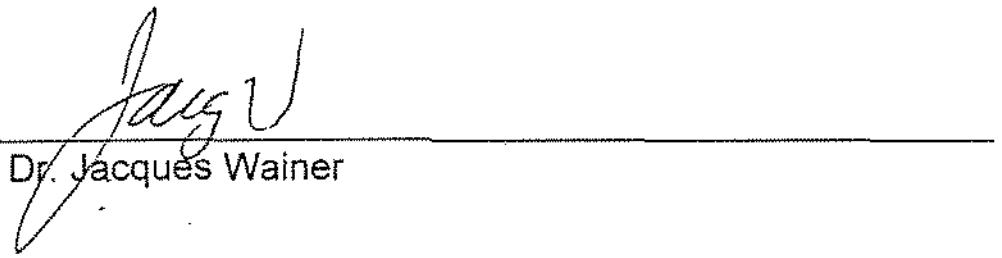
Dissertação apresentada ao Instituto de Computação, UNICAMP, como requisito parcial para a obtenção do título de Mestre em Ciência da Computação.

Tese de Mestrado defendida e aprovada em 02 de setembro de 1998 pela Banca Examinadora composta pelos Professores Doutores

Prof. Dr. Cirano Iochpe



Profa. Dra. Eliane Martins



Prof. Dr. Jacques Wainer

Dedico este trabalho aos meus pais, pelo amor e incentivo
dispensados ao longo de toda minha vida.
Para que onde quer que eles estejam, que eles saibam
do imenso amor que sinto por eles.

"Mas volte o olhar para trás e pergunte a si mesmo se foi tão penoso o caminho.
Difícil apenas ?
Não terá sido belo também ?
Podia imaginar outro tão belo e tão fácil ?"
Herman Hesse (1885-1962)

Agradecimentos

Escrever um agradecimento para quem deve tanto e a tantos como eu é uma tarefa tão difícil, talvez até mesmo mais difícil, que escrever esta dissertação completa. Então, aqui eu irei apenas tentar, mas por favor os nomes que não foram citados aqui me perdoem, eles não foram citados por esquecimento, mas por falta de espaço.

À Deus, por me permitir realizar este sonho;

Aos meus pais, pelo apoio **incondicional**, amor, amizade e presença constante, mesmo que distante, durante estes dois longos anos;

Aos meus irmãos, cunhadas e sobrinhas, pelo apoio, amor e amizade. A distância que nos separava me ensinou a amá-los cada vez mais;

Por não concordar com Humberto Eco que diz que agradecer ao orientador não é de bom gosto, pois este seria um agradecimento óbvio; quero agradecer aos meus orientadores prof. Dr. Jacques Wainer e prof. Dra. Cecília M. F. Rubira, inicialmente, por terem me aceito como orientando, e, principalmente, pelos inúmeros ensinamentos que me transmitiram durante este período;

Aos professores do Instituto de Computação da Universidade Estadual de Campinas, que nas disciplinas, seminários e bancas contribuíram para minha formação profissional;

À minha namorada, Patrícia Nascimento, pelo apoio, amizade e amor durante, principalmente, o início e o final desta jornada;

Aos amigos de república, Alexandre, Delano e Marcos André; pela companhia, amizade, incentivo, paciência, críticas, sugestões, e principalmente, pelos momentos de alegria vividos juntos;

Aos companheiros do *Conselho*; os membros efetivos: Alexandre, Delano e Marcos André, e os colaboradores: Elder, Lucasf, Marcelo de Jesus e outros, pelos momentos de descontração e diversão;

Aos colegas da turma *msc96*, em especial, Alessandra, Adriana Luporini, Amanda, Cristina, Freud, Gisele, Glácіaf, Gutemberg, Janne, Marília, Mário, Pagani, Patrícia e Selma, pelo companheirismo, amizade, críticas, sugestões, festas, e outros "eventos sociais" realizados neste período;

Aos membros do Laboratório de Sistemas Distribuídos (LSD), em especial Oliva e Islene, pelas trocas de informações e idéias sobre diversos assuntos, desde o LaTeX, passando

por Java, até o processo de escrita da dissertação;

Aos alunos do curso de Bacharelado em Ciência da Computação (turma de 94) da Universidade Federal do Pará, pelo apoio na realização dos testes do framework ABCDE e pelas sugestões para sua melhoria;

Ao Departamento de Informática da Universidade Federal do Pará pelo apoio financeiro e pela liberação de atividades para a execução deste trabalho;

À Coordenadoria de Aperfeiçoamento de Pessoal de Nível Superior (CAPES) e Universidade Federal do Pará - Programa de Incentivo à Capacitação de Docentes e Técnicos, pelo apoio financeiro a esta pesquisa consolidado na forma de uma bolsa de Mestrado;

À pequena parcela do povo brasileiro que paga seus impostos, e mesmo sem saber, contribui de forma significativa para o progresso tecnólogico, científico e cultural deste país;

At last, but no at least, a todos aqueles que contribuíram de uma forma ou outra para a realização deste trabalho.

A todos vocês, o meu mais sincero Obrigado.

C.R.B.S.

Resumo

Esta dissertação apresenta um modelo de cooperação para o desenvolvimento cooperativo de software e o framework para editores de diagramas colaborativos ABCDE (*Annotation Based Cooperative Diagram Editor*). A cooperação entre grupos de usuários é obtida através da utilização de anotações sobre diagramas. Anotações são uma forma de comunicar idéias ou opiniões sobre um documento.

Um framework é um projeto abstrato orientado a objetos que pode ser adaptado segundo as necessidades da aplicação. Frameworks fornecem um grau de reutilização de até 80%, pois oferecem reutilização de projeto, servindo como moldes para a construção de aplicações dentro de um domínio. As aplicações instanciadas a partir do framework ABCDE são editores cooperativos de diagramas de classes da notação UML. O ponto adaptável do ABCDE é o modelo de cooperação que ele implementa, o que permite a construção de editores específicos para outras atividades do processo de desenvolvimento de software.

As contribuições desta dissertação são: (i) a especificação de um modelo de cooperação que se baseia na utilização de anotações como mecanismo de auxílio a cooperação, para o desenvolvimento colaborativo de software. Ele oferece apoio à tarefa de revisão e co-autoria, sendo independente da metodologia utilizada para o desenvolvimento do software, podendo ser estendido para apoiar outras atividades do processo de desenvolvimento de software; (ii) o desenvolvimento do framework ABCDE (*Annotation Based Cooperative Diagram Editor*) que permite a construção de editores de diagramas cooperativos, pois implementa o modelo de cooperação desenvolvido; e (iii) a apresentação do sistema ABCDE-Web que apóia o desenvolvimento cooperativo de software através da Internet.

Abstract

This document presents an annotation model for cooperative software development, and an object-oriented application framework for diagram editors called ABCDE (Annotation Based Cooperative Diagram Editor). Cooperation among users is achieved by using annotations on diagrams. Annotations are used by the authors to communicate ideas or opinions about a document.

A framework is an abstract object-oriented design that can be tailored according to specific applications. The framework developed in this work, ABCDE, provides support for the construction of annotation based cooperative diagram editors for class diagrams of UML methodology. The main configurable aspect of ABCDE is the annotation model used, so editors built from ABCDE can be tailored to particular needs of different tasks in software development.

The contributions of this work are: (i) the specification of a cooperation model for cooperative software development. This model uses annotations as a mean to support collaboration, supports the revision and co-authoring tasks, and is independent of methodology for software development. Moreover, it can be extended to support different tasks of software development process; (ii) the framework ABCDE (Annotation Based Cooperative Diagram Editor), which main hot-spot is the annotation model used, because it implements the cooperation model developed; and (iii) the ABCDE-Web system, a prototype develop from ABCDE that can be used for software development on the Web.

Conteúdo

Lista de Tabelas

Lista de Figuras

Capítulo 1

Introdução

A construção dos sistemas atuais de software exige requisitos muito difíceis de serem alcançados como disponibilidade, persistência, distribuição, tolerância a falhas, etc. Um número cada vez maior de pessoas de diferentes especialidades precisa cooperar para desenvolver tais sistemas, o que torna a cooperação uma atividade essencial neste processo. Isto é confirmado por Vessey & Sravanapudi[VS95] que afirmam que projetistas de grandes sistemas passam 70% de seu tempo trabalhando em conjunto.

Apesar desta crescente necessidade de apoiar o desenvolvimento cooperativo de software, as ferramentas de desenvolvimento de software hoje existentes enfocam, em sua grande maioria, o processo individual de desenvolvimento. Na literatura ainda existem poucas ferramentas que apóiam a cooperação entre programadores e desenvolvedores. Como exemplo, temos a ferramenta de Ramesh & Dhar[RD92] para definição de requisitos, o ICARO[LF96] para análise e projeto, e o ICICLE[BSM90] para codificação. Em especial, o ICARO possui um modelo de cooperação simples e limitado à tarefa de revisão.

Para dificultar ainda mais este quadro, a atividade de desenvolvimento de software, a cada dia, é confrontada com um ambiente cada vez mais competitivo, que requer melhor qualidade em seus produtos e um curto período de desenvolvimento. Para resolver este problema, é necessário aumentar a produtividade do processo de desenvolvimento e a qualidade do software desenvolvido. Isto pode ser obtido através de um aumento na reutilização de software.

No entanto, a reutilização de classes e objetos, como têm sido realizada, ainda não é suficiente, pois oferece um baixo grau de reutilização[Fir93]. A utilização de frameworks é uma abordagem proposta na literatura que visa oferecer um elevado grau de reutilização. Um framework é um projeto genérico em um domínio, que pode ser adaptado à aplicações específicas. Um framework bem projetado pode fornecer um grau de reutilização de até 80%[Fir93], pois além da reutilização de código, ele também oferece reutilização de projeto. A reutilização de projeto é mais importante que a reutilização de código, porque o projeto

é mais difícil de criar e recriar do que o código[JF88].

1.1 A Dissertação

No contexto descrito acima, a idéia central desta dissertação consiste na utilização de anotações sobre diagramas como mecanismo de auxílio à cooperação entre grupos de analistas envolvidos na atividade de construção de diagramas. Para isto, é definido um modelo de cooperação que consiste na especificação dos tipos de anotações existentes, no conjunto de operações que podem ser efetuadas sobre estas anotações e que definem o comportamento destas, nos estágios do processo de colaboração, e finalmente, os aspectos relacionados à construção de um editor de diagramas que apóia a utilização de anotações.

Este modelo de cooperação é flexível o suficiente para que possa ser estendido de modo a oferecer apoio adequado aos aspectos específicos de uma determinada atividade do processo de desenvolvimento de software. Entretanto, a construção destes modelos de cooperação, e, conseqüentemente, de diferentes ferramentas que implementem estes modelos é uma atividade custosa e demorada. Idealmente, deve-se fornecer uma infra-estrutura para a construção destes modelos e de suas ferramentas, de modo a tornar este processo mais simples e produtivo. Sob este ponto de vista, esta dissertação apresenta as seguintes contribuições:

(i) a especificação de um modelo de cooperação para o desenvolvimento colaborativo de software[SWR97]. Este modelo baseia-se na utilização de anotações como mecanismo de auxílio à cooperação e foi desenvolvido a partir de idéias provenientes da atividade de escrita colaborativa, que foram estendidas para permitir a inclusão de anotações em diagramas. Ele oferece apoio à tarefa de revisão e co-autoria e é independente da metodologia utilizada para o desenvolvimento do software. Além disso, este modelo pode ser estendido para apoiar outras atividades do processo de desenvolvimento de software.

(ii) o desenvolvimento do framework ABCDE (*Annotation Based Cooperative Diagram Editor*)[SRW98] que permite a construção de editores de diagramas cooperativos, pois implementa o modelo de cooperação desenvolvido. O principal ponto adaptável deste framework é o modelo de anotações implementado, permitindo assim que os editores construídos a partir dele possam ser utilizados em diferentes atividades do processo de desenvolvimento de software. Os pontos adaptáveis de um framework correspondem aos aspectos do domínio da aplicação que devem ser mantidos flexíveis[Pre94a]. Editores construídos a partir deste framework permitem a edição de diagramas de classes com a notação UML[BRJ97, FS97b] e fornecem apoio a anotações baseado no modelo de cooperação definido. A construção do framework

ABCDE foi feita utilizando-se a abordagem de projeto orientado a pontos adaptáveis proposta por Pree[Pre94a] em conjunto com os padrões de projeto de Gamma et *al.*[GHJV94]. A linguagem de programação utilizada na implementação do ABCDE foi Java[1][Fla97]. O projeto do framework ABCDE enfoca aspectos cooperativos de um editor de diagramas, as funcionalidades de um editor convencional são brevemente discutidas.

(iii) a apresentação do sistema ABCDE-Web[SWR98] que apóia o desenvolvimento cooperativo de software através da Internet. Este editor foi desenvolvido a partir do framework ABCDE e permite a cooperação entre grupos dispersos, cujos membros podem estar em diferentes organizações, e até mesmo, diferentes países. Sua utilização requer apenas a utilização de um *browser* que ofereça apoio completo ao JDK 1.1.

1.2 Organização do Texto da Dissertação

A dissertação está organizada da seguinte forma:

Capítulo 2, efetua um levantamento bibliográfico sobre os diversos aspectos relacionados à atividade de autoria cooperativa. A autoria cooperativa consiste na união de um grupo de usuários, homogêneos ou heterogêneos, para o desenvolvimento de um artefato comum. Os aspectos mais importantes deste capítulo são as possíveis formas de cooperação que ocorrem no processo de autoria cooperativa e a apresentação das anotações como mecanismo de auxílio à cooperação.

Capítulo 3, descreve os conceitos básicos de frameworks, padrões de projeto[GHJV94], metapadrões[Pre94a] e descreve as principais metodologias existentes que podem ser utilizadas no desenvolvimento de frameworks.

Capitulo 4, apresenta o modelo de cooperação desenvolvido para o desenvolvimento cooperativo de software, suas características gerais, e possíveis extensões deste modelo para as atividades de inspeção de software e decisões de projeto.

Capítulo 5, descreve o projeto do framework utilizando a abordagem de projeto orientado a pontos adaptáveis[Pre94a] em conjunto com os padrões de projeto[GHJV94]. Além disso, o capítulo 5 também descreve a implementação do editor ABCDE-Web que apóia o desenvolvimento cooperativo de software através da Internet.

Capítulo 6, apresenta algumas conclusões e sugestões para trabalhos futuros.

[1]Java é uma marca registrada da Sun Microsystems, Inc.

Capítulo 2

Autoria Cooperativa

A autoria cooperativa consiste na união de um grupo de usuários, homogêneos ou heterogêneos, para o desenvolvimento de um artefato comum. Este artefato pode ser um texto, uma figura, um hiper-documento, um diagrama, etc. Para apoiar esta atividade diversos protótipos de editores cooperativos surgiram nos últimos anos, como por exemplo, Quilt[FKL88], PREP[NKCM90], GROVE[EGR91], SEPIA[HW92, SHL+92], XNetwork[RS92], etc.

O desenvolvimento do artefato pode ser feito de maneira síncrona, onde um grupo de autores pode editá-lo ao mesmo tempo, ou assíncrona, onde a cada momento o artefato somente pode ser acessado por um único autor. Em qualquer uma destas formas de interação, o editor deve oferecer mecanismos de comunicação entre os autores, mecanismos de percepção das atividades dos outros autores, etc. Além disso, a construção de editores cooperativos cria novos requisitos ao projeto de interfaces, o que levanta vários problemas e requer a descoberta de novas soluções. Cada um destes aspectos será discutido brevemente nas próximas seções. Maiores informações podem ser encontradas em [FKL88, NKCM90, EGR91, PB92, DB92, HW92, BNPM93, MM93]. Ao final, alguns sistemas de autoria cooperativa são apresentados.

2.1 Formas de Cooperação

A autoria cooperativa pode ser feita basicamente de duas formas: revisão e co-autoria. Na *Revisão*, existe um único autor do documento e existem vários revisores que podem fazer comentários sobre o texto, propor alterações, etc; porém, somente o autor pode fazer alterações efetivas no documento. Na *Co-Autoria*, existe mais de um autor para o documento e cada um dos autores tem direitos similares sobre o mesmo.

2.1.1 Revisão

A revisão caracteriza-se pela existência de um único autor que pode modificar o artefato, e por vários revisores que podem apenas fazer anotações sobre o mesmo. A comunicação entre autor e revisores normalmente é feita através de anotações. As anotações são uma forma de comunicar idéias ou opiniões sobre um documento. A revisão também está associada ao ciclo edição-revisão-incorporação[NKCM90]. O autor edita o documento, depois entrega-o para os revisores, que devem avaliá-lo. Os revisores analisam o documento e inserem suas anotações. Estas anotações são avaliadas posteriormente pelo autor, que decide se incorpora ou não as sugestões dos revisores.

Nesta forma de cooperação existe uma definição explicíta de papéis: autores, revisores, editores, etc. Um papel descreve como um conjunto de indivíduos se relaciona com o objeto compartilhado e com os outros indivíduos restantes do grupo[DB92]. Tipicamente, a cada papel é atribuído um conjunto de operações que podem ser efetuadas. Por exemplo, o papel de revisor permite apenas que o usuário faça anotações sobre o documento, ele não permite que o usuário modifique o documento. A atribuição de papéis também é utilizada como um mecanismo de coordenação das atividades dos usuários, e ainda como um mecanismo de controle de acesso a segmentos do documento. No entanto, sua atribuição precipitada pode restringir o potencial criativo dos usuários, ou seja, cada usuário somente efetua as operações específicas de seu papel, uma vez que ele não se sente motivado a efetuar outras operações[NKCM90].

Alguns exemplos de editores que implementam o modelo de revisão são o Quilt[FKL88] e o PREP[NKCM90].

2.1.2 Co-Autoria

A co-autoria caracteriza-se pela existência de um grupo de usuários, onde cada um deles possui direitos similares sobre o artefato. Nesta atividade, normalmente não existe a definição explicíta de papéis, cada co-autor pode efetuar quaisquer alterações no documento.

Apesar de não existirem papéis explicítos, pode ser desejável um controle de acesso a determinados segmentos do documento. Isto pode ser obtido através da especificação de restrições de acesso a partes do documento.

O editor cooperativo mais famoso que implementa a co-autoria é o GROVE[EGR91]. Este editor não define papéis, apenas permite a especificação de restrições de acesso ao documento. Segundo seus autores, a coordenação das atividades é feita através de protocolos sociais definidos pelos usuários.

2.2 Formas de Interação

A interação entre os autores de um documento é uma característica essencial em qualquer sistema de autoria cooperativa, pois ela é necessária para efetivar a cooperação.

Qualquer sistema cooperativo pode ser classificado de duas maneiras em relação ao aspecto temporal da cooperação. *Sistemas síncronos*, que permitem que a comunicação e a cooperação sejam feitas em tempo real, e *sistemas assíncronos*, onde esta interação não é feita em tempo real. Classificações dos sistemas cooperativos em relação a outros aspectos podem ser encontradas em [EGR91].

2.2.1 Editores Síncronos

Os editores síncronos permitem que um grupo de pessoas edite um documento ao mesmo tempo. Normalmente, o documento é dividido em segmentos lógicos e o editor gerencia a sincronização e a consistência do documento[EGR91].

Um dos aspectos mais importantes no desenvolvimento de editores síncronos, consiste no protocolo a ser utilizado para manter a consistência do documento. Isto é necessário, uma vez que a edição simultânea do documento por vários usuários pode ser conflitante. Na literatura encontram-se, basicamente, três abordagens para resolver este problema: um mecanismo de *locking* explícito, onde o usuário precisa travar o objeto antes de editá-lo; um mecanismo de locking implicíto, onde o sistema trava automaticamente o objeto que o usuário está editando; e sem a utilização de mecanismos de locking. Neste último caso, o controle da consistência do documento é feito através de um algoritmo para ordenação dos eventos de edição.

Outro aspecto importante a ser considerado é o projeto de interfaces. A interface de um sistema síncrono deve permitir que cada usuário se mantenha informado a respeito das ações dos outros usuários de forma *não-intrusiva*, além de permitir que dois usuários tenham acesso à mesma informação simultaneamente para, por exemplo, decidir qual a melhor ação a ser tomada. Tais aspectos serão discutidos na seção 2.5 (interfaces).

2.2.2 Editores Assíncronos

Em um editor assíncrono somente um usuário pode editar o documento a cada momento. A cooperação entre os co-autores é possível através de mecanismos que permitam a coleta de informações sobre as atividades executadas pelos co-autores. Assim, no momento em que um co-autor edita um documento, ele terá, a sua disposição, informações sobre as operações que foram efetuadas pelos outros co-autores desde a última vez em que ele acessou o documento.

Assim como os editores síncronos, o projeto de interfaces dos editores assíncronos também é considerado um dos aspectos fundamentais. Um dos aspectos consiste na indicação das atividades efetuadas pelos outros co-autores. Esta indicação deve ser feita de modo não-intrusivo, visando não atrapalhar a edição do documento pelo co-autor atual. Tais aspectos serão discutidos na seção 2.5 (interfaces).

2.2.3 Editores Síncronos ou Assíncronos ?

A distinção entre editores síncronos e assíncronos é bem definida. No entanto, a distinção entre as situações e tarefas adequadas a cada tipo de editor não é. Editores assíncronos se mostram mais adequados a situações onde a responsabilidade sobre o artefato recai sobre um único participante, o autor, e existe um momento preciso onde uma "versão" do artefato está terminada e pode ser revisada. Um exemplo desta situação é o processo de inspeção de software[Pre95]. Pode-se perceber claramente que sistemas assíncronos, normalmente, implementam a revisão como mecanismo de cooperação.

Um editor síncrono é mais adequado à situações onde todos os usuários são responsáveis pelo artefato e onde o processo de construção do mesmo é uma atividade contínua e dinâmica, como por exemplo, no processo de especificação de requisitos de software. Pode-se perceber claramente que sistemas síncronos, normalmente, implementam a co-autoria como mecanismo de cooperação.

Posner & Baecker[PB92] concluíram que ambas as estratégias, síncrona e assíncrona, são utilizadas em diferentes fases de uma atividade de autoria cooperativa, e que um sistema que apóie esta atividade deve fornecer apoio às duas formas de interação, além de permitir uma suave transição entre elas. Tais sistemas são chamados semi-síncronos[DB92]. Minör & Magnusson[MM93] concluíram que a atividade de desenvolvimento de software também é uma atividade semi-síncrona.

2.3 Anotações

As anotações são uma forma de comunicar idéias, sugerir modificações, levantar questões, etc, sobre um documento. Um co-autor faz a leitura do documento e insere suas anotações junto a trechos do mesmo que serão lidas posteriormente por outros co-autores. Estas anotações são inseridas com o objetivo de aperfeiçoar o documento, podendo ser usadas como registro das argumentações ocorridas durante o desenvolvimento do documento. Além disso, elas servem como um mecanismo de comunicação entre co-autores. Esta comunicação pode ser feita utilizando-se anotações de diversas formas: (i) para apresentar idéias ou opiniões, (ii) para propor alterações e (iii) para levantar questões.

Ao se inserir uma anotação em um documento, deve-se registrar o autor da mesma. No

entanto, é interessante também permitir anotações anônimas, o que permite que problemas como, inibição, pressões sociais ou hierárquicas, entre outros, sejam resolvidos[dSBCC95]. Anotações estão associadas a determinados trechos de documentos, como por exemplo, linhas, parágrafos, seções, etc, dependendo da implementação do editor. Por exemplo, o PREP[NKCM90] associa anotações à paragráfos, enquanto que o SHADOW[FP95] associa anotações a trechos de texto.

As anotações geralmente são de dois tipos: *comentários*, que podem ser textuais ou vocais e estão associados a trechos do documento; e *substituições*, que correspondem a sugestões de alterações. Em outras palavras, comentários são uma forma de texto em um meta-nível, ou seja, texto sobre o texto original. Eles contém afirmações, como por exemplo, "Este parágrafo não está claro", as quais são associadas com o parágrafo em questão.

Substituições são segmentos de texto que podem substituir o texto original. As substituições são extremamente importantes em qualquer editor que apóie anotações, pois segundo Neurwith *et al.*[NKCM90] os usuários ficam insatisfeitos se não podem propor alterações no documento, ou seja, " (os revisores) querem ter a possibilidade de reescrever peças de texto e não simplesmente inserir anotações (...)". Neurwith *et al.* também afirmam que, em muitos casos, a reescrita é muito mais eficiente que a tentativa de diagnosticar as deficiências do texto original.

Um dos problemas em se utilizar anotações refere-se ao projeto de interfaces do editor. Ao mesmo tempo em que é importante apresentar as anotações associadas aos elementos do documento, para que os co-autores possam perceber quais elementos foram anotados; a presença de tais anotações é considerada intrusiva e pode dificultar o trabalho dos autores.

2.4 Percepção

Uma das questões mais importantes em qualquer sistema cooperativo refere-se à percepção[1] de um indivíduo sobre o trabalho do restante do grupo. Em outras palavras, um co-autor precisa, de alguma forma, estar ciente do trabalho dos outros co-autores para que ele possa executar corretamente o seu próprio trabalho. Esta percepção serve como uma forma de coordenação das atividades e ajuda a evitar trabalho conflitante ou redundante.

Dourish & Belloti[DB92] definem a percepção como:

> "(...) o entendimento das atividades dos outros (usuários) o que fornece o contexto para a sua própria atividade (...)"

Sendo que o contexto é definido não apenas como o conteúdo das contribuições individuais, mas o seu significado para o grupo como um todo e seu objetivo. Este contexto

[1] Em Inglês: *awareness*.

é usado para garantir que cada contribuição individual seja relevante para o grupo como um todo, e para avaliar as ações individuais em relação ao progresso e objetivos do grupo.

Em sistemas não-cooperativos, a questão da percepção não é precisa ser tratada porque um único autor é capaz de gerenciar todo o contexto de seu trabalho. Assim, se ele efetua uma mudança no documento, e esta mudança implica em outras alterações no documento, ele está ciente de que estas alterações devem ser feitas. Isto não ocorre em sistemas cooperativos, onde um co-autor precisa ter ciência de que uma determinada alteração foi feita no documento por outro usuário, pois isto implica em mudanças na parte do documento sob sua responsabilidade. Somente através desta percepção é que um usuário pode compreender a atividade dos outros usuários e assim adequar o seu trabalho às necessidades do grupo.

Segundo [DB92] a informação sobre a percepção pode ser dividida em dois níveis: a informação de alto-nível que se refere ao *significado* das ações dos outros usuários, permitindo ao usuário estruturar sua atividade e evitar duplicação de esforço; e a informação de baixo-nível, que se refere ao *conteúdo* do trabalho dos outros usuários.

Os mecanismos mais comuns de oferecer informação sobre a percepção são o informacional e a atribuição de papéis[DB92]. O mecanismo *informacional*, consiste em fornecer ao sistema facilidades através das quais os usuários podem informar as alterações efetuadas. Isto pode ser feito através do envio de mensagens (correio eletrônico), anotações ou arquivos de *log*. Esta informação também pode ser coletada automaticamente, como por exemplo no editor Quilt[FKL88]. No entanto, a coleta automática não permite a coleta de informações sobre o conteúdo do trabalho dos outros usuários. A *atribuição de papéis*, permite a coordenação das atividades, facilita o controle de acesso à regiões do diagrama e diminui o grau de incerteza a respeito do trabalho dos outros co-autores, pois fornece uma percepção clara sobre o que os outros co-autores podem fazer[Die96].

Os dois mecanismos apresentam problemas[DB92], que vão desde a adição de trabalho extra aos co-autores no mecanismo informacional, o que pode levar ao fracasso do sistema cooperativo[Gru94], até a restrição do potencial criativo dos co-autores devido a uma atribuição prematura dos papéis, ou seja, cada co-autor somente efetua as operações específicas do papel que lhe foi atribuído, uma vez que ele não se sente motivado a efetuar outras operações[NKCM90].

Em sistemas assíncronos pode-se afirmar que a informação sobre a percepção consiste em identificar as ações que os outros usuários *executaram* no documento. Tal informação pode ser obtida através de mecanismos como correio eletrônico; arquivos de *log*; registro automático das ações dos outros autores, de modo que um usuário possa executá-las novamente[2]; e através da comparação de estados do artefato. O estado inicial corresponde ao último estado conhecido pelo usuário, e o estado final corresponde ao estado atual do

[2]Conhecido na literatura como *audit trail recording* ou *history lists*.

artefato. Normalmente, este último mecanismo é obtido através da comparação de versões do artefato, como é feito por exemplo no Quilt [FKL88]. Em qualquer um dos mecanismos acima deve ficar a cargo do co-autor recuperar as informações.

Em sistemas síncronos pode-se afirmar que a percepção consiste em identificar as ações que os outros usuários *estão executando* em um determinado momento. Assim, a primeira informação necessária consiste na identificação dos usuários correntes da sessão. Isto normalmente é feito através da apresentação dos nomes e/ou fotos dos usuários presentes na sessão. Canais de áudio e vídeo também podem ser utilizados de modo a oferecer um nível informal de percepção, ou seja, os co-autores podem dialogar para obter a informação sobre o trabalho dos outros co-autores.

O mecanismo mais utilizado em sistemas síncronos são recursos de interface do sistema, como por exemplo WYSIWIS (*What You See Is What I See*), telepointers, etc, que serão discutidos na próxima seção. Nestes sistemas, um dos principais problemas que ocorrem refere-se ao desempenho, pois as alterações efetuadas em uma instância do documento precisam ser propagadas para todas as outras instâncias ativas, o que consome tempo e sobrecarrega a rede[Die96].

2.5 Interfaces

Muitas idéias provindas da pesquisa em interfaces de sistemas mono-usuários podem ser utilizadas em sistemas cooperativos; no entanto, é necessária uma mudança de perspectiva [EGR91]. No primeiro caso, o usuário está interagindo com o computador e está no controle das ações, tendo conhecimento de que suas próprias operações podem modificar o artefato. No entanto, em sistemas cooperativos existem outros usuários interagindo com o sistema, provocando modificações no mesmo sem que outros usuários tenham controle sobre isto. A interface do sistema, além de ser um mecanismo de comunicação do usuário com o computador, também torna-se um mecanismo de comunicação de um usuário com outros usuários[dSBCC95].

Segundo Bentley *et al.*[BRSS94] diferente dos sistemas multi-usuários, que dão a ilusão que o usuário está trabalhando sozinho, a interface dos sistemas cooperativos deve encorajar a cooperação e permitir que todos os co-autores percebam a presença e a atividade dos outros co-autores de forma *não-intrusiva.*

A forma como as informações compartilhadas são apresentadas nas interfaces das instâncias ativas do sistema é denominada *acoplamento da interface.* Este acoplamente refere-se a editores síncronos, onde mais de um usuário utiliza o editor ao mesmo tempo. Bentley *et al.*[BRSS94] definem três tipos de acoplamento:

- Interfaces fortemente acopladas: cada usuário compartilha as mesmas informações na mesma forma de apresentação. Ou seja, todos os usuários possuem a mesma visão

do documento. Neste caso, quando um usuário executa a rolagem da janela, o mesmo acontece em todas as instâncias ativas do editor. Esta forma refere-se também a dimensão e posição da janela de apresentação e é conhecida como WYSIWIS (*What You See IS What I See*) forte[SBF+86][3].

- Interfaces com acoplamento médio: cada usuário compartilha as mesmas informações, mas a forma de apresentação é diferente. Assim, um co-autor pode estar visualizando um gráfico, enquanto que outro está visualizando os mesmos dados na forma de uma tabela. Este esquema é conhecido como WYSIWIS relaxado[SBF+86].

- Interfaces sem acoplamento: as informações apresentadas em cada interface das instâncias ativas são totalmente diferentes. Assim, um co-autor pode estar visualizando uma seção do documento, enquanto que outro co-autor está visualizando uma seção diferente.

Sistemas síncronos também implementam o mecanismo de telepointer. Um telepointer é um cursor visível em todas as janelas que a cada momento é controlado por um único usuário. Desta forma, o usuário que possui o telepointer pode apontar regiões do documento para discussão, marcar regiões do documento, etc, e todos os outros usuários ativos poderão perceber a região que está sendo apontada.

É importante que os sistemas de autoria cooperativa permitam a visualização das informações compartilhadas em janelas diferentes, as quais possam ser configuradas pelo autor. Em especial, é importante que cada co-autor tenha disponível uma janela de edição privativa onde ele possa escrever e validar suas contribuições antes de divulgá-las ao grupo[Die96]. Esta janela age como um bloco de rascunho individual, onde cada co-autor tem liberdade de criação sem que se sinta pressionado por aspectos hierárquicos ou psicológicos.

Uma outra abordagem interessante utilizada no sistema SASSE[BNPM93] (*Synchronous Asynchronous Structured Shared Editor*) consiste na utilização de múltiplas barras de rolagem. A barra mais à direita na janela de texto do usuário pertence ao próprio usuário, enquanto que uma outra barra de rolagem com pontos coloridos marcados corresponde a barra de rolagem global. O usuário pode se mover para os pontos marcados e visualizar os trabalhos dos outros co-autores a partir desta barra. Através de estudos de usabilidade feitos pelos autores do SASSE foi comprovado que estas barras aumentam a percepção dos usuários.

Em editores assíncronos, existe apenas um usuário utilizando o editor a cada momento, logo o projeto de interfaces destes editores refere-se à apresentação das alterações efetuadas pelos outros autores. Esta apresentação normalmente é feita com a utilização de cores

[3]Em Inglês: *strict WYSIWIS*.

e/ou ícones para identificar as partes do objeto que foram modificadas. Além disso, mecanismos como a comparação de estados do objeto e registro automático das ações dos outros co-autores são utilizados.

A utilização de cores e/ou ícones é amplamente utilizada em sistemas cooperativos, síncronos ou assíncronos, para a apresentação de características ou estados de um objeto. Como por exemplo, quem são os autores responsáveis por um segmento do documento, que regiões do documento estão sendo editadas no momento, que alterações foram feitas em um determinado documento, etc.

O projeto de interfaces de editores cooperativos que utilizam anotações também possui problemas. O problema refere-se à apresentação das anotações de forma não-intrusiva, pois, a indicação das anotações diretamente sobre o artefato pode prejudicar sua visualização, e conseqüentemente, sua avaliação[RS92]. Mesmo que as anotações sejam apenas indicadas nos pontos do documento onde elas foram efetuadas, como no SHADOW[FP95], sua presença ainda gera problemas. Neurwith *et al.*[NKCM90] afirmam que a indicação das anotações na forma de hiper-textos, onde o usuário precisa procurar anotações, clicar para visualizá-las, procurar anotações, clicar, e assim por diante, como nos sistemas Quilt[FKL88], XNetwork[RS92] e SHADOW[FP95], não é desejável, uma vez que os usuários querem ter a possibilidade de avaliar rapidamente todas as anotações que foram efetuadas. A solução adotada por Neurwith *et al.* no editor PREP[NKCM90] consiste na utilização de colunas alinhadas. Assim, uma coluna contém, por exemplo, o conteúdo do texto, outra contém as anotações de um autor, outra o planejamento do autor sobre o texto, e assim por diante. Cada coluna contém seus dados alinhados ao paragráfo sobre o qual foram construídos. Uma outra justificativa para a indicação das anotações separadas do documento é que qualquer sistema cooperativo deve permitir que seus usuários o utilizem de forma não-cooperativa e desta forma, as anotações poderiam atrapalhar o trabalho do usuário.

Outra abordagem existente na literatura para o problema das anotações consiste na utilização de filtros para a apresentação seletiva de anotações. Esta idéia foi proposta originalmente por Engelbart[EGR91] e também é citada por Reeves & Shipman[RS92] no editor XNetwork(2.6.4). A idéia básica é apresentar apenas um subconjunto das anotações, ao invés de apresentar todas. Por exemplo, um filtro de autoria permitiria a apresentação apenas das anotações inseridas por um determinado co-autor, um filtro de data apresentaria apenas as anotações inseridas após uma determinada data, e assim por diante.

2.6 Sistemas Existentes

Um estudo sobre os principais sistemas de autoria cooperativa existentes foi realizado como forma de obter uma visão geral sobre a área. O objetivo deste estudo é buscar idéias e soluções já experimentadas de problemas da área ou simplesmente buscar material para novas soluções.

2.6.1 GROVE

GROVE[EGR91] (*GRoup Outline Viewing Editor*) é um editor de textos síncrono projetado para ser utilizado por um grupo de usuários para a edição de textos. Cada usuário pode ver e manipular uma ou mais visões do documento em múltiplas janelas na sua interface. Uma visão corresponde a um conjunto de itens do documento e pode ser apresentada de diferentes formas para usuários diferentes.

As visões do GROVE podem ser classificadas em *privadas*, nas quais somente o usuário que a criou pode ter acesso, *compartilhadas*, nas quais um conjunto de usuários pode acessá-la, e *públicas*, nas quais todos os usuários podem acessá-la.

O GROVE também utiliza o conceito de janelas de grupo, que são responsáveis por apresentar as visões compartilhadas e públicas. Além disso, elas também indicam quais usuários estão participando da sessão em um dado momento. Estas informações são fornecidas através da apresentação das fotos ou nomes dos usuários que estão participando da sessão na própria janela de grupo. Conforme os usuários entram ou saem da sessão, suas fotos aparecem e desaparecem da janela. As janelas de grupo utilizam um esquema de cores pré-definido para indicar as permissões de leitura e escrita dos itens, para indicar se algum usuário está editando o item naquele momento, etc. Além disso, o usuário também pode modificar as permissões de leitura e escrita destes itens.

Os usuários podem entrar ou deixar uma sessão a qualquer momento. Quando um usuário entra, ele recebe a cópia mais atualizada do documento, a não ser que ele requeira explicitamente uma versão mais antiga. O usuário pode modificar o documento utilizando as operações padrão de edição em uma janela de grupo. Quando isto acontece, todos os usuários são automaticamente notificados das modificações. Uma sessão termina quando não existem mais usuários utilizando o sistema.

GROVE utiliza uma arquitetura que possui uma instância do editor e uma cópia replicada do documento na estação de cada usuário. O controle de concorrência é obtido através da adoção de uma técnica de transformação de operações, que permite a resolução automática de conflitos. Quando um co-autor realiza alguma operação, ela é executada imediatamente na sua cópia local. Após a execução local, ela é distribuída para outras cópias, de acordo com um vetor de estados que controla e indica as operações mais recentemente processadas em cada réplica do documento. Cada instância do GROVE mantém

um vetor de estados para sua réplica. Antes de realizar a operação que chegou de outra instância, é feita uma comparação entre os dois vetores de estado: se forem iguais, a operação se realiza, caso contrário, ocorre uma transformação automática dos dados para ficarem consistentes, antes da operação ser processada.

O GROVE implementa um modelo de cooperação fortemente acoplado, onde cada usuário pode ver e editar livremente qualquer parte do documento sem a necessidade de mecanismos de *locking*. Isto permite que dois usuários editem a mesma linha do documento ao mesmo tempo. Não existe uma regra pré-determinada para arbitrar o acesso concorrente a uma mesma região do documento. A resolução de conflitos é feita de maneira direta entre os dois usuários através do canal de comunicação de áudio existente. Apesar da total liberdade deste modelo de cooperação, os autores relatam um aumento de eficiência do processo de escrita cooperativa e relatam que os conflitos são surpreendentemente **infreqüentes**. Segundo seus autores, isto ocorre porque após um determinado período de aprendizado do editor, os próprios usuários se encarregam de estabelecer protocolos sociais para a cooperação.

O modelo de interface do GROVE utiliza a abordagem WYSIWIS relaxada[SBF+86] onde cada usuário tem total controle sobre detalhes, como posição e tamanho da janela, forma de visualização dos dados, etc. Além disso, o GROVE utiliza um modelo chamados pelos autores de modelo de nuvens[4]. Neste modelo as modificações feitas por usuários são automaticamente visíveis, enquanto que as modificações feitas por outros usuários são indicadas através da presença de nuvens sobre o texto original. A posição e o tamanho da nuvem indicam a localização e a extensão aproximada da modificação. Quando o usuário pára de escrever por um tempo, as nuvens desaparecem e o texto é apresentado segundo uma gradação de cores que vai do azul claro até o preto. O motivo para este modelo é que enquanto um usuário está escrevendo, ele está apenas *remotamente interessado* nas modificações feitas pelos outros usuários, as quais devem ser inicialmente indicadas e posteriormente apresentadas[EGR91].

GROVE não prevê a definição explicíta de papéis. As atividades dos usuários são conduzidas seguindo-se apenas os protocolos sociais.

2.6.2 Quilt

Quilt[FKL88] é uma ferramenta de apoio a autoria cooperativa com recursos de anotações, troca de mensagens e conferências com auxílio de computador. Seus mecanismos são implementados de forma a ficarem independentes do editor e do sistema de correio eletrônico, permitindo assim que os usuários continuem a utilizar os sistemas com os quais já estão familiarizados.

[4] Em Inglês: *cloudburst*.

Os requisitos coletados pelos projetistas do Quilt a partir de pesquisas, entrevistas, questionários e observações, levaram-os a concluir que qualquer ferramenta de apoio à autoria cooperativa deve fornecer apoio à comunicação entre os co-autores, além das capacidades comuns de edição. Assim, o objetivo principal do Quilt é a melhoria da comunicação entre os co-autores. Isto é feito através de mecanismos de anotações, correio eletrônico e conferências por computador.

Quilt utiliza uma estrutura de hiper-textos para auxiliar o processo de escrita. Um documento no Quilt consiste em um nodo principal onde o documento é armazenado, e nodos de hiper-textos associados a este documento ou a outros nodos.

As anotações correspondem a nodos que são feitos sobre outros elementos da estrutura do hiper-texto. Seu modelo de cooperação também permite anotações sobre anotações. Cada anotação pode apresentar as seguintes permissões de acesso:(i) privadas, que são visíveis somente para o autor do nodo; (ii) públicas, visíveis para qualquer usuário que tenha permissão de leitura no nodo; ou (iii) mensagens diretas, como anotações no texto ou como mensagens no sistema de correio eletrônico, que só podem ser lidas por usuários ou grupo de usuários específicos.

O sistema permite a criação de versões do documento e suas ligações com a finalidade de manter um histórico das atividades, e também oferece um mecanismo de comparação automático de versões que mostra as modificações ocorridas entre as versões.

A definição de papéis é utilizada como mecanismo de coordenação de atividades. Quilt oferece uma série de papéis e modelos de cooperação pré-definidos e permite a criação de novos modelos e/ou papéis.

O editor Quilt utiliza dois mecanismos para manter os usuários informados a respeito do trabalho dos outros usuários: um esquema de *log* e mensagens semi-estruturadas. Um log é mantido para cada nodo do hiper-texto armazenando a interação de cada co-autor com aquele nodo. As entradas para este log podem ser geradas (i) pelo usuário, onde ele pode resumir as alterações feitas no nodo; (ii) pelo próprio Quilt, indicando por exemplo os parágrafos que foram modificados, o autor das modificações, etc; e (iii) uma combinação de ambos. O segundo mecanismo empregado é a utilização de mensagens semi-estruturadas[MGL+87], as quais podem ser criadas e manipuladas automaticamente pelo sistema, permitindo, por exemplo, que gatilhos (*triggers*) sejam criados e disparados quando necessário.

A arquitetura do Quilt é implementada em camadas. A camada de mais alto nível consiste em um processo monitor colocado entre o editor de textos e o usuário. Este editor mapeia os comandos do usuário para o editor, para a camada de interface do Quilt ou para a camada de banco de dados, conforme necessário. O editor de textos utilizado é o próprio editor com o qual o usuário já está familiarizado. O banco de dados armazena informações sobre usuários, estilos de cooperação, simula um hiper-texto para armazenar

anotações, etc. Finalmente, a última camada é a camada de interface que é responsável pela comunicação entre o sistema e o usuário.

2.6.3 PREP

PREP[NKCM90] (*work in PREParation*) é um editor assíncrono que pode ser usado por um grupo de autores para a escrita de um documento, ou também pode ser usado como uma ferramenta individual.

Um documento PREP é estruturado como um conjunto de segmentos, os quais podem ser textos, gráficos, tabelas, etc. Um segmento corresponde a uma idéia e todos os segmentos são armazenados em um banco de dados compartilhado. O PREP permite a criação de ligações entre os segmentos, expressando relacionamentos entre as idéias que eles representam.

A principal diferença entre o PREP e outros escritores cooperativos é a sua interface. Apesar de ser baseada em um sistema de hiper-textos, sua interface não é similar a tais sistemas, onde para que o usuário acesse a informação ele precisa seguir as ligações nodo após nodo. Ao invés disso, a interface do PREP é apresentada como uma grade onde as colunas apresentam-se alinhadas e cada coluna pode possuir diferentes conteúdos. Por exemplo, um usuário pode utilizar uma coluna para descrever o planejamento do texto e outra coluna para o conteúdo propriamente dito. O revisor utiliza uma terceira coluna para apresentar seus comentários que são visualmente alinhados ao conteúdo do texto possibilitando aos leitores um claro relacionamento entre comentários, conteúdo e planejamento do texto. Este alinhamento também permite que os leitores tenham uma rápida e fácil visualização dos comentários sobre o texto.

A possibilidade do autor apresentar o planejamento do texto, permite a comunicação entre autor e revisores. Além disso, facilita o trabalho dos revisores que não precisam inferir sobre o trabalho do autor[NKCM90].

O PREP permite a criação de revisões do texto, as quais correspondem ao texto do documento após um ciclo edição-revisão-incorporação. O autor ao terminar de incorporar as modificações sugeridas pelos revisores pode criar tais revisões permitindo que as atualizações nos segmentos não destruam o conteúdo anterior, o qual pode ser consultado posteriormente para comparações.

2.6.4 XNetwork

XNetwork[RS92] é um protótipo assíncrono de um ambiente de projeto[FGNR92] para apoiar o projeto cooperativo de redes de computadores. Ele consiste em duas partes distintas: um componente de construção, cujos itens pertencem ao domínio da aplicação onde os projetistas podem construir diagramas de projeto, e um componente de anotações,

que permite que os projetistas incluam comentários sobre os elementos do projeto. A comunicação entre os projetistas é feita através destas anotações. Esta abordagem é diferente dos sistemas de anotações anteriormente citados, pois integra os componentes de construção e anotação.

Uma das principais características do XNetwork é a integração das discussões sobre o projeto ao próprio projeto (diagrama). Isto evita os seguintes problemas [RS92]:

- decisões de projeto sejam esquecidas de ser implementadas;
- problemas previamente resolvidos tenham de ser refeitos; e
- permite uma correspondência direta entre uma característica do artefato e a discussão que a motivou.

O XNetwork permite a visualização do projeto em diferentes níveis de abstração. Assim, ele permite que a subrede que está sendo projetada seja observada em relação à rede global a qual ela está ligada. Isto é necessário devido ao tamanho e complexidade dos projetos de redes de computadores existentes que possuem até centenas de computadores. Além disso, diferentes visões da mesma rede podem ser criadas. No entanto, se tais visões são modificadas, elas acarretam modificações no projeto original. Se o usuário não desejar modificar o projeto original, ele pode criar cópias do projeto. As cópias do projeto podem ser modificadas não acarretando mudanças no projeto original.

Quando uma discussão não é mais necessária no componente de construção, ela pode ser movida para uma página de argumentação. Estas páginas podem conter informação textual, assim como as partes do projeto que foram anotadas, permitindo que a discussão sobre o projeto seja armazenada de maneira estruturada.

A versão do XNetwork apresentada em [RS92] apresenta a limitação de que um pequeno número de anotações no diagrama de projeto já é suficiente para dificultar a sua visualização. Os autores propõem a utilização de filtros para evitar este problema.

2.6.5 SEPIA

SEPIA[HW92, SHL+92] é um sistema de autoria cooperativa de hiper-documentos que pode ser utilizado tanto na forma síncrona como assíncrona. Ele apóia a criação de hiper-documentos através do apoio à diferentes aspectos do processo de criação de hiper-documentos, chamados *espaços de atividades*. Cada um dos espaços pode ser visualizado separadamente através de um *browser* que oferece apoio a tarefas específicas daquele aspecto.

O sistema SEPIA inicialmente foi projetado como um sistema não-cooperativo e depois foi estendido para oferecer suporte a cooperação. Os elementos básicos do SEPIA são

nodos atômicos, nodos compostos, ligações rotuladas, e nodos e ligações de anotações. Assim, ainda em sua versão não-cooperativa, o SEPIA já oferecia apoio a tarefa de revisão, pois permitia que outros usuários fizessem anotações sobre o trabalho dos outros usuários.

O sistema não possui apoio à organização do processo de cooperação, embora a estrutura do hiper-documento permita que nodos compostos sejam associados a grupos de autores. Tais nodos são utilizados para agrupar informações relacionadas e manter uma estrutura hierarquicamente organizada das informações. Múltiplos nodos podem conter o mesmo nodo composto o que permite o compartilhamento de informações entre os co-autores. Existe apenas uma única janela para cada nodo composto e um usuário pode abrir vários nodos compostos em sua estação de trabalho.

A cooperação entre os co-autores para a criação de documentos pode acontecer de três formas diferentes:

- Individual: somente um usuário encontra-se trabalhando em um determinado nodo. Esta forma corresponde ao modo assíncrono e não requer nenhum mecanismo de colaboração, podendo ser feita da mesma forma que em editores não-cooperativos. O banco de dados do SEPIA faz todo controle de acesso ao documento compartilhado.

- Fracamente acoplada: existem dois co-autores trabalhando em um mesmo nodo composto. Neste estilo, a percepção sobre o trabalho dos outros autores torna-se fundamental para evitar trabalho redundante ou conflitante. Logo, cada co-autor pode visualizar a ação dos outros usuários e é livre para navegar no documento. Um mecanismo de *locking*, indicado através de cores, é utilizado para evitar acesso concorrente ao mesmo objeto. Este mecanismo também é utilizado na próxima forma de cooperação.

- Fortemente acoplada: existem dois co-autores compartilhando a mesma visão do conteúdo do nodo composto. Esta forma corresponde à edição síncrona e ao modelo WYSIWIS e utiliza:

  - visões compartilhadas, onde cada autor visualiza o mesmo espaço de atividades em termos de tamanho e conteúdo da visão;
  - um *telepointer* usado para a indicação de itens que estão acessíveis na janela compartilhada; e
  - um canal de áudio que apóia a comunicação verbal entre os co-autores.

  A rolagem da janela compartilhada de um co-autor implica em uma rolagem idêntica nas janelas dos co-autores atuais. O telepointer em conjunto com o canal de áudio permite aos co-autores apontarem itens do nodo composto e efetuarem uma discussão verbal sobre o conteúdo do mesmo.

A característica mais importante do SEPIA é a possibilidade de uma transição suave entre os três modos de cooperação. Se vários co-autores estão acessando um mesmo hiperdocumento, em algum momento eles podem abrir um mesmo nodo composto, o que inicia automaticamente uma sessão fracamente acoplada. Caso um usuário entre (saia) nesta sessão, todos os browsers dos espaços de atividades abertos são atualizados, indicando a presença (ausência) do usuário. A qualquer momento um usuário pode solicitar a abertura de uma sessão fortemente acoplada, indicando quais usuários ativos ele gostaria que participassem da sessão. O sistema automaticamente pergunta aos usuários selecionados se eles querem participar desta sessão. Após a resposta de todos os usuários, a sessão será iniciada com os usuários que confirmaram sua participação.

Em cada browser de cada espaço de atividades existe uma linha de status que indica o nome dos usuários correntes que estão visitando o nodo e o modo de cooperação destes usuários. Esta linha é atualizada sempre que um usuário entra (sai) de uma sessão.

O sistema SEPIA não define papéis explicitamente, entretanto, o controle de acesso aos nodos compostos serve como forma de coordenação das atividades dos co-autores[DB92].

2.6.6 ICARO

ICARO (*Interface for Collaborative ARgumentation of Object-oriented systems*) [LF96] é um editor de diagramas cooperativo construído a partir de uma ferramenta *CASE* comercial. Esta ferramenta foi modificada para gerar páginas *html* a partir dos diagramas. Estas páginas contém *Applets*, que são pequenos programas escritos na linguagem Java que podem ser trazidos pela Internet através de um *browser HTML* e são executados na máquina cliente. Estes applets permitem que os projetistas possam discutir sobre o diagrama através da utilização de anotações. As anotações utilizadas no ICARO limitam-se a comentários, e podem ser efetuadas sobre elementos unitários do diagrama. A discussão também é disponibilizada em páginas *html*, embora não esteja diretamente integrada ao diagrama.

O modelo de cooperação adotado é o modelo de revisão. Os projetistas limitam-se a sugerir alterações no diagrama, pois não tem acesso ao diagrama original. O gerente de configuração leva a requisição de alteração até uma reunião para avaliação, se ela for aprovada o diagrama é alterado e as devidas medidas são tomadas.

A principal característica deste editor é a utilização da Internet como ambiente para a disseminação dos artefatos e da discussão sobre eles. Isto permite a cooperação entre grupos de trabalho dispersos, cujos membros podem estar em diferentes organizações e até mesmo diferentes países.

2.6.7 Dias e Xexéo

Dias e Xexéo[DX97b, DX97a] descrevem um editor de diagramas cooperativo síncrono, implementado utilizando a linguagem de programação Java sobre uma arquitetura cliente-servidor. O servidor mantém uma base centralizada com uma cópia de cada documento compartilhado e, através do gerente de cooperação, controla a edição concorrente de documentos. Cada cliente mantém uma cópia consistente do documento. Este editor não permite a inserção de anotações sobre os elementos do diagrama.

O mecanismo de controle de concorrência utiliza um esquema de *locking* que opera em dois níveis: o da informação semântica e o da representação visual. Enquanto o servidor gerencia a parte semântica do modelo, o cliente é que se responsabiliza pela representação visual do mesmo. Esta idéia é a mesma do padrão de projeto Observador[GHJV94, 293] e permite que, quando um usuário esteja editando um elemento, apenas a informação semântica seja travada, e não sua representação visual. Assim, os outros usuários podem mover, modificar o tamanho ou a aparência visual do elemento que está sendo editado por outro usuário.

2.6.8 Microsoft Word

O Microsoft Word 97[Mic97][5] é um dos mais famosos editores de textos, e provavelmente o mais utilizado, para microcomputadores pessoais. Em suas versões anteriores, o Word não incluía mecanismos para apoiar o trabalho em grupo. Entretanto, versões mais recentes deste editor começam a incluir tais mecanismos.

O Word é um editor assíncrono que implementa a revisão como modelo de cooperação. A revisão pode ser feita através de duas maneiras *exclusivas*: controle de alterações e comentários[Mic97]. O controle de alterações corresponde a anotações do tipo substituição (seção 2.3). A principal limitação do Word, é que ele não permite que os dois mecanismos sejam utilizados ao mesmo tempo. O usuário utiliza comentários *ou* substituições.

O *controle de alterações*, permite que os outros autores editem o documento da maneira habitual. Conforme os usuário incluem, removem ou movimentam trechos de texto do documento, o editor se encarrega de colocar marcas de revisão no documento. Estas marcas de revisão indicam as alterações que foram feitas no texto. As marcas de revisão apresentam uma formatação especial conforme o tipo de modificação que foi feito. O Word permite que estas formatações sejam modificadas a critério do usuário. Além disso, ele coleta informações como o autor da revisão, data e hora da revisão, etc. Assim, quando o autor recebe novamente o documento, ele pode identificar as revisões que foram feitas. O Word também permite que as marcas de revisão sejam incorporadas ao documento original, evitando que o usuário tenha de refazer o trabalho feito pelos revisores.

[5] Word é uma marca registrada da Microsoft Corporation.

Os *comentários*, são utilizados pelos revisores para escrever textos sobre o texto original. Os trechos de texto que foram comentados são indicados no texto, através da modificação de suas cores. Além disso, uma marca de comentário numerada é inserida ao final do trecho que foi comentado. Para que o autor visualize o comentário, ele necessita clicar sobre esta marca. O Word também permite comentário vocais.

O usuário pode marcar determinadas regiões do texto para que os revisores concentrem-se nestas áreas. Existe um mecanismo no Word que permite que as revisões não sejam apresentadas no texto, desta forma, o usuário pode continuar seu trabalho sem ser atrapalhado pela presença das revisões.

Além das anotações, o Word oferece as seguintes funcionalidades: *Controle de Versões*, tal que os usuários podem rastrear e armazenar todas as versões anteriores de um documento em um arquivo. E,*Merging de documentos*, tal que os usuários podem facilmente consolidar alterações feitas em múltiplas cópias de um documento de volta ao original. Isto permite que os usuários compartilhem documentos, mesmo não estando conectados a uma rede.

2.7 Resumo

A autoria cooperativa consiste na união de um grupo de usuários para o desenvolvimento de um artefato comum. Este artefato pode ser um texto, um hiper-documento, um diagrama, etc. O desenvolvimento cooperativo de um artefato pode ser feito de duas formas: interação síncrona ou assíncrona. Na interação síncrona os usuários trabalham sobre o artefato ao mesmo tempo, enquanto que na interação assíncrona apenas um usuário pode acessar o artefato a cada momento. Ambas as estratégias são utilizadas em diferentes fases de uma atividade de autoria cooperativa, e um sistema que apóie esta atividade deve fornecer apoio às duas formas de interação, assim como uma suave transição entre elas. Um sistema que possui esta característica é dito semi-síncrono.

A autoria cooperativa pode ser feita, basicamente, de duas formas: a primeira, chamada *Revisão*, onde existe um único autor do documento e existem vários revisores que podem fazer comentários sobre o texto, propor alterações, etc, porém somente o autor pode fazer alterações efetivas no documento. A segunda, chamada *Co-Autoria*, onde existe mais de um autor do documento e cada um dos autores tem direitos similares sobre o documento.

Um mecanismo de auxílio à cooperação implementado em diversos editores cooperativos, é a possibilidade de usuários inserirem anotações no documento. As anotações são uma forma de comunicar idéias, sugerir modificações, levantar questões, etc. sobre um documento. Um co-autor faz a leitura do documento, insere suas anotações junto à trechos do documento e elas serão lidas posteriormente por outros co-autores. Desta

maneira, elas servem como um mecanismo de comunicação entre usuários.

As anotações podem ser de dois tipos: *comentários*, que podem ser textuais ou vocais e estão associados a trechos do documento; e *substituições*, que correspondem a sugestões de possíveis alterações no documento. Em outras palavras, comentários são uma forma de texto em um meta-nível, ou seja, texto sobre o texto original, enquanto que substituições são segmentos de texto que podem substituir o texto original.

Uma questão importante em qualquer sistema cooperativo refere-se à percepção de um indivíduo sobre o trabalho do restante do grupo. Em outras palavras, um co-autor precisa, de alguma forma, estar ciente do trabalho dos outros co-autores, para que ele possa executar corretamente o seu próprio trabalho. Em sistemas assíncronos pode-se afirmar que a informação sobre a percepção consiste em identificar as ações que os outros usuários *executaram* no documento. Tal informação pode ser obtida através de mecanismos como correio eletrônico, arquivos de *log*, registro automático das ações dos outros autores e através da comparação de estados do artefato.

Em sistemas síncronos pode-se afirmar que a percepção consiste em identificar as ações que os outros usuários *estão executando* em um determinado momento. Neste caso, a primeira informação necessária consiste na identificação dos usuários correntes da sessão que é feito através da apresentação dos nomes e/ou fotos dos usuários presentes na sessão. Os mecanismos mais utilizados nestes sistemas são recursos de interface com usuário, como por exemplo WYSIWIS, telepointers, etc.

A interface dos sistemas cooperativos deve encorajar a cooperação e permitir que todos os co-autores percebam a presença e a atividade dos outros co-autores de forma *não-intrusiva*. Isto é diferente dos sistemas multi-usuários que dão a ilusão que o usuário está trabalhando sozinho. Para isso é necessária uma mudança de perspectiva [EGR91]. Novas soluções devem ser criadas para os problemas que aparecem, tanto em editores síncronos quanto assíncronos. Em editores síncronos, a forma como as informações compartilhadas são apresentadas nas interfaces das instâncias ativas do sistema é denominada *acoplamento da interface*. Bentley *et al.*[BRSS94] define três tipos de acoplamento: *interfaces fortemente acopladas*, cada usuário compartilha as mesmas informações na mesma forma de apresentação, ou seja, todos os usuários possuem a mesma visão do documento. *Interfaces com acoplamento médio*, cada usuário compartilha as mesmas informações, mas a forma de apresentação é diferente. E, finalmente, *interfaces sem acoplamento*, as informações apresentadas em cada interface são diferentes, ou seja, um co-autor pode estar visualizando uma seção do documento, enquanto que outro co-autor está visualizando uma seção diferente. Em editores assíncronos, existe apenas um usuário utilizando o editor a cada momento, logo o projeto de interfaces destes editores refere-se à apresentação das alterações anteriormente efetuadas pelos outros autores. Isto é feito com a utilização de cores e/ou ícones para identificar as partes do objeto que foram modificadas.

Capítulo 3

Frameworks, Padrões de Projeto e Metapadrões

Neste capítulo são descritos os conceitos básicos que norteiam o desenvolvimento desta dissertação. Na seção 3.1 é descrito o conceito de framework, as principais classificações existentes na literatura, as formas de reutilização que eles oferecem e as vantagens em utilizá-los. Seção 3.2 apresenta o conceito de padrões de projeto, suas vantagens e as abordagens existentes na literatura que foram utilizadas neste trabalho. Seção 3.3 descreve o conceito de metapadrões que são utilizados na abordagem de projeto orientado a pontos adaptáveis[Pre94a] para a construção de frameworks. Esta abordagem e a abordagem de projeto dirigido a exemplos[Joh93] são apresentadas na seção 3.5 que finaliza este capítulo.

3.1 Frameworks

Um *framework* é um projeto genérico em um domínio que pode ser adaptado à aplicações específicas, servindo como um molde para a construção de aplicações. Isto é feito através da especificação das classes e das colaborações entre elas. Segundo Orfaly[Orf95]:

> "Um framework é o projeto de um conjunto de objetos que colaboram para pôr em prática um conjunto de responsabilidades. Frameworks são um modo de reutilizar projetos de alto nível."

Em outras palavras, um framework é um projeto orientado a objetos abstrato que pode ser adaptado segundo as necessidades da aplicação[WBJ90]. Deste modo, um framework funciona como um molde para a construção de aplicações ou subsistemas, dentro de um domínio[CP95]. Todas as aplicações construídas a partir de um mesmo framework apresentam a mesma estrutura, diferenciando-se em seu comportamento, que é dependente da aplicação. Isto torna as aplicações desenvolvidas a partir do framework mais fáceis

de manter e mais consistentes para os usuários, que não precisam aprender diferentes aplicações.

Um framework pode fornecer um grau de reutilização de até 80%[Fir93], pois além de fornecer reutilização de código, que é oferecido por classes e objetos, ele também oferece reutilização de projeto, liberando o desenvolvedor dos aspectos comuns daquele domínio de aplicação. A reutilização de projeto permite uma reutilização em larga escala, que é uma condição necessária para o aumento de produtividade na atividade de desenvolvimento de software. Além disso, a reutilização de projeto é mais importante que a reutilização de código, porque o projeto é mais difícil de criar e recriar do que o código[JF88].

Neste ponto, é importante diferenciar um framework de bibliotecas de classes orientadas a objetos e bibliotecas de procedimentos. Um framework corresponde a um projeto de alto nível, consistindo de classes abstratas e concretas que especificam uma arquitetura para aplicações. Uma biblioteca de classes orientada a objetos é apenas um conjunto de classes, não necessariamente relacionadas, que fornece um conjunto de serviços disponibilizado através da interface pública de suas classes. De maneira similar, uma biblioteca de procedimentos fornece uma série de serviços através de chamadas às suas rotinas. Nenhum dos dois tipos de biblioteca permite a reutilização de projeto.

O modo como os frameworks operam também é diferente de outras abordagens. Em um framework bem-projetado, o desenvolvedor especifica somente o código de métodos de classes específicas que corresponderão a sua aplicação. O próprio framework irá se encarregar de chamar este código quando for necessário. Em bibliotecas de classes orientadas a objeto ou procedurais, além do código da aplicação propriamente dita, o desenvolvedor deve especificar o fluxo de execução, a estrutura do programa, etc. Figura 3.1 apresenta uma visão destas duas abordagens. Esta é uma diferença básica entre frameworks e outras abordagens. Tabela 3.1 adaptada de [Orf95] apresenta outras diferenças.

3.1.1 Classificação

O primeiro framework amplamente utilizado foi o framework para interfaces com usuário do *Smalltalk* chamado MVC (*Model/View/Controller*) [JF88]. Outro framework amplamente conhecido é o framework para interfaces com usuário *MacApp*, projetado especificamente para implementar aplicações no ambiente *Macintosh*.

Apesar da sua utilização bastante difundida na área de interfaces com usuários, os frameworks não são limitados a esta categoria de aplicações. Eles podem ser aplicados a qualquer área do projeto de software. Como por exemplo, o framework CHOICE[CIM92] para sistemas operacionais, o framework desenvolvido por Huni et al.[HJE95] para protocolos de redes de computadores, etc.

Firesmith[Fir93] sugere uma classificação que se relaciona ao domínio da aplicação a ser desenvolvida utilizando o framework:

Característica	Frameworks	Bibl. OO	Bibl. Procedurais
Modelo da Aplicação	Os frameworks são a aplicação. Os frameworks tratam todo o fluxo de controle.	É necessário criar o fluxo de controle da aplicação e a"cola" que liga as diferentes classes.	É necessário criar o fluxo de controle da aplicação e a lógica que invoca a biblioteca.
Estrutura da Aplicação	Múltiplos frameworks cooperando.	Uma única aplicação consistindo de várias bibliotecas de classes.	Um única aplicação "linkada" às bibliotecas.
Obtenção dos Serviços	Os frameworks são a aplicação.	Pela herança a partir da biblioteca de classes.	Pelas chamadas a biblioteca.
Customização dos Serviços	O framework chama sua aplicação. Pode-se herdar partes do framework.	Pela criação de novas subclasses.	Pela escrita de novo código e chamadas a a biblioteca.
Granularidade do Controle	Média. Pode-se herdar somente algumas partes do framework.	Alta. Pode-se herdar qualquer classe da biblioteca.	Alta. Pode-se escrever qualquer coisa a partir do zero.
O quanto é necessário escrever (código)	Muito pouco.	Regular.	Muito.
Custos de Manutenção	Baixo.	Regular.	Alto.
Redução de Complexidade	Alta. Escreve-se poucos trechos de código. Os frameworks chamam este código quando necessário.	Baixa. É necessário identificar classes, desenvolver o programa e integrar as diferentes bibliotecas de classes.	Baixa. É necessário desenvolver o programa inteiro e entender como as bibliotecas trabalham juntas.
Tempo de Desenvolvimento de Novas Aplicações	Baixo.	Regular. Dependendo da classe a ser reutilizada.	Alto.
Reutilização de Código	Muito Alta.	Alta. As classes são reutilizadas.	Regular.Algumas funções são reutilizadas.

Tabela 3.1: Quadro Comparativo entre Frameworks, Bibliotecas de Classes e Bibliotecas Procedurais[Orf95]

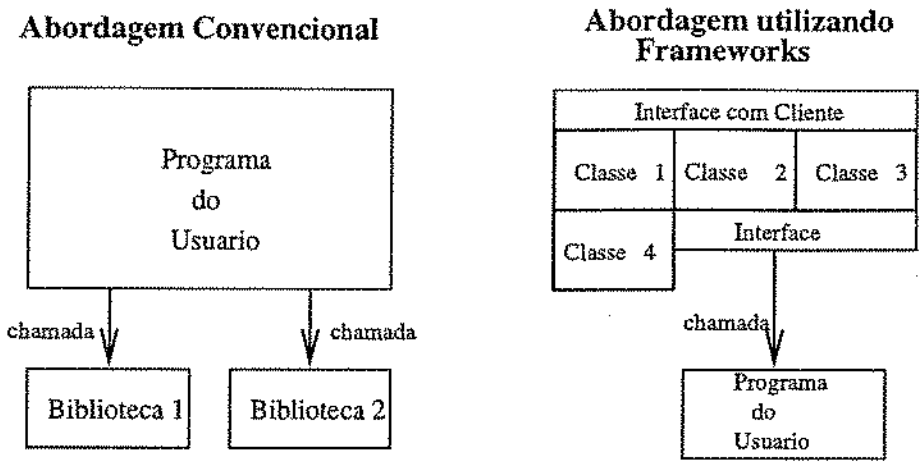


Figura 3.1: Abordagem convencional *vs.* Abordagem utilizando Frameworks

- Frameworks de Domínios: capturam classes, padrões, mecanismos e cenários indispensáveis em um domínio de aplicação particular, como por exemplo telecomunicações, automação industrial, etc.
- Frameworks para Interfaces com Usuário: fornecem classes de objetos para a construção de interfaces gráficas, o que permite que as aplicações desenvolvidas a partir dele possuam o mesmo *look and feel.*
- Frameworks para Banco de Dados: fornecem independência a partir de/e interoperabilidade com bases de objetos, bancos de dados relacionais, etc.
- Frameworks para Ambientes: fornecem independência a partir de/e portabilidade entre múltiplos sistemas operacionais e plataformas de hardware.

Os frameworks não precisam ser orientados a objetos, neste caso teremos:

- Frameworks Procedurais: são os frameworks que não são baseados em conceitos provenientes do paradigma orientado a objetos. A necessidade destes tipos de frameworks provém da quantidade e importância do código já produzido desta forma[Orf95].
- Frameworks Orientados a Objetos: são construídos utilizando o paradigma orientado a objetos. As próprias definições apresentadas neste trabalho referem-se diretamente a conceitos orientados a objetos, evidenciando assim uma tendência na construção e adoção destes tipos de frameworks. Durante o restante deste trabalho utilizaremos simplificadamente a palavra framework para representar frameworks orientados a objetos.

3.1.2 Vantagens da utilização de Frameworks

A utilização de frameworks fornece uma série de vantagens à atividade de desenvolvimento de software, como por exemplo:

- O desenvolvimento de novas aplicações torna-se mais rápido e mais barato, uma vez que utiliza componentes pré-fabricados e pré-testados. O desenvolvedor não precisa descobrir novas classes, projetar interfaces, etc. Basicamente, ele somente necessita reescrever alguns métodos específicos de determinadas classes. Além disso, a estrutura do programa já está especificada e o fluxo de execução definido.

- Os frameworks permitem a reutilização de código e projeto. Normalmente, o reuso de código é obtido através do polimorfismo e/ou herança. O reuso do projeto é obtido a partir do próprio framework, uma vez que ele foi projetado como uma solução genérica para uma determinada categoria de problemas.

- A manutenção é reduzida. Os frameworks fornecem a maior parte do código necessário às aplicações, tornando os custos de manutenção menores. Devido a herança, quando um erro em um framework é corrigido, ou uma característica adicionada, os benefícios são imediatamente estendidos às novas classes[Orf95].

- Possibilidade de desenvolvimento de aplicações mais complexas. A criação de frameworks baseados em outros frameworks permite o desenvolvimento de aplicações cada vez mais complexas. Como por exemplo, o framework ET++ SwapsManager[EG92] que é um framework para aplicações comerciais desenvolvido a partir do framework ET++[WGM88].

3.1.3 Instanciação de Aplicações a partir de Frameworks

Um framework serve como um molde para a construção de aplicações dentro de um domínio. Assim, diferentes aplicações podem ser instanciadas a partir deles. Segundo Johnson e Foote[JF88], esta instanciação pode ser feita de duas formas:

- **Frameworks caixa-branca**

Estes frameworks também são chamados de frameworks dirigidos à arquitetura. O comportamento de uma aplicação específica construída a partir deste framework, frequentemente, é especificado através da redefinição de subclasses e métodos de uma ou mais classes. Cada método a ser adicionado precisa manter as convenções internas das superclasses, tornando-se necessário o conhecimento de seu projeto e de sua implementação, o que torna sua utilização mais difícil. Resumidamente, pode-se dizer que o framework é

reutilizado através da especialização de seus componentes. Esta forma de reutilização dos componentes do framework é chamada de reutilização baseada em herança.

Estes frameworks requerem uma documentação explicíta do conjunto de classes e seus respectivos métodos, que devem ser redefinidos pelo usuário para que ele possa utilizá-lo. Por exemplo, para a criação de um editor de hiper-documentos a partir do framework ET++[WGM88, WGM89], o usuário necessita obrigatoriamente gerar uma subclasse da classe *Application*, a qual redefinirá o método *doMakeManager()*, e uma subclasse da classe *Document* que conterá a definição do hiperdocumento[Pre94a, 201–222]. A documentação de frameworks é discutida na seção 3.1.4.

• **Frameworks caixa-preta**

Estes frameworks também são chamados de frameworks dirigidos a dados. Sua customização é feita através da disponibilização de um conjunto de componentes que fornecem o comportamento de aplicações específicas. Cada um destes componentes precisa entender um protocolo particular. Todos, ou a maioria deles, são fornecidos por uma biblioteca de componentes. A interface entre os componentes pode ser definida por um protocolo, logo o usuário só precisa entender a interface externa dos componentes. Neste caso, o framework é reutilizado através da instanciação de seus componentes. Estes frameworks são mais fáceis de usar, no entanto são menos flexíveis que os frameworks caixa-branca[JF88, Tal95, FS97a]. Esta forma de reutilização dos componentes do framework é chamada de reutilização baseada em composição.

Na verdade, geralmente um framework possui características dos frameworks caixa-branca e dos caixa-preta. Ou seja, alguns de seus componentes são reutilizados através de herança, enquanto outros são reutilizados através de composição. Até mesmo durante a construção do framework, dependendo da situação, podem ser criados componentes caixa-branca e caixa-preta. Por isso, costuma-se dizer que esta classificação corresponde a um espectro contínuo, onde os pontos extremos correspondem às definições apresentadas. De agora em diante, o termo framework caixa-branca será utilizado para indicar que este framework baseia-se mais frequentemente em herança do que em composição como mecanismo de reutilização. O inverso é válido para o termo framework caixa-preta.

Normalmente, um framework recém-desenvolvido é do tipo caixa-branca, pois o domínio da aplicação ainda não é suficientemente compreendido tal que seja possível parametrizar o seu comportamento. A partir de testes do framework e a consequente reorganização de sua estrutura de classes, o domínio da aplicação passa a ser bem compreendido e os aspectos que devem ser flexíveis no framework são definidos. Neste momento, o framework pode ter seu comportamento customizado através de diferentes combinações de classes,

ou seja, ele se torna um framework caixa-preta.

• **Vantagens e Desvantagens**

A vantagem da reutilização baseada em herança é que ela permite que novos componentes sejam construídos com pouco esforço através da criação de subclasses. As classes do framework implementam a maior parte das funcionalidades desejadas, o que permite propagar automaticamente às subclasses as mudanças realizadas nas classes do framework[Orf95]. No entanto, isto torna-se uma desvantagem se considerarmos que as subclasses podem precisar ser modificadas caso as superclasses o sejam. Além disso, uma outra restrição desta abordagem é que os métodos são determinados estaticamente, de tal forma que não se pode modificar dinamicamente os serviços executados.

A reutilização baseada em composição oferece a vantagem de permitir a adaptação dinâmica das aplicações. As classes implementam funções específicas que são utilizadas através de interfaces bem definidas. Comportamentos complexos são criados através da composição de objetos, sendo que estes objetos podem manter referências para outros objetos. Através destas referências, um objeto pode delegar a responsabilidade de executar um dado serviço a outro objeto. Este objeto, que realizará efetivamente o serviço solicitado, depende da configuração de instâncias criada.

A reutilização baseada em composição é mais desejável que a baseada em herança, pois permite maior flexibilidade ao framework. Entretanto, isto requer um conjunto de classes que implemente *todas* as funcionalidades do domínio. Como isto é muito difícil de ser obtido, frequentemente surge a necessidade de se desenvolver novas classes que implementem serviços não disponíveis. Estas classes são implementadas reutilizando as funcionalidades de classes existentes através da reutilização baseada em herança. Deste modo, composição e herança são técnicas complementares que, quando utilizadas adequadamente, contribuem para o desenvolvimento de sistemas flexíveis e reutilizáveis[Tal95, Cam97].

3.1.4 Documentação de Frameworks

Se a tarefa de documentação de uma aplicação convencional é considerada importante, ela torna-se especialmente importante quando a aplicação a ser desenvolvida é um framework. Afinal, se os outros desenvolvedores não entenderem o framework, eles não o utilizarão[Tal95]. Mais do que isso, a complexidade de se documentar um framework é maior que a de documentar uma aplicação convencional, devido à natureza abstrata dos frameworks.

A documentação do framework tem dois propósitos distintos. Primeiro, comunicar informações relativas ao projeto do framework e outras informações relacionadas ao seu desenvolvimento. Estas informações são importantes porque tornam claro para o usuário as restrições e regras que precisam ser seguidas para que o framework possa ser utilizado.

Como exemplo tem-se o número de objetos a serem criados, a ordem de criação destes objetos, etc. Estas regras não farão sentido para o usuário, se ele não tiver acesso à estas informações, logo ele poderá utilizar o framework de maneira errada. Além disso, o usuário tem de entender o projeto do framework para avaliar se ele pode ou não ser utilizado na construção da aplicação desejada, e para entender como ele deve ser adaptado para que isto seja possível.

Finalmente, existe a necessidade de documentar os passos que devem ser seguidos pelos usuários para a correta utilização do framework. Por exemplo, se o framework é do tipo caixa-branca, a documentação do mesmo deve apresentar as classes e seus métodos que devem ser redefinidos para que o framework possa ser utilizado. Neste caso, a documentação deve descrever como utilizar o framework sem entrar em detalhes de como é seu funcionamento interno.

As notações desenvolvidas para descrever aplicações orientadas a objetos específicas não são adequadas para a tarefa de documentação de frameworks, pois elas não descrevem adequadamente os fluxos de controle genéricos implementados pelas classes abstratas do framework. Assim, elas não oferecem mecanismos adequados para ressaltar aspectos próprios dos frameworks, como por exemplo, as partes que precisam ser redefinidas. Devido a estas limitações, vários autores tem proposto diferentes técnicas especialmente projetadas para documentar frameworks, como por exemplo, *cookbooks*[Cam97], contratos[HHG90, Hol92][1], padrões de projeto[GHJV93, GHJV94] (seção 3.2) e meta-padrões[Pre94b, Pre94a] (seção 3.3).

Cookbooks são uma descrição de um conjunto de receitas[2], onde cada uma delas descreve como utilizar uma parte do framework.

A técnica de contratos baseia-se em uma linguagem semi-formal que descreve os mecanismos de colaboração entre as classes do framework. [CI92] utiliza esta técnica para descrever parte do framework de um sistema operacional.

Padrões de projeto são utilizados como mecanismo de documentação de duas maneiras distintas: (i) como mecanismo para descrever a arquitetura do framework[BJ94, HJE95] o que permite que o usuário entenda o projeto e consequentemente entenda o funcionamento e as restrições; e (ii) como mecanismo para descrever a utilização do framework[Joh92].

A abordagem de meta-padrões é utilizada como um mecanismo de auxílio no processo de construção de framework, pois adiciona flexibilidade ao projeto do mesmo. Ao mesmo tempo, ela também pode ser usada como um mecanismo de documentação, pois descreve como utilizar o framework.

[1] Em Inglês: *contracts*.
[2] Em Inglês: *recipes*.

3.2 Padrões de Projeto

Padrões de projeto[3] são um meio para representar, registrar e reutilizar micro-arquiteturas de projeto repetitivas, bem como a experiência acumulada por projetistas durante o desenvolvimento de software. Além desta definição, existem outras encontradas na literatura:

- Padrões de projeto constituem um conjunto de regras descrevendo como realizar determinadas tarefas no desenvolvimento de software[Pre94b].
- Padrões de projeto são um modo de capturar experiências de projeto de modo que outras pessoas possam utilizá-las[GHJV93].

Apesar das diferentes definições, existe um consenso de que padrões de projeto são, basicamente, uma parcela de experiência destilada, descrita como uma solução para um problema em uma dada situação[Cam96]. Ou seja, um padrão de projeto é uma forma de capturar uma solução para um problema que se repete em contextos similares.

O termo padrão de projeto surgiu na década de 70 com o trabalho do arquiteto Christopher Alexander, que encontrou temas recorrentes na arquitetura e os capturou em descrições e instruções que ele chamou de padrões. Para Alexander, cada padrão descreve um problema que ocorre repetidamente em um ambiente, ele descreve o núcleo da solução do problema, de modo que você pode usar esta solução um milhão de vezes.

Durante a década de 90, os projetistas de software descobriram a idéia de Alexander e tem-na aplicado no desenvolvimento de software. A adoção da idéia de problema-solução-contexto aplicada a padrões de projeto em software, obriga que um padrão apresente quatro elementos essenciais[GHJV94]:

1. O Nome do padrão: usado para descrever um problema, sua solução e suas conseqüências. Deste modo, o vocabulário dos projetistas é ampliado, facilitando o desenvolvimento de software, uma vez que existirá um vocabulário comum que permitirá a troca de experiência entre projetistas através da comunicação das decisões de projeto, e facilitará a documentação do sistema.
2. O Problema: descreve quando aplicar o padrão. Explica o problema em questão e o contexto. Pode descrever problemas específicos ou genéricos. Normalmente, pelo menos dois exemplos são apresentados para cada padrão.
3. A Solução: descreve os elementos que formam o padrão, seus relacionamentos, responsabilidades e documentação. A solução é apresentada para um problema genérico e uma estrutura geral.

[3]Em Inglês: *Design Patterns*.

4. As Conseqüências: são os resultados e decisões que a utilização daquele padrão acarretam. A descrição destas conseqüências é vital para uma avaliação dos custos e benefícios de um padrão e, consequentemente, para auxiliar a decisão sobre sua utilização. As conseqüências de um padrão incluem, por exemplo, seu impacto sobre a flexibilidade, portabilidade e expansibilidade do sistema.

3.2.1 Vantagens dos Padrões de Projeto

A utilização de padrões de projeto apresenta as seguintes vantagens[GHJV93, GHJV94, Pre94b, Pre94a, Sch95b, SS95]:

- Os padrões de projeto formam um vocabulário comum para os projetistas se comunicarem e para a documentação e exploração das alternativas de projeto;
- Reduzem a complexidade do sistema, pois nomeiam e identificam abstrações que estão em um nível de abstração acima das classes e instâncias;
- Permitem larga reutilização de arquiteturas de software, mesmo que a reutilização de algoritmos, implementações, interfaces ou projetos detalhados não seja permitida;
- Constituem uma base de experiência reutilizável para a construção de software. Eles fornecem um modo de reutilizar o conhecimento *(expertise)* de projetistas experientes, e com isso facilitam o treinamento de novos desenvolvedores;
- Reduzem o tempo de aprendizado de bibliotecas de classes e frameworks. Uma vez que um usuário tenha aprendido uma biblioteca, ele pode reutilizar esta experiência para aprender novas bibliotecas; e
- Atuam como "blocos de construção" que podem ser utilizados para criar aplicações mais complexas.

3.2.2 Categorias de Padrões

Padrões de projeto possuem diferentes níveis de abstração e podem auxiliar em todas as fases do ciclo de vida de desenvolvimento. Segundo Buschmann[BMR+96], os padrões podem ser agrupados em três categorias, a saber:

- Padrões de Arquitetura: expressam um esquema da organização global da estrutura do sistema. Eles provêem um conjunto de subsistemas pré-definidos, especificando os relacionamentos entre eles e estabelecendo regras para esses relacionamentos.

- Padrões de Projeto: fornecem um esquema para refinar os subsistemas ou componentes do sistema de software. Esses padrões possuem um grau de granulosidade considerado médio e são independentes de linguagem de programação. Padrões de projeto, geralmente, correspondem a uma abstração de duas, três ou um pequeno número de classes, que é útil repetidamente no desenvolvimento de software orientado a objetos[Coa92].

- Idiomas: são os padrões de nível mais baixo, específicos para um determinada linguagem de programação. Eles descrevem como implementar aspectos particulares de componentes e dos relacionamentos entre eles usando características específicas da linguagem alvo.

Além desta classificação, os padrões podem ser categorizados segundo o domínio no qual se aplicam. Assim, podem existir padrões genéricos, que podem ser aplicados em diferentes domínios; mas também podem existir padrões específicos para determinados domínios como por exemplo, tolerância a falhas, criptografia, telecomunicações, etc.

As próximas seções apresentarão de maneira breve os padrões de projeto e idiomas utilizados no desenvolvimento do framework ABCDE.

3.2.3 Idiomas

Idiomas são os padrões de nível mais baixo, específicos para um determinada linguagem de programação. Eles revelam modos úteis de combinar conceitos básicos de uma linguagem e ensinam como evitar armadilhas e livrar-se das deficiências das linguagens orientadas a objetos. Além disso, formam uma base para a padronização da nomenclatura e da estrutura do código fonte. Sob este ponto de vista, eles agem como diretrizes de projeto ou, ainda, como convenções para o modo de atribuir nomes.

Existem diversos idiomas na literatura para diferentes linguagens de programação. Nesta dissertação apresentaremos os idiomas identificados por Coplien[Cop91] e Wolgang Pree[Pre94a][4] que são descritos a seguir:

A Abordagem de Coplien

Coplien[Cop91] sugere uma abordagem para a declaração de variáveis e geração de objetos utilizando a linguagem C++[Str91], segundo a qual uma classe adere a forma canônica e ortodoxa se a definição da classe contém:

- Um construtor *default*: a linguagem C++ não fornece a sintaxe necessária para a passagem de parâmetros para os métodos construtores de um vetor. Ao invés disso,

[4] O estudo apresentado nesta seção é baseado no estudo efetuado por [Pre94a, 72–79].

ela executa o método construtor *default* para cada elemento do vetor. Para evitar problemas na declaração de vetores Coplien recomenda que um construtor *default* seja sempre declarado.

- Um construtor de atribuição e cópia: esta recomendação é feita para evitar problemas de alocação de memória. O problema consiste em dadas duas variáveis estáticas que referenciam áreas dinamicamente alocadas, ao se efetuar a atribuição (cópia) não se deve permitir que áreas de memória inutilizadas continuem ativas. Ou seja, ao atribuir uma variável a a uma variável b, a área não mais referenciada por b deve ser desalocada, desde que ela não seja utilizada por outras variáveis.

- Definição de um *destructor*: o *destructor* teria por objetivo liberar posições de memória alocadas pela variável, desde que não utilizadas por outras variáveis.

Segundo Pree[Pre94a] a abordagem de Coplien aplica-se a variáveis estáticas, sendo que as duas recomendações iniciais tornam-se desnecessárias ao se utilizar variáveis alocadas dinamicamente.

A Abordagem de Pree

Pree[Pre94a] apresenta uma outra abordagem que se refere a convenções de nomes a serem dados a variáveis, constantes e métodos. O trabalho de Pree é baseado nas convenções descritas no framework ET++[WGM88, WGM89]. A utilização de convenções de nomenclatura em frameworks é importante pois facilita o aprendizado e utilização do mesmo.

A abordagem consiste de convenções para:

- Nomes de classe e nomes de métodos: devem iniciar com letras maiúsculas seguidos de letras minúsculas. Variáveis locais devem iniciar com letras minúsculas. Se um nome consiste de várias palavras, a segunda e as palavras subsequentes iniciam com letras maiúsculas, como por exemplo *DoLeftButtonDownCommand(...)*.

- Variáveis globais: devem iniciar com o prefixo *g*, como: *gFileDialog, gApplication.*

- Constantes: devem iniciar com o prefixo *c*, como: *cldNone*. Constantes que identificam comandos pré-definidos no framework iniciam com *c* seguidas de letras maiúsculas: *cSAVEAS, cPRINT*. Constantes que pertencem a tipos enumerados iniciam com o prefixo *e*: *eVObjHLeft, eVObjHCenter*.

- Métodos: quando definem o valor de uma variável instanciada devem iniciar com *Set*, como *SetOrigin(...)*. Métodos que retornam tais valores devem iniciar com *Get*,

como *GetOrigin(...)*. Métodos abstratos que devem ser redefinidos em subclasses devem iniciar com *Do*, como *DoKeyCommand(...)*. Métodos implementando a criação dinâmica de objetos devem iniciar com *Make* ou *Create*, como *DoMakeManager()*. Métodos que desenham objetos na tela devem iniciar com *Gr*, como *GrPaintRect()*.

3.2.4 O Catálogo de Padrões de Projeto

O chamado catálogo de padrões de projeto corresponde à abordagem proposta por Gamma *et al.*[GHJV93, GHJV94]. Este trabalho baseou-se, principalmente, na experiência obtida pelo autor durante seu trabalho com o framework ET++, no qual estruturas de projeto elegantes, flexíveis e bem-estruturadas foram identificadas. Gamma também identificou estas estruturas em outros frameworks. A partir disso, com a adoção da idéia de Christopher Alexander de problema-solução-contexto aplicada a padrões de projeto, ele expôs um catálogo com vinte e três padrões publicado em 1994.

Os padrões de projeto propostos por Gamma et al. "(...) não são projetos como listas encadeadas e tabelas de *hashing* que podem ser codificadas em classes e assim reusadas. Eles não são projetos complexos de domínios específicos para uma aplicação inteira ou subsistemas (...) eles são descrições de objetos e classes que se comunicam e que são customizadas para resolver problemas gerais de projeto em um contexto particular (...)".

Um padrão de projeto pode ser considerado como uma micro-arquitetura. Como uma arquitetura uma vez que pode servir como uma planta *(blueprint)* que permite várias implementações. Ele é micro no sentido de definir algo menor que uma aplicação completa[GHJV94].

Descrição dos Padrões de Projeto

Para Gamma et al. são necessárias mais que notações gráficas para descrever os padrões de projeto, pois é preciso recordar decisões, alternativas e decisões que levaram ao padrão, e isto não pode ser feito apenas com a utilização de notações gráficas. Gamma utiliza um formato dividido em seções, o que torna seus padrões fáceis de aprender, comparar e usar. As seções são as seguintes:

- Nome do Padrão e Classificação: o nome do padrão e sua classificação são apresentados.
- Intenção: esta seção deve responder às seguintes questões: O que o padrão faz? Qual sua razão e sua intenção? Que características particulares do problema o padrão resolve?
- Também conhecido como: outros nomes conhecidos para o padrão.

- Motivação: descreve um cenário com um problema prático onde o padrão é utilizado para resolver o problema.
- Aplicabilidade: esta seção deve responder à seguinte questão: em que situação o padrão pode ser aplicado?
- Estrutura: descreve a representação gráfica das classes do padrão através de uma notação baseada na OMT[RBP+91], além de diagramas de interação.
- Participantes: descreve as classes e objetos que participam do padrão, além de suas responsabilidades.
- Colaborações: descreve como os participantes colaboram para atingir as suas responsabilidades.
- Conseqüências: apresenta os resultados e decisões que devem ser tomadas para a implementação do padrão de projeto.
- Trechos de Código: fragmentos de código C++ ou Smalltalk que ilustram como o padrão pode ser implementado.
- Usos Conhecidos: exemplos do padrão encontrados em sistemas reais. No mínimo dois exemplos de diferentes domínio são apresentados.
- Padrões Relacionados: descreve outros padrões relacionados que podem substituir ou complementar o padrão que está sendo descrito.

Além disso, Gamma propõe uma classificação de seus padrões visando facilitar sua utilização e aprendizado. Esta classificação é apresentada detalhadamente em [GHJV94].

Exemplo de um Padrão de Projeto

O padrão de projeto Estado[GHJV94, 305], descrito na Figura 3.2, apresenta uma solução genérica para o problema de modelagem de objetos que apresentam diferentes estados "lógicos" e também comportamentos variados dependendo do estado lógico corrente. O padrão define uma hierarquia de estados paralela à hierarquia de classes e o objeto da aplicação delega as operações dependentes de estado para um objeto representando o seu estado corrente. Em tempo de execução o objeto da aplicação pode mudar seu estado lógico, implicando também numa alteração em seu comportamento.

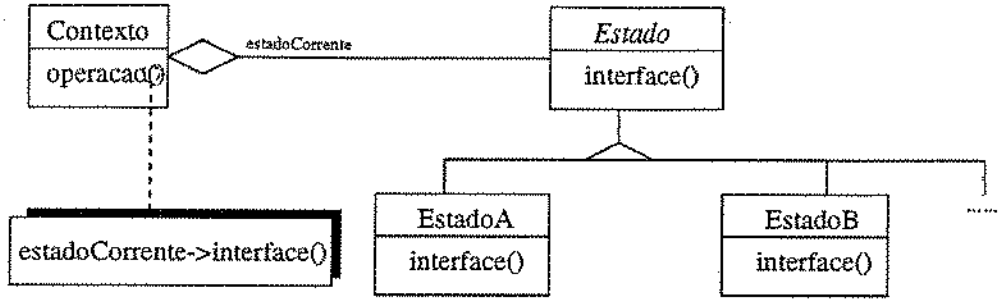


Figura 3.2: Exemplo do padrão Estado[GHVJ94, 305].

3.3 Metapadrões

Os metapadrões[Pre94b, Pre94a][5] podem ser utilizados para construir frameworks independentes de domínio. Eles descrevem os padrões de projeto em um nível de abstração mais alto, por isso são chamados metapadrões. Segundo Pree[Pre94a] eles podem ser aplicados para classificar e descrever qualquer padrão de projeto.

Os metapadrões baseiam-se na utilização de métodos *template* e *hook*. Um método *template* é um método complexo, que faz chamadas a métodos mais simples, que são os métodos *hook*. A especificação destes métodos deve ser feita utilizando-se o princípio definido por Weinand et al.[WGM89]:

> " O comportamento que está espalhado por vários métodos em uma classe deve ser baseado em um conjunto mínimo de métodos dinâmicos que devem ser sobrecarregados. Isto permite que os clientes derivem subclasses a partir de uma classe existente, sobrecarregando apenas poucos métodos para adaptar o seu comportamento da classe."

Em outras palavras, um framework bem projetado deve conter um conjunto flexível de métodos *template* e *hook*. Métodos *template* implementam os pontos fixos[Pre94a][6] do framework, ou seja, às partes comuns das aplicações correlatas. Diferentemente, os métodos *hook* implementam os pontos adaptáveis[Pre94a][7] que são as partes que podem ser estendidas para cada aplicação específica.

O conceito de métodos *template* e *hook* é ilustrado na Figura 3.3. Chamadas de métodos são expressas por setas. O método $M1()$ é um método *template* que faz chamadas aos métodos *hook* $M2()$ e $M3()$. $M2()$ consiste em um ponto adaptável que deve ser preenchido, através de redefinição utilizando-se uma subclasse da classe B; e $M3()$ é um ponto adaptável que pode ou não ser substituído.

[5] Em Inglês: *Metapatterns*
[6] Em Inglês: *Frozen spots.*
[7] Em Inglês: *Hot spots.*

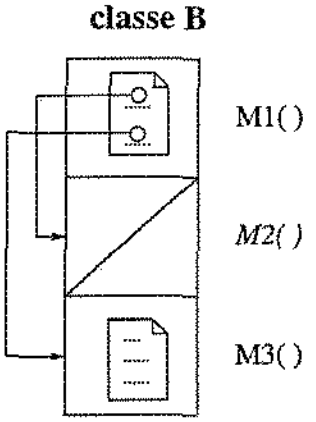


Figura 3.3: Um método *template* invocando seus métodos *hook*[Pre94a, 109].

Quando os métodos *template* e *hook* são unificados em uma única classe, o comportamento desta classe somente pode ser modificado através da criação de uma subclasse que irá redefinir os métodos *hook*. Entretanto, em algumas situações é necessário uma flexibilidade maior para permitir adaptações em tempo de execução. Neste caso, métodos *template* e *hook* são declarados em classes separadas, e a classe que contém o método *template* deve conhecer a classe que implementa o método *hook* para que as mensagens possam ser enviadas. Isto é feito através de um relacionamento de agregação entre os objetos das classes correspondentes. Figura 3.4 apresenta o metapadrão 1:1 Conexão[Pre94a, 124] que ilustra esta situação. O método T() implementa um método *template*, enquanto que o método H() implementa um método *hook*. A classe que implementa os métodos *hook* é chamada classe *hook* (H), enquanto que a classe que implementa os métodos *template* é chamada classe *template* (T). Em outras palavras, uma classe *hook* parametriza uma classe *template* [Pre94b].

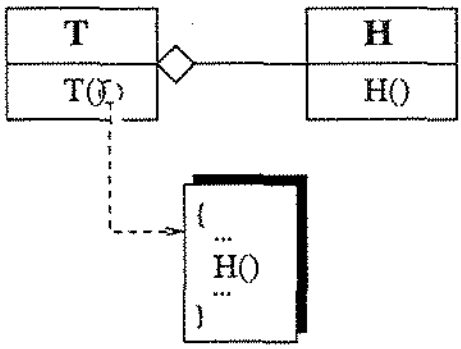


Figura 3.4: Metapadrão 1:1 Conexão[Pre94a, 124]

O exemplo a seguir ilustra a utilização de métodos *template* e *hook*. Em um framework para editores de diagramas, um método *template* seria o método que cria a *pallete* com

os elementos da notação de desenvolvimento de software que este editor apóia. Este método não contém a definição dos elementos da pallete, ela apenas é responsável por chamar outros métodos que contém esta definição. Estes métodos são os métodos *hook*, que contém a definição dos elementos da notação do editor. Assim, para que um usuário do framework modificasse a notação apoiada, bastaria que ele modificasse estes métodos *hook*. O próprio framework, através dos seu métodos *template* se encarregaria de chamar estes métodos quando fosse necessário. A grande dificuldade do projeto de frameworks consiste na identificação dos seus pontos adaptáveis. Idealmente, os usuários do framework somente deveriam ter que sobrecarregar os métodos *hook*, pois são eles que implementam os pontos adaptáveis.

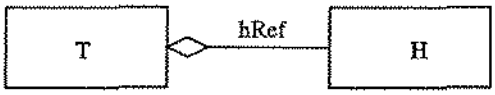
Pree define um conjunto fechado de sete metapadrões baseados nas noções de métodos *hook* e *template* (Figura 3.5). O metapadrão Unificação representa o caso especial no qual os métodos *hook* e *template* são unificados em uma única classe. Este conjunto é dito fechado, porque todas as estruturas baseadas em métodos *template* e *hook* requeridas para a construção de um framework independentes de domínio, podem ser derivadas a partir dos elementos deste conjunto.

Segundo Pree, a abordagem dos metapadrões complementa a abordagem de Gamma, pois os mesmos permitem a visualização dos pontos adaptáveis de um framework. Isto facilita o entendimento, e consequentemente utilização do framework. Cada metapadrão ocorre provavelmente várias vezes em um framework, logo os metapadrões podem ser vistos como um meio de capturar e classificar o projeto de um framework.

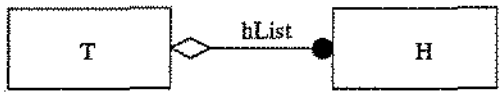
3.4 Comparação entre Padrões de Projeto e Metapadrões

A abordagem de padrões de projeto, assim como a de metapadrões, pode ser utilizada no projeto de um framework. As duas abordagens possuem o objetivo comum de tornar o framework flexível. Entretanto, se for feita uma comparação entre as duas abordagens pode-se ressaltar alguns aspectos importantes:

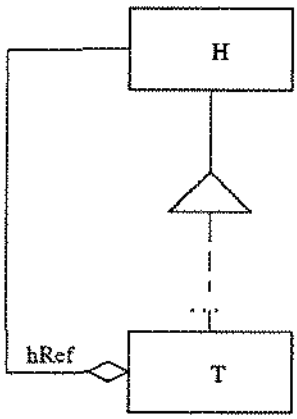
- Metapadrões podem contribuir para documentar o projeto de qualquer framework, independente de seu domínio[Pre94a].
- Metapadrões são mais abstratos que padrões de projeto. Um mesmo metapadrão pode ser utilizado no projeto de um framework na solução de diferentes problemas, enquanto que padrões de projeto são aplicados a apenas um único problema. Por outro lado, os padrões de projeto por serem mais concretos são mais fáceis de manusear e entender.



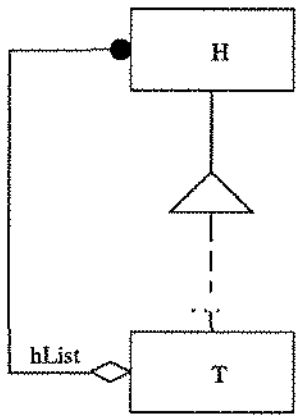
1:1 Conexão



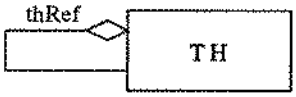
1:N Conexão



1:1 Conexão Recursiva

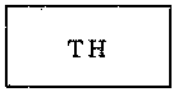


1:N Conexão Recursiva



1:1 Unificação Recursiva

1:N Unificação Recursiva



Unificação

Figura 3.5: Metapadrões

- Padrões de projeto provêem um guia mais concreto para compreensão dos pontos adaptáveis de um framework[Sch95a].
- Metapadrões são mais flexíveis que os padrões de projeto. Porém, esta vantagem é também uma desvantagem, uma vez que para problemas similares, soluções diferentes podem ser desenvolvidas[Sch96].
- O catálogo de padrões de projeto, como um complemento para as metodologias de análise e projeto orientado a objetos (como é visto por muitos autores), é insuficiente para apoiar o desenvolvimento de frameworks[Pre94a].
- O catálogo de padrões de projeto tem-se tornado cada vez mais completo[Sch96].

Apesar de ser uma abordagem mais concreta, o catálogo de padrões de projeto existente ainda não é suficiente para apoiar o efetivo desenvolvimento de frameworks. Isto ocorre porque ainda não existe um número suficiente de padrões de projeto para tornar flexível os diversos pontos adaptáveis de um framework. Por outro lado, os metapadrões são mais abstratos, porém independentes de domínio. Em outras palavras, cada uma das abordagens apresenta vantagens e desvantagens, sendo mais adequado utilizar uma combinação das duas conforme proposto por Schmid[Sch95a, Sch96].

3.5 Metodologias de Desenvolvimento de Frameworks

Assim como existem metodologias para o desenvolvimento de aplicações orientadas a objetos, como OMT[RBP+91], Booch[Boo94] e Fusion[CAB+94], também existem propostas de metodologias para o desenvolvimento de frameworks. Esta seção apresentará as duas principais metodologias encontradas na literatura: o projeto dirigido por exemplos de Johnson[Joh93] e o projeto orientado a pontos adaptáveis de Pree [Pre94b, Pre94a].

O objetivo de qualquer metodologia de desenvolvimento de frameworks é desenvolvê-lo tal que ele seja[Tal95]:

- Completo: um framework deve fornecer as características desejadas pelos clientes e uma implementação *default* sempre que possível. Além disso, ele deve fornecer classes concretas que derivem de classes abstratas do framework, assim como implementações *default* dos métodos destas classes, para que os clientes possam entender o framework. Isto permite que eles possam se concentrar nos aspectos específicos de suas aplicações.
- Flexível: um framework deve permitir que suas abstrações sejam usadas em diferentes contextos.

- Extensível: os clientes podem facilmente modificar e adicionar funcionalidades. O framework deve fornecer ganchos (*hooks*) aos clientes para que eles possam customizar o comportamento das aplicações através da criação de novas subclasses, ou através da combinação de instâncias de objetos.

- Compreensível: as interações entre o framework e o código do cliente devem ser claras e o framework bem-documentado. Deve-se seguir padronizações durante o projeto e a implementação visando um rápido aprendizado, e também fornecer aplicações desenvolvidas a partir do framework.

3.5.1 Projeto dirigido por Exemplos

Segundo Johnson[Joh93] o desenvolvimento de um framework é feito a partir de um processo de aprendizado sobre um determinado domínio. Este aprendizado ocorre a partir do estudo de aplicações previamente desenvolvidas ou a partir do desenvolvimento de novas aplicações. Como as pessoas pensam de forma concreta, a construção de um framework, um projeto abstrato, deve ser obtido a partir da generalização de aplicações concretas, que são as aplicações desenvolvidas anteriormente[SP96]. Os aspectos comuns das aplicações se transformam em classes abstratas, enquanto que os detalhes específicos de cada aplicação se transformam em classes concretas, especializações das abstratas.

O processo de desenvolvimento segundo o projeto dirigido por exemplos consiste nos seguintes passos:

1. Análise do domínio do problema:
   - aprender as abstrações previamente conhecidas;
   - coletar um número mínimo de quatro exemplos de programas que poderiam ser desenvolvidos a partir do framework; e
   - avaliar a importância de cada exemplo em relação ao domínio do framework.
2. Projetar uma hierarquia de abstrações (classes) que possa ser especializada, de modo a garantir a abrangência de todos os exemplos. A hierarquia de classes projetada corresponde ao framework. Neste passo Johnson recomenda a utilização de padrões de projeto como um mecanismo de melhoria da qualidade do projeto.
3. Testar o framework verificando se ele pode ser utilizado para instanciar os exemplos. Ou seja, deve-se implementar cada um dos exemplos utilizados no passo inicial utilizando o framework desenvolvido.

Esta sequência de passos pode ser considerada ideal[Joh93]. No entanto, ela não é seguida devido às seguintes limitações:

- A análise dos exemplos demanda tempo. Esta fase é responsável por grande parte do custo do framework.

- Na medida em que as aplicações antigas, os exemplos em potencial, estão funcionando corretamente, não existe incentivo para converte-las em novas aplicações (geradas a partir do framework).

- Gasta-se mais tempo na análise de algumas aplicações, as aplicações a serem desenvolvidas, em detrimento de outras, as aplicações já desenvolvidas.

Em função destas dificuldades, Johnson[Joh93] estabelece um procedimento economicamente mais adequado ao desenvolvimento de frameworks. A partir de duas novas aplicações que precisem ser desenvolvidas, deve-se desenvolver um framework em paralelo. Ou seja, devem-se criar três grupos de desenvolvimento: um grupo para cada uma das aplicações e um grupo para o framework, procurando maximizar a troca de informações entre os três grupos. Para que o framework seja desenvolvido corretamente, os grupos devem conter pessoas com experiência no domínio das aplicações, garantindo desta forma que informações de outras aplicações, as aplicações desenvolvidas anteriormente por estas pessoas, sejam utilizadas.

3.5.2 Projeto Orientado a Pontos Adaptáveis

Esta abordagem, desenvolvida por Pree[Pre94a, 228–231], consiste na identificação dos pontos fixos e dos pontos adaptáveis do domínio de uma aplicação. Os pontos fixos correspondem às partes comuns das aplicações correlatas, enquanto que os pontos adaptáveis são as partes que podem ser estendidas para cada aplicação específica, dando ao framework a capacidade de ser flexível e se moldar a diferentes aplicações. A flexibilidade do framework é obtida através da utilização dos metapadrões (seção 3.3).

A essência da abordagem de Pree consiste em identificar os pontos adaptáveis na estrutura de classes de um domínio, e a partir daí, construir um framework[SP96]. Figura 3.6 apresenta as etapas do projeto dirigido a pontos adaptáveis:

- A etapa inicial consiste na identificação e definição da estrutura de classes do framework. Esta etapa requer a utilização de uma metodologia de desenvolvimento de software. Neste trabalho a metodologia utilizada foi a OMT[RBP+91].

- A segunda etapa consiste na identificação dos pontos adaptáveis. É preciso identificar os aspectos que variam de aplicação para aplicação, assim como o grau de flexibilidade desejado.

- O projeto do framework consiste na modificação da estrutura definida inicialmente de modo a obter a flexibilidade desejada. Pree propõe a utilização de metapadrões nesta etapa.
- A etapa final consiste em um refinamento da estrutura do framework. Ao final, o framework é avaliado para verificar se os pontos adaptáveis oferecem o grau de flexibilidade desejado. Caso isto não ocorra, retorna-se a etapa de identificaçao de pontos adaptáveis.

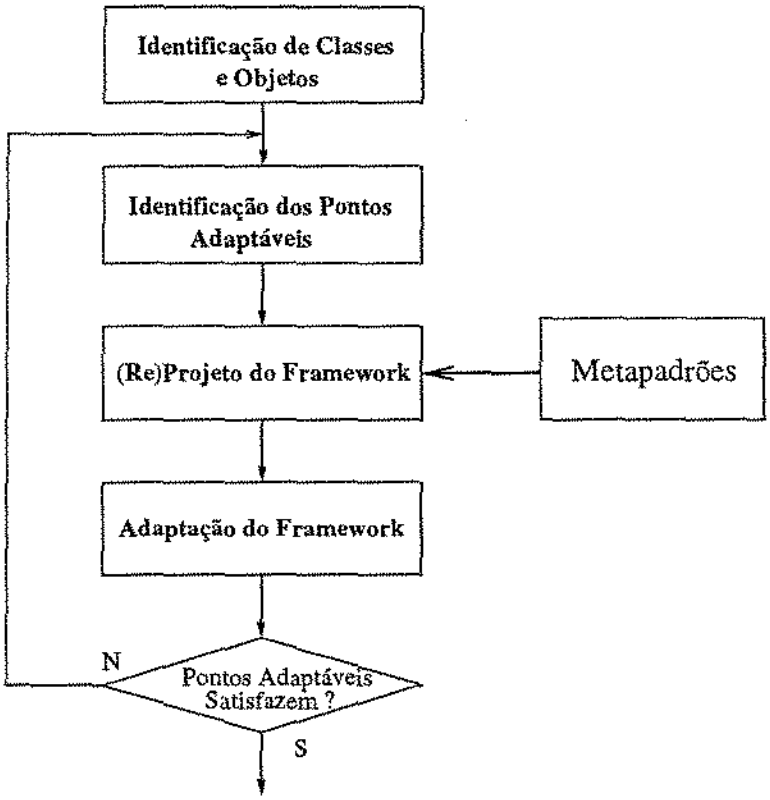


Figura 3.6: Projeto orientado a pontos adaptáveis[Pree94b, 230]

3.5.3 Comparação

Aspectos Comuns

A principal característica comum é a busca de informações do domínio do framework em aplicações previamente desenvolvidas. Isto é feito diretamente no projeto dirigido a exemplos, que se baseia fundamentalmente nestas aplicações; e indiretamente no projeto orientado a pontos adaptáveis, que recomenda a utilização de metodologias de análise e

projeto orientado a objetos. Estas metodologias geralmente recomendam a consulta de sistemas similares como fonte de informação.

Finalmente, o último aspecto comum consiste em aplicar testes ao framework. O teste do framework consiste no desenvolvimento de aplicações a partir dele. Pode-se desenvolver as próprias aplicações utilizadas como exemplo[Joh93]. Todas as metodologias recomendam esta fase para finalizar o processo de desenvolvimento.

Aspectos Particulares

O projeto dirigido a exemplos diferencia-se pela enfâse na análise de aplicações previamente desenvolvidas. Se não houver um número mínimo de exemplos pode-se até mesmo desenvolver aplicações para serem utilizadas como exemplos. Para Johnson[Joh93], a partir deste conjunto de exemplos é que a descrição do domínio poderá ser originada, por isso a importância destes exemplos.

Diferentemente, o projeto orientado a pontos adaptáveis enfoca os aspectos flexíveis do framework, ou seja, os pontos adaptáveis que mudam de aplicação para aplicação. A estrutura de classes é modificada através dos metapadrões, para adaptar os pontos adaptáveis encontrados. Se após a fase de testes, o framework ainda não for considerado adequado, a atenção volta-se à redefinição dos pontos adaptáveis.

3.6 Resumo

Um framework[JF88, WBJ90] é um projeto genérico em um domínio que pode ser adaptado à aplicações específicas, servindo como um molde para a construção de aplicações. Todas as aplicações construídas a partir de um mesmo framework apresentam a mesma estrutura, diferenciando-se em seu comportamento.

Frameworks podem ser classificados segundo a forma como as aplicações são construídas a partir dele[JF88]. Um framework é chamado caixa-branca se o comportamento de uma aplicação específica construída a partir dele é especificado através da redefinição de subclasses e métodos de uma ou mais classes. Pode-se dizer que este framework é reutilizado através da especialização de seus componentes. Frameworks são chamados caixa-preta quando sua customização é feita através da disponibilização de um conjunto de componentes que fornecem o comportamento de aplicações específicas. Cada um destes componentes precisa entender um protocolo particular. Todos, ou a maioria deles, são fornecidos por uma biblioteca de componentes. Neste caso, este framework é reutilizado através da instanciação de seus componentes. Estes frameworks são mais fáceis de usar, no entanto são menos flexíveis que os frameworks caixa-branca[JF88, Tal95, FS97a].

Geralmente, um framework possui características dos frameworks caixa-branca e dos

caixa-preta. Em outras palavras, alguns de seus componentes são reutilizados através de herança, enquanto outros são reutilizados através de composição. Por isso costuma-se dizer que esta classificação corresponde a um espectro contínuo onde os pontos extremos correspondem às definições apresentadas.

Assim como existem metodologias para o desenvolvimento de aplicações orientadas a objetos, também existem metodologias para o desenvolvimento de frameworks. As duas principais são: o projeto dirigido a exemplos[Joh93] e o projeto orientado a pontos adaptáveis[Pre94a, 228–231]. O projeto dirigido a exemplos caracteriza-se pela enfâse na análise de aplicações previamente desenvolvidas. A partir destes exemplos os aspectos comuns às aplicações são abstraídos e posicionados em classes abstratas, enquanto que as particularidades corresponderão às classes concretas.

Diferentemente, o projeto orientado a pontos adaptáveis enfoca os aspectos flexíveis do framework, ou seja, os pontos adaptáveis, que são os que mudam de aplicação para aplicação. Inicialmente, é feita a modelagem utilizando-se uma metodologia de análise e projeto orientado a objetos. O framework é gerado através de uma série de transformações na estrutura gerada pela metodologia, tal que os pontos adaptáveis identificados possam ser flexíveis o suficiente para acomodar o domínio do framework. Metapadrões[Pre94b] são utilizados para realizar estas transformações.

Padrões de projetos são, basicamente, uma parcela de experiência destilada, descrita como uma solução para um problema em uma dada situação[Cam96]. Eles têm sido propostos como um meio de representar, registrar e reutilizar micro-arquiteturas de projeto repetitivas, bem como a experiência acumulada pelo projetista ao desenvolver estas estruturas. Eles formam uma base de experiência para construir sistemas reutilizáveis, atuando como blocos pré-fabricados para a construção de sistemas complexos.

A experiência capturada através dos padrões de projeto pode existir em diferentes níveis de abstração, desde linguagens de programação até arquiteturas de alto nível. Em linguagens de programação a principal abordagem é a proposta por Pree[Pre94a] que age basicamente como convenções no modo de atribuir nomes a variáveis, métodos, etc. Em projeto orientado a objetos a abordagem mais utilizada é a proposta por Gamma et al.[GHJV94] que consiste em um catálogo de padrões que são descrições de três a quatro objetos e/ou classes que se comunicam e que são customizadas para resolver problemas gerais de projeto em um contexto particular. Gamma descreve para cada padrão um nome; o problema que ele resolve; a solução adotada pelo padrão, apresentada através de diagramas de classes; e as conseqüências de sua utilização.

Metapadrões[Pre94b, Pre94a] descrevem como construir frameworks independentes de um domínio específico. Eles são uma abordagem elegante e poderosa que pode ser utilizada para classificar e descrever padrões de projeto em um meta nível. Os metapadrões não substituem as abordagens de padrões de projeto, mas complementam-as, pois apenas estes

padrões não são suficientes para apoiar o desenvolvimento efetivo de frameworks.

Os metapadrões baseiam-se na utilização de métodos *template* e *hook*. Com alguma imprecisão, pode-se dizer que métodos complexos (métodos *template*) podem ser implementados baseados em métodos elementares (métodos *hook*). Os métodos *template* são um modo de definir o comportamento abstrato, o fluxo de controle genérico ou o relacionamento entre objetos de um framework. Em outras palavras, estes métodos descrevem os pontos fixos do framework, aqueles aspectos comuns ao domínio de aplicação do mesmo; enquanto que os métodos *hook* implementam os pontos adaptáveis deste framework. Deste modo, um framework bem projetado deve conter um conjunto flexível de métodos *template*. Idealmente, os clientes do framework somente tem que sobrecarregar os métodos *hook*.

Capítulo 4

O Modelo de Cooperação

Este capítulo apresenta o modelo de cooperação a ser implementado pelo framework ABCDE. Este modelo baseia-se em experiências provindas da atividade de escrita colaborativa apresentadas no Capítulo 2, que foram adaptadas para as atividades de análise e projeto de sistemas[SWR97]. Seção 4.1 apresenta a idéia geral do modelo, que consiste na utilização de anotações sobre diagramas. Seção 4.2 apresenta o modelo de revisão, enquanto que a seção 4.3 apresenta o modelo de co-autoria. Seção 4.4 apresenta as características gerais do modelo, válidas para o modelo de revisão e também para o modelo de co-autoria. Ao final, a seção 4.5 apresenta algumas possíveis extensões do modelo de cooperação para atividades específicas da engenharia de software.

4.1 Anotações em Diagramas

As anotações são uma forma de comunicar idéias ou opiniões sobre um documento. Elas servem como um mecanismo indireto de comunicação entre os autores de um determinado documento, permitindo que um autor apresente idéias ou opiniões, proponha alterações, apresente suas dúvidas, etc aos outros autores do diagrama.

O autor constrói o documento, um diagrama por exemplo, e entrega-o para o outro usuário que deve revisá-lo. Este revisor analisa o diagrama e insere suas anotações associadas a elementos semânticos do diagrama. Estas anotações serão avaliadas posteriormente pelo autor. As anotações são inseridas com o intuito de aperfeiçoar o diagrama e também podem ser utilizadas como registro do processo de discussão ocorrido durante o desenvolvimento do diagrama.

Além disso, segundo Neurwith et al.[NKCM90], no processo de escrita colaborativa é desejável que co-autores e/ou revisores tenham acesso ao planejamento do texto a ser desenvolvido, pois a comunicação sobre os esboços e restrições pode melhorar o desempenho do grupo. Esta melhoria ocorre porque evita que co-autores e revisores tenham que

inferir sobre o trabalho do autor.

No desenvolvimento cooperativo de software tal característica também é importante, pois permite ao autor justificar suas decisões de modelagem e/ou apresentar suas dúvidas aos outros projetistas. Por exemplo, uma dúvida comum entre projetistas de software orientado a objetos é se um determinado conceito deve ou não ser modelado como uma classe. A partir desta decisão, novas decisões que são influenciadas por esta precisam ser tomadas, no entanto, tais decisões só fazem sentido no contexto da primeira decisão. Se os outros projetistas tem acesso à justificativa da decisão original, o entendimento das outras decisões torna-se mais fácil.

Isto pode ser obtido permitindo que o autor faça anotações sobre seu próprio diagrama. Utilizando as anotações, o autor pode justificar suas decisões, apresentar dúvidas, apresentar decisões relacionadas, etc.

Toda anotação possui um dono, que é o usuário que a inseriu. No entanto, é interessante também permitir anotações anônimas para evitar os problemas discutidos na seção 2.3.

A utilização de anotações em diagramas foi utilizada inicialmente no ambiente XNetwork [RS92] que permite o projeto cooperativo de redes de computadores (seção 2.6.4).

4.1.1 Tipos de Anotações

As anotações são utilizadas como um mecanismo de comunicação entre os autores de um diagrama. No entanto, esta comunicação pode ser de diferentes formas, como por exemplo, através da apresentação de opiniões, através da apresentação de sugestões de alterações, através de questionamentos, etc. Para modelar estas diferentes formas de comunicação, diferentes tipos de anotações são utilizadas:

- **Comentário**
  Corresponde a um trecho de texto que está associado a um conjunto de elementos do diagrama. Geralmente, eles descrevem opiniões ou idéias de seus autores sobre os elementos anotados. Comentários contém frases como: "Esta classe não está modelada corretamente.", "Você esqueceu o atributo X nesta classe.", "Este relacionamento pode ser modelado como uma agregação.", "Qual a necessidade do atributo Y nesta classe?", etc.

  Figura 4.1 apresenta um exemplo de uma anotação do tipo comentário sobre um atributo matrícula da classe Aluno.

  Figura 4.2 apresenta um exemplo de um comentário feito sobre um relacionamento de agregação e suas respectivas classes.

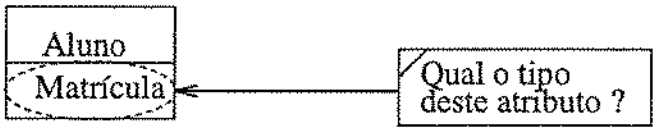


Figura 4.1: Exemplo de um Comentário

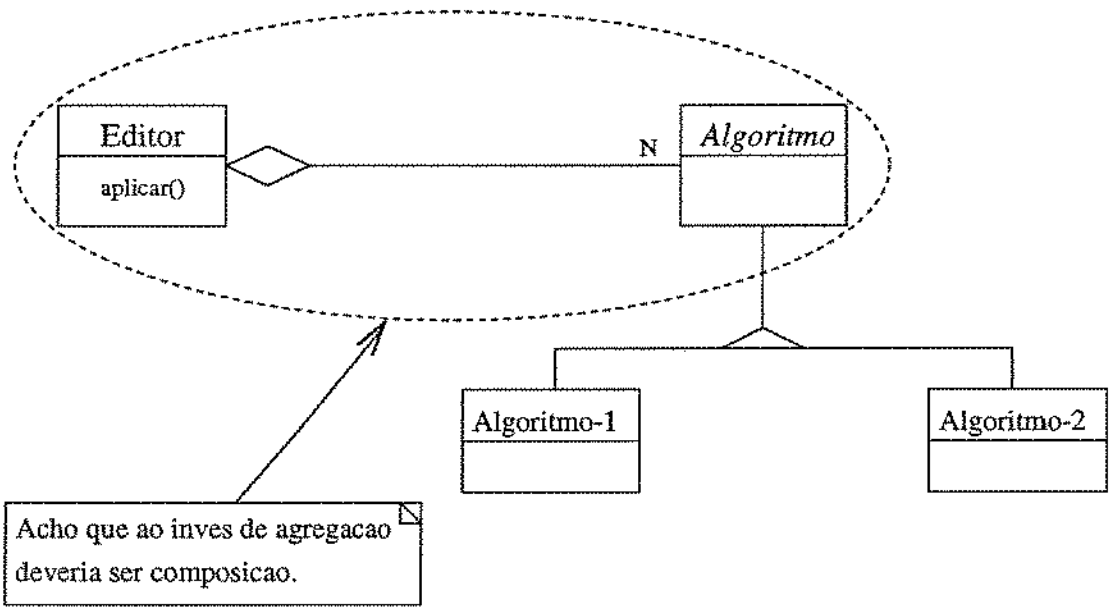


Figura 4.2: Outro exemplo de Comentário

- **Substituição**
  Corresponde a uma sugestão de alteração no diagrama original. Ela deve ser apresentada diretamente sobre o diagrama original, através da substituição dos elementos anotados pelos elementos que compõem a anotação. A existência de substituições é importante em um sistema que apóie anotações, pois determinados usuários ficam insatisfeitos se não podem propor alterações no documento. Além disso, em muitos casos refazer o diagrama, ou parte dele, é muito mais eficiente que a tentativa de diagnosticar as deficiências do diagrama original[NKCM90].

  Uma substituição pode ser feita sobre um elemento ou um conjunto de elementos do diagrama. Figura 4.3 apresenta um exemplo de uma anotação do tipo substituição, onde ao invés de modelar o conceito de Carburador como atributo de um Motor, sugere-se que ele seja considerado uma parte do mesmo, ou seja, cria-se um relacionamento de agregação entre a classe Motor e a classe Carburador.

  O projeto de uma ferramenta que apóie substituições deve permitir que elas sejam avaliadas diretamente no contexto original onde elas foram criadas[NKCM90]. Em outras palavras, quando o usuário solicitar a apresentação de uma substituição, esta ferramenta deve remover do diagrama os elementos que foram anotados e em seu lugar, inserir os elementos que compõem a substituição. No entanto, um problema de visualização do diagrama pode ocorrer. Este problema, e sua solução são descritos na seção 4.4.7.

  Outro problema que pode ocorrer é um problema de consistência no momento em que a substituição está sendo construída. Este problema refere-se à alteração de um relacionamento entre elementos por outro relacionamento que semanticamente não pode ser feito entre estes elementos. Por exemplo, se um usuário tentasse substituir um relacionamento de instanciação existente entre uma classe e uma metaclasse em um diagrama UML[BRJ97] por um relacionamento de agregação, um editor que implementasse este modelo não deveria permitir que esta substituição fosse criada, pois segundo a metodologia UML o único relacionamento que pode existir entre uma classe e uma metaclasse é instanciação. Este problema pode ser resolvido se o editor também implementar o metamodelo da metodologia, pois o mesmo define o conjunto de modelos válidos que podem ser construídos com a metodologia[Rum95].

- **Proposta**
  Corresponde a um conjunto de anotações relacionadas que são tratadas como uma anotação única. Assim, todas as operações que são aplicadas a uma proposta, também são aplicadas às anotações que a compõem. Por exemplo, quando uma proposta é visualizada, todas as anotações que a compõem também são visualizadas.

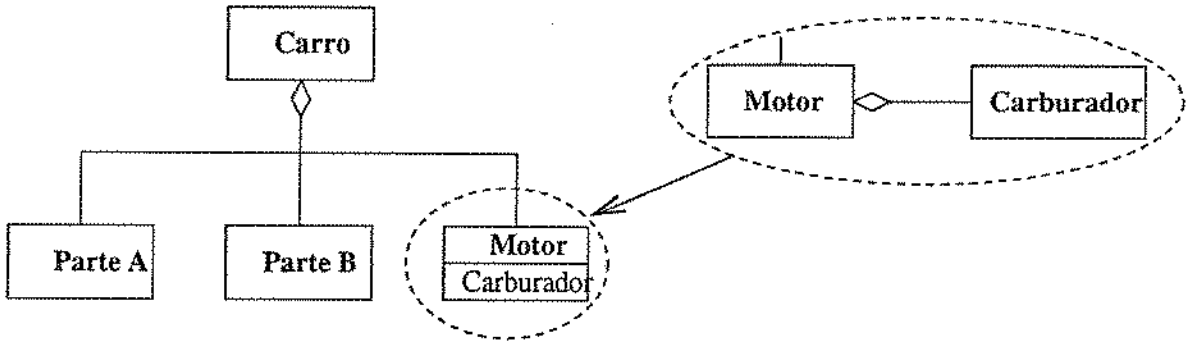


Figura 4.3: Exemplo de uma Substituição

Figura 4.4 apresenta duas soluções para a implementação de uma pilha a partir de uma lista: herança ou agregação[RBP+91]. O Modelo 1 apresenta a implementação utilizando-se herança, enquanto que o Modelo 2 apresenta a mesma modelagem utilizando-se agregação. Se for utilizada agregação (Modelo 2), então deve-se necessariamente incluir um atributo do tipo lista na classe pilha. Ou seja, a modificação no relacionamento entre lista e pilha deve ser acompanhada da inclusão de um atributo em pilha. A proposta seria formada por uma substituição no relacionamento de herança, por um relacionamento de agregação e uma substituição que inclui um novo atributo na classe Pilha. Esta é uma situação comum no desenvolvimento de sistemas, onde uma decisão de modelagem influencia outras decisões.

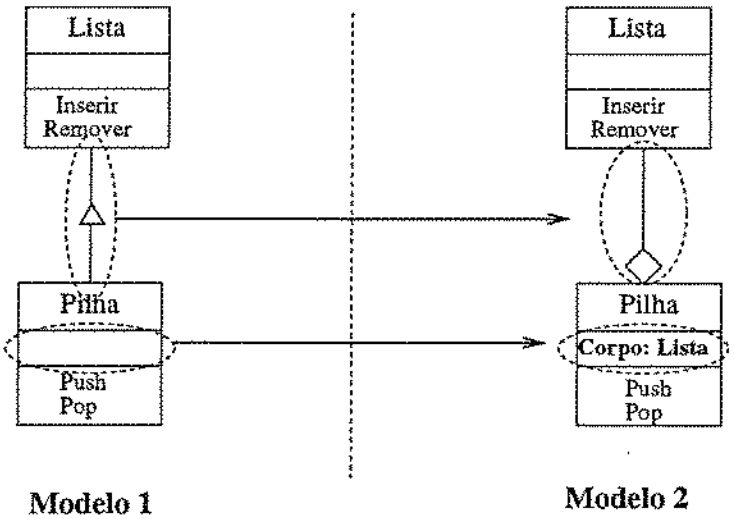


Figura 4.4: Exemplo de uma Proposta

Os comentários também podem ser *flutuantes*, ou seja, comentários que não estão associados a elementos específicos do diagrama, e sim ao diagrama como um todo. Isto

permite que a discussão seja feita em diferentes níveis, desde a discussão sobre elementos unitários do diagrama, até a discussão sobre características gerais do mesmo[RS92]. As substituições também não precisam estar associadas à elementos do diagrama, pois o modelo proposto deve permitir que as várias fases de uma metodologia sejam auxiliadas. Por exemplo, a metodologia OMT[RBP+91] propõe uma fase de identificação de classes. Um projetista pode desejar sugerir a inserção de uma classe que não foi identificada por outro usuário, logo este projetista não deve associar a anotação a nenhum elemento específico do diagrama e sim ao diagrama como um todo.

Figura 4.5 apresenta o modelo de objetos utilizando a notação UML[BRJ97] para os diferentes tipos de anotações. Uma anotação pode ser um comentário, uma substituição ou uma proposta. Uma substituição é formada por elementos do diagrama, enquanto que a proposta pode ser composta recursivamente por uma ou mais anotações.

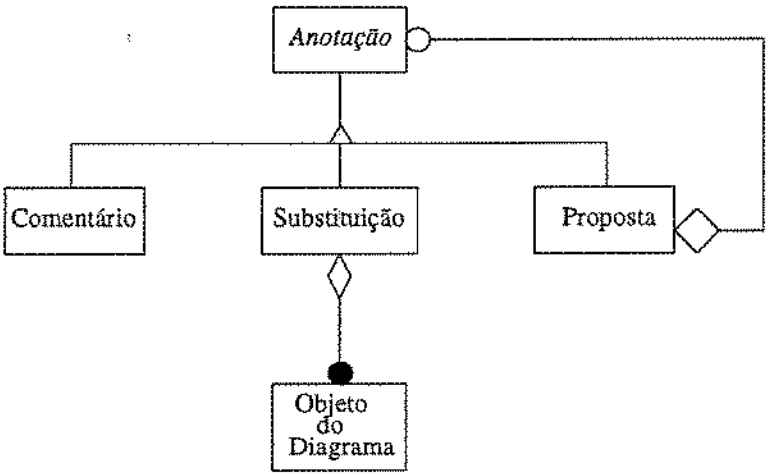


Figura 4.5: Modelo de Objetos para Anotações

4.1.2 Escopo de Visibilidade

Toda anotação apresenta um escopo de visibilidade, ou seja, a especificação do conjunto de usuários que pode visualizá-la. O escopo de visibilidade pode ser utilizado para delimitar discussões entre usuários. Por exemplo, dois usuários encarregados de modelar uma determinada hierarquia podem delimitar sua discussão e evitar que a mesma atrapalhe o trabalho das outras pessoas.

O escopo de visibilidade de uma anotação pode ser:

- Público: qualquer usuário pode visualizar a anotação;
- Específico: apenas um conjunto determinado de usuários pode visualizar a anotação.

Em casos extremos apenas o revisor e o autor podem visualizar a anotação. Este modo de operação é apropriado quando não se deseja que um revisor influencie o outro, como por exemplo em inspeções formais de software[Pre95].

4.1.3 Estados

O modelo de cooperação proposto associa estados às anotações. Uma anotação pode estar em dois estados exclusivos: *ativa*, quando a anotação está visível dentro do domínio de visibilidade do usuário; ou *inativa*, quando a anotação não está mais visível dentro deste domínio de visibilidade. O domínio de visibilidade de um usuário corresponde ao conjunto de anotações que o usuário pode visualizar, ou seja, às anotações cujo escopo de visibilidade incluem este usuário.

No momento em que uma anotação é criada ela encontra-se ativa. Ela torna-se inativa através de uma requisição explicíta do usuário. A partir deste momento, ela não é mais apresentada no domínio de visibilidade daquele usuário. Uma anotação inativa ainda pode ser visualizada, mas para isso é necessário que o usuário requeira explicitamente sua visualização.

Todas as anotações, ativas e inativas, são sempre registradas. Assim, elas podem ser utilizadas como uma forma de documentar a discussão que conduziu o processo de construção do diagrama.

4.2 O Modelo de Revisão

4.2.1 Descrição do Modelo

Este modelo de cooperação apresenta uma definição explícita de papéis: o autor, que é o responsável pela criação do diagrama, e os revisores, que incluem as anotações sobre o diagrama. Os revisores não podem alterar o diagrama, eles somente podem fazer comentários e sugerir modificações através da inclusão de anotações. Cabe ao autor aceitar ou não as sugestões e efetuar as devidas alterações no diagrama. A definição de papéis auxilia a tarefa de coordenação e ajuda a aumentar a percepção, pois fornece uma idéia sobre o significado do trabalho dos outros usuários.

4.2.2 Operações sobre as Anotações

Toda anotação possui um conjunto de operações associadas. Estas operações correspondem a métodos em objetos, e assim como no paradigma orientado a objetos, elas definem o comportamento de um objeto. A partir da modificação destes métodos pode-se modi-

ficar o comportamento das anotações. As operações dependem do tipo de anotação e do papel usuário que solicita a operação.

Um autor de um diagrama pode efetuar as seguintes operações sobre um comentário:

- Visualizar: corresponde a apresentação do comentário para o autor.
- Tornar Inativo: o autor já avaliou o comentário, logo não deseja mais considerá-lo. O comentário é, então, retirado do domínio de visibilidade do usuário. No momento em que uma anotação se torna inativa, todas as anotações que apresentam dependência forte com a mesma passam a incluir esta anotação. Assim, a aplicação de uma anotação que depende fortemente de uma outra que se tornou inativa implica inicialmente, na aplicação da anotação que foi removida, e depois, na aplicação da própria anotação.

As operações permitidas sobre uma substituição e/ou proposta são:

- Aplicar: para que um autor avalie a anotação proposta, ele deve ser capaz de analisá-la no contexto existente[NKCM90]. Por exemplo, dada uma substituição, para que o autor possa analisá-la, ele deve visualizar a substituição sobre o diagrama original. Desta forma, ele pode verificar como a substituição influencia o diagrama no contexto original. Então, a aplicação consiste na retirada do diagrama dos elementos anotados, seguida da inserção da substituição no diagrama.
- Incorporar: quando uma anotação é incorporada, as alterações existentes em uma substituição ou proposta tornam-se parte efetiva do diagrama. Esta operação só pode ser efetuada pelo autor.
- Tornar Inativa: a mesma operação definida para um comentário.

Um revisor pode efetuar as operações abaixo, assim como o autor:

- Criar anotação: uma anotação, comentário, substituição ou proposta, pode ser inserida em um diagrama, podendo estar associada ou não à elementos do mesmo.
- Visualizar: operação realizada sobre um comentário que consiste na apresentação do mesmo.
- Aplicar: operação realizada sobre uma substituição ou proposta que consiste na apresentação do mesmo.

4.2.3 Estágios de Utilização

De maneira similar à tarefa de revisão na autoria cooperativa, o modelo proposto baseia-se no ciclo edição-revisão-incorporação para sua utilização. Optou-se por renomear algumas fases, pois este modelo pode ser aplicado à atividade de desenvolvimento de software.

- **Criação**
  Neste estágio inicial, o autor efetua a construção do diagrama. Se considerarmos que o autor não insere nenhuma anotação em seu próprio diagrama, pode-se afirmar que a funcionalidade de um editor que implementa o modelo proposto no estágio de criação é similar à funcionalidade de um editor convencional de diagramas.

- **Discussão**
  Neste estágio intermediário, a discussão é efetuada pelos revisores que são responsáveis pela inserção de anotações no diagrama. O modelo permite que anotações sejam inseridas sobre outras anotações. Assim, as anotações inseridas no diagrama se relacionam umas com as outras de duas formas: dependência forte e independência. *Dependência forte* entre anotações, ocorre quando uma anotação é inserida sobre elementos comuns de uma anotação que foi anteriormente aplicada. Assim, para que a anotação possa ser aplicada/visualizada é necessário inicialmente aplicar a primeira substituição, e após isso, aplicar/visualizar a anotação realmente desejada. Figura 4.6 apresenta um exemplo de dependência forte, onde um comentário é feito sobre outro comentário. Figura 4.7 apresenta um segundo exemplo de dependência forte entre duas anotações. O modelo de objetos 1 é o modelo original, o modelo 2 é criado a partir da incorporação da substituição S1 ao modelo original. O modelo 3 é criado a partir da incorporação da substituição S2 ao modelo 2. Note que para que a substituição S2 possa ser aplicada, é necessário que a substituição S1 já tenha sido feita.

  Independência entre anotações, ocorre quando as duas anotações não estão relacionadas, não havendo obrigatoriedade da aplicação/visualização de qualquer uma delas para que a outra possa ser aplicada/visualizada.

- **Incorporação**
  Neste estágio final, o autor pode avaliar as anotações que foram inseridas pelos revisores no estágio anterior. O autor, ao avaliar as anotações, pode incorporá-las ou não ao diagrama. Mesmo que as anotações não sejam incorporadas ao diagrama, elas são armazenadas de modo a documentar as discussões sobre o projeto.

Após a definição inicial do modelo de objetos, os estágios de criação e incorporação podem se mesclar, de tal forma que o autor pode incorporar anotações ao diagrama ao mesmo tempo em que insere novos elementos.

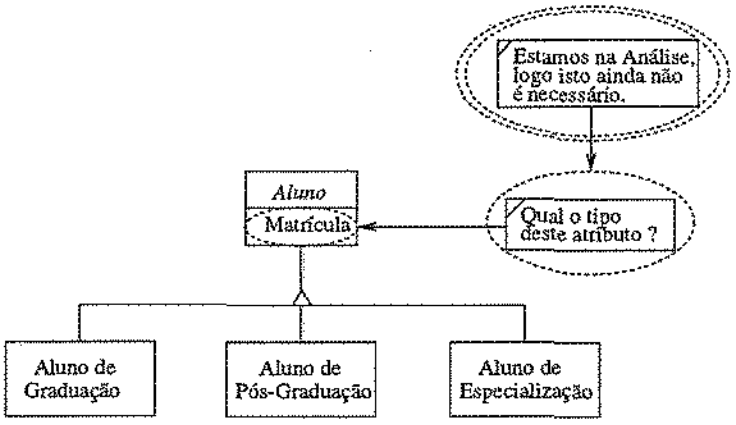


Figura 4.6: Exemplo de Dependência Forte entre Anotações

4.3 Modelo de Co-Autoria

O modelo de co-autoria estende o modelo de revisão descrito anteriormente, tornando a distinção entre autor e revisores menos precisa. Enquanto que no modelo de revisão somente o autor pode criar elementos no diagrama, neste modelo todos os usuários podem criar elementos. Todos os usuários também podem alterar elementos desde que eles sejam donos destes elementos. Assim, todos os usuários desempenham o papel de autores. Se o usuário não é o dono de um elemento, então ele não pode alterá-lo. No entanto, ele poderá propor alterações ao mesmo através da inserção de anotações. Assim, todos os usuários também desempenham o papel de revisores. A distinção entre os papéis que um usuário desempenha é feita a partir do atributo *dono* de cada elemento que o usuário está manipulando.

4.3.1 Descrição do Modelo

Neste modelo existem vários co-autores e cada um deles tem direito de criação e discussão sobre os elementos do diagrama. Não existe a definição explícita de papéis, todos os usuários são co-autores do documento. A exceção é o gerente, que deverá ser escolhido entre os co-autores usando critérios próprios do grupo. Este gerente terá os mesmos privilégios e restrições dos outros co-autores, no entanto, terá atividades extras como a especificação do conjunto de usuários que pode acessar o diagrama, a possibilidade de utilizar anotações anônimas, etc. Cada diagrama possui um único gerente.

Os elementos criados por um determinado co-autor constituem o domínio daquele co-autor. Cada co-autor age apenas como um revisor nos domínios dos outros co-autores. O diagrama consiste na união dos domínios dos vários co-autores.

Todo elemento do diagrama, ao ser inserido, apresenta um dono, que é o co-autor

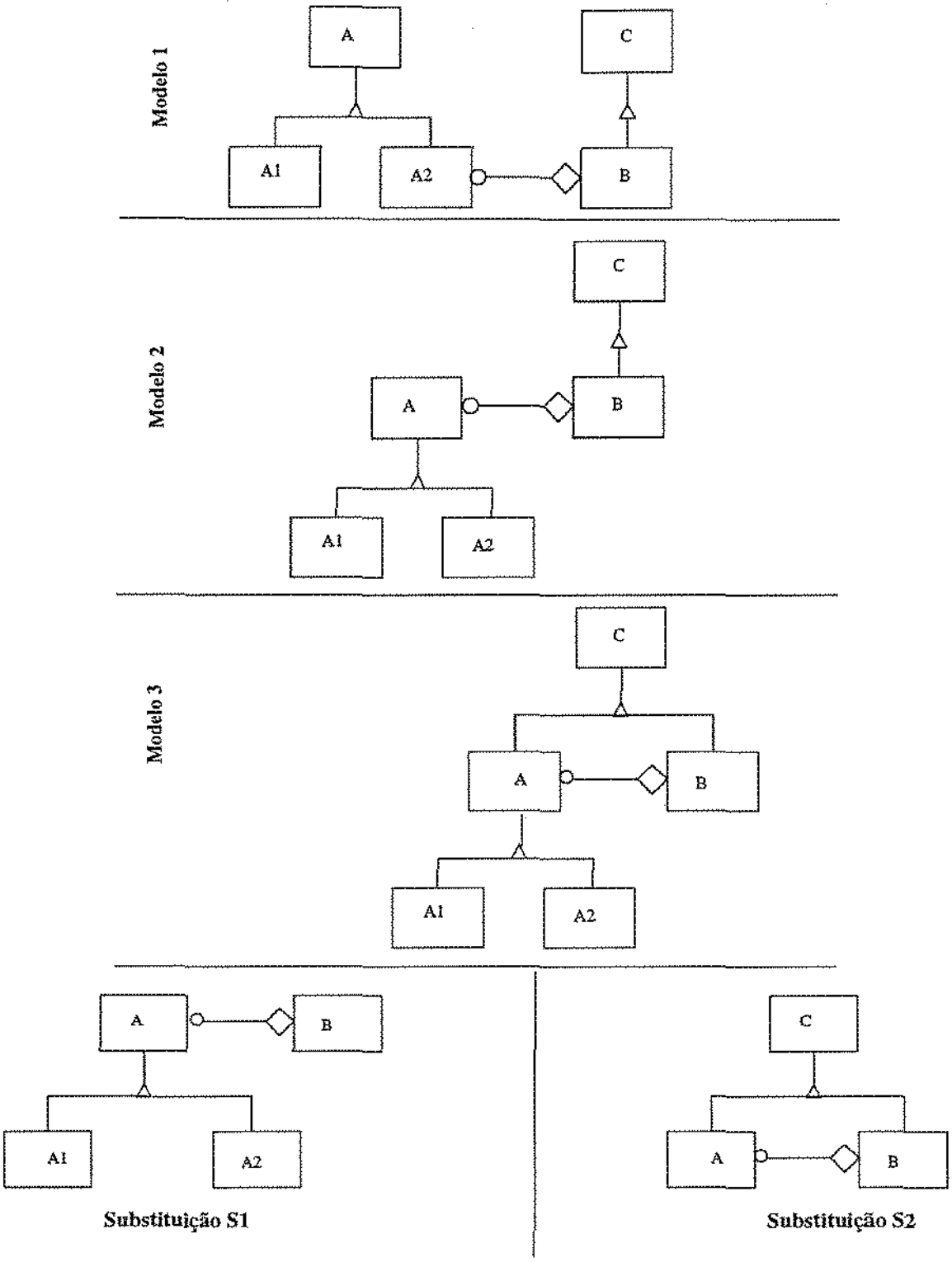


Figura 4.7: Outro exemplo de Dependência Forte entre Anotações

que o inseriu. Este elemento compõe unicamente o domínio daquele co-autor. O dono de um elemento pode alterá-lo de modo que um elemento pode: (i) apresentar um único dono; (ii) não apresentar donos, neste caso o elemento é dito público; ou (iii) apresentar múltiplos donos. Um elemento com múltiplos donos pertence ao domínio de cada um deles. Um elemento público pertence ao domínio de todos os co-autores.

4.3.2 Operações sobre Anotações

Além das operações definidas na seção 4.2.2, outras operações devem ser criadas para o modelo de co-autoria. Elas referem-se a modificações nos donos dos elementos do diagrama, como por exemplo, adicionar novos donos a um elemento, tornar o elemento um objeto público, etc.

O gerente do diagrama pode efetuar todas as operações permitidas aos co-autores. Além disso, ele também tem privilégios como desabilitar a inserção de elementos do diagrama durante discussões, modificar donos de elementos, etc.

4.3.3 Estágios de Utilização

Os estágios de desenvolvimento do modelo de co-autoria são os mesmos do modelo de revisão descritos na seção 4.2.3. No entanto, cada um destes estágios ocorre no domínio de cada co-autor do diagrama. Os estágios podem ocorrer em paralelo, de tal forma que ao mesmo tempo em que um autor pode estar no estágio de incorporação em seu domínio, ele também pode estar no estágio de discussão no domínio de outro co-autor.

A discussão sobre um problema de modelagem pode afetar todo o diagrama e não somente o domínio do co-autor responsável. Assim, é desejável que o gerente do diagrama possa desabilitar o estágio de criação de todos os co-autores, de modo a evitar possíveis inconsistências no diagrama. Assim, o gerente determina que todos os outros co-autores devem esperar que aquela discussão termine para que possam prosseguir seus trabalhos. Desta forma o gerente pode coordenar melhor o trabalho dos co-autores, pois permite que a percepção sobre o trabalho do grupo seja ampliada.

4.4 Características Gerais do Modelo

Esta seção descreve as características gerais do modelo de cooperação proposto. Estas características são válidas tanto para o modelo de revisão, quanto para o modelo de co-autoria.

4.4.1 Independência de Metodologia

As anotações podem ser feitas, isoladamente, ou sobre um conjunto de elementos semânticos da notação do diagrama. Todos os elementos que possuem algum significado semântico na metodologia utilizada podem ser anotados. Por exemplo, no caso do modelo de objetos da metodologia UML[BRJ97], os elementos semânticos da notação são classes, relacionamentos, atributos, cardinalidades, etc. Se a notação do diagrama a ser construído fosse a notação de Diagrama de Fluxo de Dados da Análise Estruturada[You92], então as anotações poderiam ser efetuadas sobre os seguintes elementos: processos, fluxos de dados, entidades externas, etc, que são os elementos semânticos desta notação. Assim, o modelo proposto é independente da metodologia de desenvolvimento de software utilizada, ou seja, ele pode ser utilizado com qualquer metodologia desejada[1].

4.4.2 Interação Semi-Síncrona

O modelo de cooperação descrito pode ser considerado um modelo semi-síncrono[DB92], ou seja, ele permite que a interação seja feita de maneira síncrona ou assíncrona conforme descrito no capítulo 2, seção 2.2.3.

O modelo de revisão pode ser implementado diretamente em um esquema de interação assíncrono, pois ele baseia-se na utilização de ciclos para sua execução: inicialmente, o autor constrói o diagrama; posteriormente, quando ele termina uma "versão" do mesmo, ele repassa-o para os revisores. O modelo de co-autoria também pode ser implementado de maneira assíncrona, pois este modelo não impede que co-autores anotem domínios de outros co-autores, permitindo assim a cooperação entre eles.

A interação síncrona pode ser implementada no modelo de revisão, pois cada revisor somente pode fazer anotações sobre os elementos do diagrama, não existindo a necessidade de mecanismos de controle de consistência do diagrama. Ou seja, um revisor não pode alterar efetivamente o diagrama, logo o controle de concorrência não é necessário.

O modelo de co-autoria também pode ser implementado em um editor síncrono, devido à existência dos conceitos de *domínio* de um elemento e *dono* dos elementos. Estes conceitos permitem que a consistência do diagrama seja mantida, pois segundo o próprio modelo, cada usuário somente pode alterar elementos de seu próprio domínio, enquanto que elementos dos domínios de outros usuários somente podem ser anotados. O *locking* do elemento é feito automaticamente quando um usuário tenta anotá-lo ou alterá-lo (se ele for o dono do elemento). Caso um determinado elemento possua vários donos e dois ou mais deles tentem alterá-lo ao mesmo tempo, define-se que o o primeiro co-autor a fazer a tentativa de alteração obterá o locking. Isto poderia ser implementado através

[1]Na verdade esta afirmativa não é totalmente verdadeira. Veja a seção 6.2.

de uma arquitetura baseada em um coordenador centralizado que seria responsável pela resolução de conflitos.

4.4.3 Flexibilidade

A existência de anotações sobre diagramas é um requisito fundamental para a cooperação durante o desenvolvimento de software, porém diferentes formas de cooperação exigem que diferentes *configurações* do diagrama sejam oferecidas. Estas configurações permitem que o modelo seja flexível o suficiente para acomodar especificidades do seu grupo de usuários.

Uma configuração baseia-se em parâmetros do diagrama, como por exemplo, a *inserção de anotações sobre anotações* não é desejável em algumas situações para evitar que um usuário influencie outro. Um exemplo desta situação ocorre nas inspeções de software[MDTR93]. Entretanto, em outras situações esta característica pode ser desejável, como por exemplo durante o processo de desenvolvimento de sistemas, pois permite que a discussão seja mais rica e produtiva, tornando o todo maior que a soma das contribuições individuais[EGR91]. Outros possíveis parâmetros são:

- O mecanismo de controle de concorrência a ser utilizado: durante o projeto cooperativo de um diagrama UML os conflitos são infreqüentes e assim podemos adotar uma política otimista de controle? Existe um princípio de localidade que faz com que os projetistas se concentrem em regiões específicas do diagrama, e assim podemos utilizar uma política otimista de controle; ou é mais apropriado permitir um mecanismo de *locking*? Pode-se perceber que diferentes políticas de controle de concorrência podem ser empregadas em diferentes situações dependendo de características próprias do grupo, do ambiente de desenvolvimento, etc. Assim, é interessante permitir que a política de controle de concorrência seja definida pelo gerente do grupo e, se necessário, modificada durante o decorrer do projeto.

- Anotações anônimas: a utilização de anotações anônimas deve ser uma decisão do autor, ou gerente, do diagrama, uma vez que apesar das vantagens que ela oferece nem sempre isto ela é necessária.

- Exclusão de anotações: a simples existência das anotações fornece indícios de como o diagrama evolui, porém o autor, ou gerente, pode desejar eliminar anotações que mais tarde mostraram-se erradas, desnecessárias ou inadequadas.

4.4.4 Registro automático das argumentações sobre o diagrama

A utilização das anotações em diagramas permite que o registro das argumentações ocorridas durante o desenvolvimento do diagrama (*design rationale*) seja feito de maneira automática. Este registro ocorre porque as anotações serão *sempre* armazenadas, mesmo que o autor torne-as inativas. Neste caso, para que elas sejam visualizadas é necessário que o usuário requeira explicitamente esta visualização.

A captura da argumentação pode trazer diversos benefícios[dSBCC95, Kar96], tais como:

- a organização dos assuntos a discutir;
- a manutenção e consistência da discussão através das reuniões realizadas;
- a coordenação de pessoas envolvidas no projeto; e
- a possibilidade de reutilizar aquela argumentação, ou seja, a manutenção e utilização do artefato são facilitadas.

Além disso, as anotações são associadas diretamente aos elementos do diagrama sobre os quais elas foram efetuadas, permitindo que a construção e a argumentação do artefato estejam integradas. Desta forma, os aspectos mais cruciais de um argumento são colocados no lugar onde eles são de maior necessidade, ou seja, no próprio diagrama[RS92].

4.4.5 Utilização de mecanismos de Versões e Lista de Ações

As versões de um diagrama correspondem aos diferentes diagramas que foram registrados pelo autor no decorrer do processo de desenvolvimento. O registro destas versões é importante, pois mostra a evolução dos diagramas no decorrer do tempo, fornecendo indicações do raciocínio e argumentações que levaram a produção do diagrama.

Quando uma nova versão do diagrama é criada, as anotações ativas restantes tornam-se automaticamente inativas. Isto equivale a dizermos que cada nova versão do diagrama corrige as deficiências da versão anterior e torna novamente o diagrama disponível para avaliação dos outros usuários.

A criação de listas de ações dos usuários também permite registrar a evolução do diagrama[FcdM94]. Uma lista de ações consiste em uma lista contendo as ações efetuadas por um usuário. Desta forma, pode-se registrar as atividades executadas pelos usuários. Isto auxilia o processo de percepção das atividades, pois permite que um usuário repita a sequência de operações de outro usuário, visando perceber a influência que elas causam em seu próprio trabalho.

Além disso, enquanto as versões permitem registrar a evolução do diagrama como um todo, as listas de ações permitem o registro isolado da evolução de cada uma das versões.

4.4.6 Utilização de Filtros e Coluna de Anotações

A utilização de filtros e de uma coluna para apresentação das anotações, chamada coluna de anotações, é adotada pelo modelo visando resolver o problema do projeto de interfaces de editores que apóiam anotações.

A coluna de anotações é utilizada para permitir a apresentação não-intrusiva das anotações. Mais do que isso, segundo Neurwith *et al.*[NKCM90] a indicação das anotações na forma de hiper-textos não é desejável, pois não permite que os usuários possam rapidamente perceber todas as anotações que foram efetuadas. Desta forma, ao lado de cada diagrama é associada uma coluna cujo objetivo é indicar as anotações efetuadas sobre aquele diagrama. No momento em que uma anotação é visualizada, os elementos que foram anotados são indicados através da modificação de suas cores. Esta solução foi adotada anteriormente por Neurwith *et al.* no editor PREP[NKCM90]. Esta coluna de anotações também permite que o editor de diagramas possa ser utilizado de maneira não-cooperativa, pois apresenta as anotações de maneira não-intrusiva.

Filtros para apresentação de anotações são utilizados aliados à coluna de anotações, permitindo que o usuário possa explorar seletivamente as anotações que foram inseridas. Esta idéia foi proposta originalmente por Engelbart[EGR91] e também é citada como uma extensão a ser implementada no editor XNetwork[RS92].

4.4.7 Utilização de Algoritmos para Redesenho Automático de Diagramas

Um editor, ou um framework, que implemente este modelo de cooperação deve possuir um mecanismo para permitir a apresentação das substituições no contexto original onde elas foram criadas, para permitir uma visualização efetiva das substituições[NKCM90].

Quando uma substituição é aplicada sobre um diagrama, ela deve retirar os elementos sobre os quais ela foi efetuada e, em seu lugar, apresentar os elementos que compõem a substituição. No entanto, o número de elementos que compõem a substituição pode ser maior que o número de elementos originais, ou ainda, eles podem apresentar uma disposição espacial diferente da original. Uma solução para tais problemas é a utilização de *algoritmos para redesenho automático de diagramas*. Estes algoritmos permitiriam que o diagrama fosse redesenhado no momento em que a substituição fosse aplicada, permitindo que elas fossem avaliadas no contexto original onde elas foram criadas.

4.5 Extensões ao Modelo de Cooperação

Nesta seção são descritas duas possíveis extensões onde o modelo de cooperação proposto pode ser utilizado: inspeção de software e decisões de projeto. As extensões que devem ser feitas ao modelo, para que ele possa se adequar às características específicas dessas aplicações são apresentadas através da especialização das anotações do modelo original.

4.5.1 Inspeção de Software

A atividade de inspeção de software é uma das atividades de Verificação & Validação que são utilizadas para garantir a qualidade de software. A inspeção é uma revisão detalhada de uma pequena quantidade de material, efetuada por profissionais de engenharia de software tecnicamente competentes. Ela é um método efetivo para revelar erros nos documentos e no código dos produtos de software[MDTR93]. O processo de inspeção requer seis passos bem definidos: planejamento, *overview*, preparação, reunião, *re-work* e *follow-up* [ABL89].

Pressman [Pre95] apresenta uma série de diretrizes a serem seguidas para a tarefa de inspeção de software, entre elas:

> "Enuncie as áreas problemáticas, mas não tente resolver cada problema anotado ou notado."

Em outras palavras, o caratér de uma inspeção é levantar problemas. As soluções para estes problemas não devem ser apresentadas, pois devem ser deixadas a cargo dos autores do produto. Logo, anotações que sugerem alterações no diagrama (substituição e proposta) não devem ser permitidas, pois fornecem indícios de como resolver o problema que apontam. Um novo tipo de anotação deve ser criado para refletir o formalismo das inspeções de software. Esta anotação, chamada Comentário Formal, é um tipo especial de comentário que apresenta as seguintes operações:

- Aceitar: com esta operação o autor afirma que concorda com o inspetor e que as correções indicadas no comentário formal já foram providenciadas. Assim, pode-se efetuar o controle estatístico dos erros que surgiram. Posteriormente, os inspetores podem verificar se suas correções foram efetuadas.
- Discordar: o autor não concorda com os inspetores a respeito das alterações solicitadas e insere um comentário, dirigido aos inspetores, explicando suas razões para discordar da alteração. Mecanismos convencionais como reuniões, telefonemas ou correio eletrônico são utilizados pelo autor e inspetores para atingir o consenso.

Figura 4.8 apresenta o modelo de objetos original, assim como a nova anotação comentário formal e suas operações específicas.

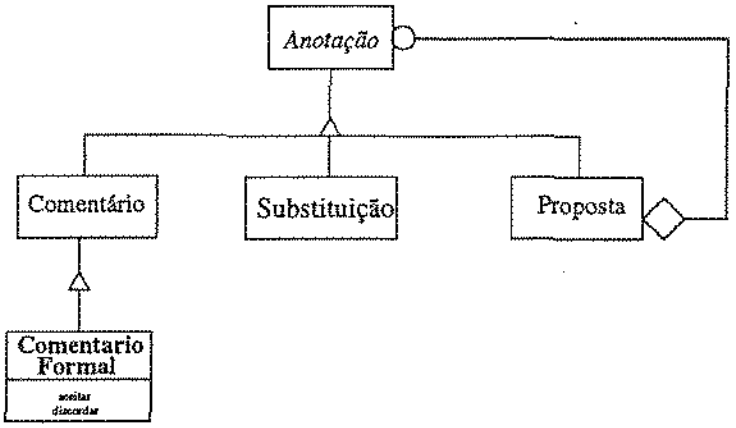


Figura 4.8: Hierarquia de Anotações para a atividade de Inspeção de Software

4.5.2 Decisões de Projeto

As decisões de projeto[2] referem-se às decisões tomadas durante o desenvolvimento de um produto. O registro das várias soluções propostas para os problemas surgidos durante o projeto é necessário, assim como o registro da justificativa para a adoção de uma determinada solução. Segundo [Kar96] a documentação destas decisões é necessária para permitir:

- reutilização de projetos anteriores;
- coordenação de pessoas envolvidas em grandes projetos;
- incitação à reflexão crítica durante o projeto; e
- facilidade na manutenção e na utilização de software.

A maioria dos sistemas de *CSCW* que armazenam as decisões de projeto o fazem de forma que elas são armazenadas separadamente do próprio artefato, o que leva a inúmeros problemas. Visando associar estas decisões ao próprio diagrama, três novos tipos de anotações foram criadas:

- Questão: é um tipo específico de comentário que apresenta o problema de modelagem que está em discussão.
- Opção: apresenta uma solução para um problema que foi levantado por uma questão. Pode ser um tipo de comentário (opção textual) ou um tipo de substituição (opção).

[2]Em Inglês: *design rationale*.

- Justificativa: é um tipo de comentário que o autor deve incluir no momento em que modificações nos elementos do diagrama são efetuadas. Estas modificações podem ser: modificações em elementos do diagrama efetuadas diretamente pelo autor; ou modificações através da aceitação de uma dentre as várias opções de uma questão; ou modificações através da incorporação de uma substituição ou proposta.

Figura 4.9 apresenta o modelo de objetos para as anotações do modelo de cooperação apresentado e as anotações específicas do processo de desenvolvimento de software com o histórico das decisões de projeto: questão, opção textual e justificativa são subclasses de comentário, e opção que é uma subclasse de substituição.

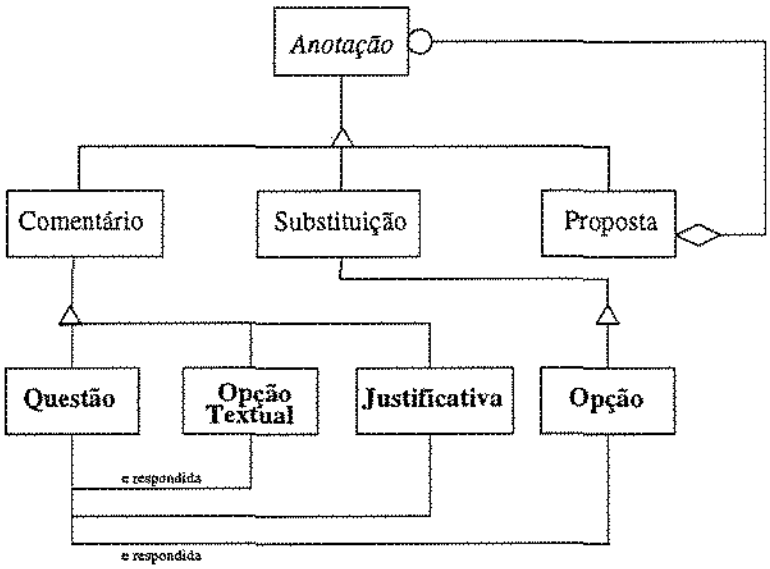


Figura 4.9: Hierarquia de Anotações para Decisão de Projeto

4.6 Resumo

Este capítulo apresenta o modelo de cooperação a ser implementado pelo framework ABCDE. Este modelo baseia-se em experiências provindas da atividade de escrita colaborativa que foram adaptadas para as atividades de análise e projeto de sistemas[SWR97]. O modelo de cooperação consiste na especificação dos tipos de anotações existentes, no conjunto de operações que podem ser efetuadas sobre estas anotações, e que definem os seus comportamentos, e nos estágios do processo de colaboração.

A cooperação entre os usuários é obtida através da inserção de anotações nos diagramas. Isto permite que o registro das argumentações ocorridas durante o desenvolvimento

do diagrama (*design rationale*) seja feito de maneira automática. Este registro ocorre porque as anotações são armazenadas associadas ao diagrama, independentemente de sua avaliação. As anotações são utilizadas como um mecanismo de comunicação entre os autores de um diagrama. Para modelar as diferentes formas de comunicação que são necessárias entre os co-autores do diagrama os seguintes tipos de anotações são utilizados.

- Comentário: corresponde a um trecho de texto que está associado a um conjunto, vazio ou não, de elementos do diagrama. Geralmente, eles descrevem opiniões ou idéias de seus autores sobre os elementos anotados, como por exemplo, "Esta classe não está modelada corretamente.", "Você esqueceu o atributo X nesta classe.",etc.

- Substituição: corresponde a uma sugestão de alteração no diagrama original. Ela pode ser feita sobre um elemento ou um conjunto de elementos do diagrama e é apresentada diretamente sobre o diagrama original através da substituição dos elementos anotados pelos elementos que compõem a anotação. Sua existência permite que outros usuários possam propor alterações no diagrama, ao invés, de somente descrever suas opiniões e idéias sobre o mesmo.

- Proposta: corresponde a um conjunto de anotações relacionadas que são tratadas como uma anotação única. Assim, todas as operações que são aplicadas a uma proposta, também são aplicadas às anotações que a compõem. Esta anotação é utilizada para permitir que usuários possam modelar o sistema utilizando decisões de modelagem que se influenciam.

O uso do modelo baseia-se em três estágios: *criação, discussão* e *incorporação*. No estágio de criação o autor efetua a construção do modelo de objetos. No estágio de discussão os revisores inserem as anotações no diagrama, logo deve-se fornecer apoio a manipulação de anotações. No estágio de incorporação o autor avalia as anotações que foram inseridas, podendo incorporá-las ou não ao diagrama. Após a definição do modelo de objetos inicial, os estágios de criação e incorporação podem se mesclar, de tal forma que o autor pode incorporar anotações ao diagrama, assim como inserir novos elementos.

A manipulação de anotações implica na criação de mecanismos de visualização das anotações, mecanismos de indicação de sua existência, etc. A indicação da existência das anotações deve ser feita de modo não-intrusivo, para isso utiliza-se uma *coluna de anotações*[NKCM90]. Se um co-autor deseja visualizar uma anotação basta ele selecionar a anotação a ser apresentada, caso contrário, ele pode editar o diagrama normalmente. As anotações apresentadas nesta coluna são apresentadas através de *filtros*[RS92] que restringem as anotações apresentadas. A visualização efetiva das anotações deve ser feita através de algum mecanismo que permita a apresentação das anotações no contexto original onde elas foram criadas[NKCM90]. Assim, quando uma substituição é aplicada sobre

um diagrama ela deve retirar os elementos sobre os quais ela foi efetuada e em seu lugar apresentar os elementos que compõem a substituição. No entanto, o número de elementos que compõem a substituição pode ser maior que o número de elementos originais, ou ainda, eles podem apresentar uma disposição espacial diferente da original. Uma solução para tais problemas é a utilização de *algoritmos para redesenho automático de diagramas*.

O modelo desenvolvido pode ser utilizado nas duas principais formas de cooperação: revisão e co-autora. Na revisão existe uma definição explícita de papéis: o autor, que é o responsável pela criação do diagrama, e os revisores, que incluem as anotações sobre o diagrama. Os revisores não podem alterar o diagrama, eles somente podem fazer comentários e sugerir modificações através da inclusão de anotações. Cabe ao autor aceitar ou não as sugestões e efetuar as devidas alterações no diagrama. Neste caso, os estágios de utilização do modelo são muito bem definidos: o autor é o responsável pela criação, os revisores pela discussão, e novamente o autor é responsável pela incorporação.

A co-autoria estende a revisão tornando a distinção entre autor e revisores menos precisa. Enquanto que no modelo de revisão somente o autor pode criar elementos no diagrama; na co-autoria todos os usuários podem criar elementos. Todos os usuários também podem alterar elementos, desde que eles sejam donos destes elementos. Assim, todos os usuários desempenham o papel de autores. Se o usuário não é o dono de um elemento, então ele não pode alterá-lo. No entanto, ele poderá propor alterações ao mesmo através da inserção de anotações. Os estágios de utilização, neste caso, podem ocorrer em paralelo, de tal forma que ao mesmo tempo em que um autor pode estar no estágio de incorporação em seu domínio, ele também pode estar no estágio de discussão no domínio de outro co-autor. O gerente do diagrama pode desabilitar o estágio de criação de todos os co-autores porque determinados problemas de modelagem podem influenciar a modelagem de todo o diagrama. Desta forma, o gerente pode coordenar melhor o trabalho dos co-autores, pois permite que a percepção sobre o trabalho do grupo seja ampliada.

O modelo proposto é flexível o suficiente para acomodar atividades específicas do processo de desenvolvimento de software. Isto pode ser feito através da especialização das anotações originalmente definidas: comentário, substituição e proposta. Duas possíveis extensões ao modelo de cooperação proposto foram apresentadas: inspeção de software e decisões de projeto.

Capítulo 5

O Framework ABCDE

Este capítulo apresenta o framework ABCDE (*Annotation Based Cooperative Diagram Editor*)[SRW98] que permite a construção de editores de diagramas cooperativos. O principal ponto adaptável deste framework é o modelo de cooperação implementado, permitindo assim que os editores instanciados a partir dele possam ser utilizados em diferentes atividades do processo de desenvolvimento de software. Isto é possível porque o framework implementa o modelo de cooperação desenvolvido no capítulo anterior. Editores construídos a partir deste framework permitem a edição cooperativa de diagramas de classes da notação UML[BRJ97].

Este capítulo está organizado como segue. Seção 5.1 apresenta a metodologia utilizada no projeto do framework. Esta metodologia baseia-se na abordagem de projeto orientado a pontos adaptáveis[Pre94a] combinada com o catálogo de padrões de projeto de Gamma et *al.*[GHJV94].

Seção 5.2 descreve as funcionalidades de um editor cooperativo qualquer de diagramas baseado em anotações. A partir desta especificação, a metodologia OMT[RBP+91] é utilizada para modelar este editor. O framework ABCDE é estruturado a partir de uma sequência de transformações na modelagem do editor utilizando-se metapadrões e padrões de projeto, visando obter a flexibilidade necessária a um framework. A seção 5.4 descreve a implementação do framework utilizando-se a linguagem de programação Java[1][Fla97].

Ao final, a seção 5.5 apresenta o sistema ABCDE-Web[SWR98] que é um editor de diagramas cooperativo que pode ser utilizado na Internet e que foi instanciado a partir do framework ABCDE. Este editor foi desenvolvido visando avaliar se os pontos adaptáveis implementados pelo ABCDE oferecem o grau de flexibilidade desejado. Além disso, ABCDE-Web contribui para o avanço do conhecimento científico na área de aplicações cooperativas na Internet pois ele implementa um modelo de cooperação bastante flexível.

[1]Java é uma marca registrada da Sun Microsystems, Inc.

5.1 Metodologia de Desenvolvimento

A metodologia utilizada durante o desenvolvimento do framework ABCDE foi a de projeto orientado a pontos adaptáveis proposta por Pree[Pre94a]. Esta metodologia foi escolhida porque ela permite que o framework possa ser construído sem que seja necessário o desenvolvimento de outras aplicações similares (exemplos). Ela propõe a utilização de metapadrões como mecanismo para flexibilizar o projeto do framework. No entanto, os metapadrões são muito abstratos, o que os torna difíceis de manusear e entender. Além disso, a excessiva flexibilidade dos metapadrões pode ser uma desvantagem, uma vez que para problemas similares, soluções diferentes podem ser desenvolvidas [Sch96].

Visando resolver estes problemas, optou-se pela combinação da abordagem de projeto orientado a pontos adaptáveis com o catálogo de padrões de projeto de Gamma et *al.*[GHJV94]. Os padrões de projeto de Gamma são mais concretos, entretanto, eles se adequam a domínios específicos de problemas. O objetivo da combinação das abordagens de metapadrões e padrões de projeto é suprir suas deficiências. Segundo Schmid[Sch96], um esforço considerável de desenvolvimento pode ser poupado quando um projetista utiliza padrões de projeto para projetar e detalhar a estrutura de pontos adaptáveis do framework.

Sempre que possível, o projeto do framework ABCDE foi feito utilizando-se padrões de projeto quando estes cobriam a flexibilidade requerida para o ponto adaptável em questão; caso contrário, os metapadrões foram utilizados. Desta forma, espera-se que o projeto do framework seja o mais concreto possível, auxiliando os usuários a entender seu projeto e funcionamento.

5.2 Especificação do Editor

Nesta seção são definidas as funcionalidades de um editor de diagramas cooperativo convencional, ou seja, os requisitos de cooperação não são abordados. As funcionalidades que se referem aos aspectos cooperativos do editor estão relacionadas ao modelo de cooperação a ser implementado pelo framework e que foram definidos no capítulo 4.

Funcionalidades de um Editor de Diagramas

A funcionalidade de um editor cooperativo para edição de diagramas é similar à funcionalidade de um editor comum de diagramas. Para isto, o editor deve permitir a manipulação dos elementos que compõem a metodologia utilizada, ou seja, deve-se permitir a inclusão e exclusão de classes, relacionamentos, atributos, métodos, etc. Além disso, o editor deve utilizar alguma forma de *representação externa*[VL90] do diagrama, que pode

ser na forma de código fonte em uma linguagem de programação ou em um meio estável (arquivo, banco de dados, etc.) para que o mesmo possa ser recuperado posteriormente.

Um editor de diagramas deve ser extensível de modo a permitir que outras metodologias de modelagem possam ser utilizadas. Rumbaugh[Rum95] define os componentes de uma metodologia como sendo um conjunto de conceitos de modelagem, uma notação para a representação destes conceitos, um processo e um conjunto de dicas e regras empíricas para otimizar o desenvolvimento. O editor fornecerá apoio apenas aos conceitos de modelagem (através do meta-modelo) e a notação da metodologia. O meta-modelo também pode ser usado para se verificar a consistência do diagrama quando novos elementos são adicionados, pois as restrições da metodologia são definidas no meta-modelo.

As *versões* do diagrama correspondem aos diferentes diagramas que foram sendo registrados pelo autor no decorrer do tempo. O registro das versões é importante, pois mostra a evolução dos diagramas no decorrer do tempo, fornecendo indicações do raciocínio e argumentações que levaram à produção final do diagrama.

A criação de uma *lista de ações*[FcdM94] dos usuários para registrar a evolução do diagrama é outro mecanismo desejável no editor. Enquanto as versões permitem registrar a evolução do diagrama como um todo, a lista de ações permite o registro da evolução de cada uma das versões.

5.3 Modelagem do Framework

Nesta seção é descrita a modelagem do editor. Na seção 5.3.1 é apresentada a modelagem utilizando a metodologia OMT[RBP+91]. Em seguida, na seção 5.3.2 os pontos adaptáveis do framework são identificados e os metapadrões adequados à flexibilidade exigida são aplicados. Na seção 5.3.3 utiliza-se a abordagem de padrões de projeto para oferecer flexibilidade a outros aspectos do framework.

A utilização dos metapadrões e dos padrões de projeto não foi feita na sequência apresentada neste capítulo. Na verdade os pontos adaptáveis do framework foram identificados, e se existisse um padrão de projeto adequado para este ponto, este padrão era utilizado, caso contrário era escolhido o metapadrão mais adequado. A sequência abaixo foi escolhida apenas por motivos didáticos.

5.3.1 Modelagem utilizando a Metodologia *OMT*

A modelagem apresentada neste trabalho consiste na construção do modelo de objetos do editor e baseia-se no trabalho de Tutumi[Tut96]. Uma sequência de passos para a construção do modelo de objetos é sugerida pela OMT, encontrando-se detalhada em [RBP+91, 152–169]. Após a aplicação destes passos os modelos de objetos das Figuras

5.1 e 5.2 foram obtidos.

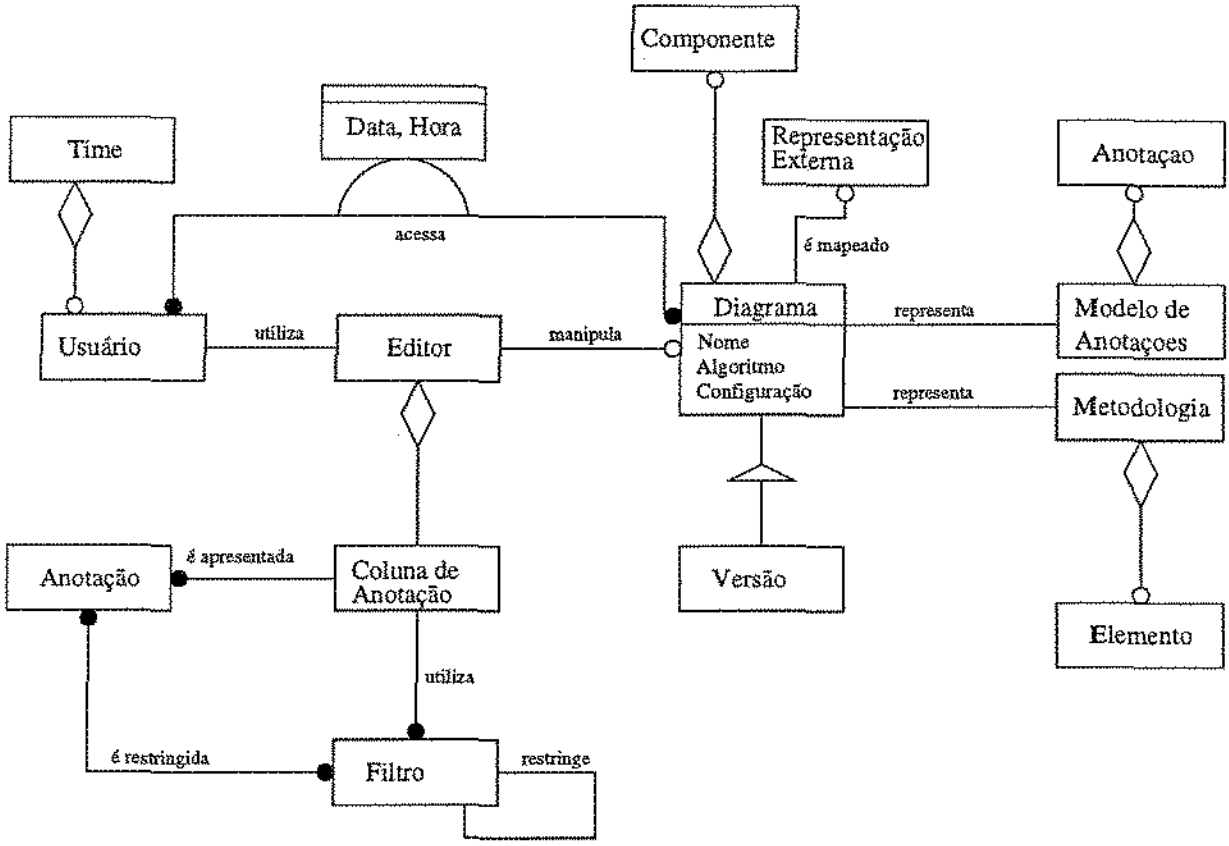


Figura 5.1: Modelo de Objetos do Editor

Um usuário utiliza o editor, que por sua vez manipula um conjunto de diagramas. Um usuário pode acessar diferentes diagramas, e este momento (data e hora) de acesso deve ser registrado. Cada diagrama é formado por um conjunto de componentes e possui associados uma lista de ações, um nome, um algoritmo e uma configuração.

Um diagrama pode ser mapeado para diferentes representações externas, porém ele é gerado a partir de uma única metodologia. Uma metodologia é composta por notação e meta-modelo, e cada um deles, por sua vez, é formado por um conjunto de elementos. Um editor possui uma coluna de anotações, que utiliza filtros para apresentar as anotações.

Os usuários do editor podem ser autores ou revisores. Cada usuário do editor possui um nome e pode inserir componentes no editor, sendo necessário o registro da data e hora desta inserção. Um componente pode ser um elemento da metodologia ou uma anotação. Uma anotação possui um escopo de visibilidade que é composto por uma lista de usuários. Uma anotação pode ser de três tipos diferentes: comentário; proposta, que é formada por outras anotações; e substituição, que é formada por um conjunto de elementos da metodologia. Uma anotação pode depender fortemente de outra anotação.

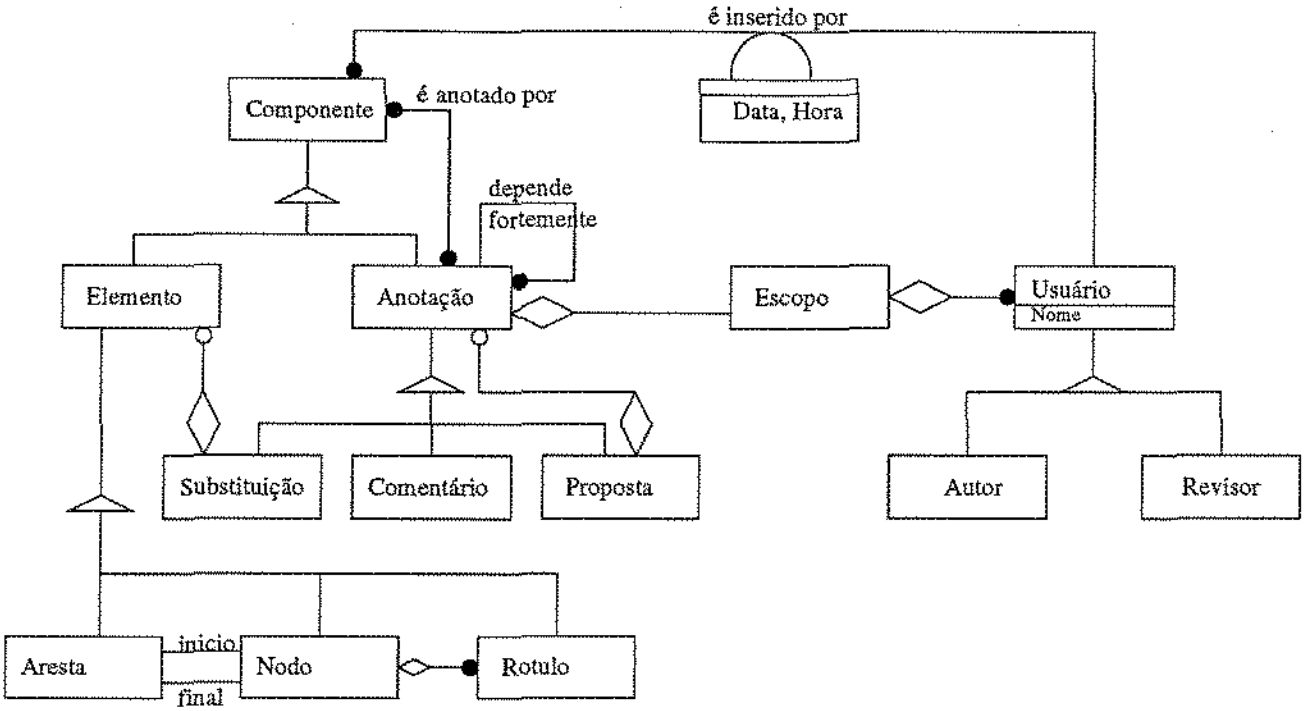


Figura 5.2: Modelo de Objetos do Editor (continuação)

5.3.2 Modelagem utilizando Metapadrões

Pontos Adaptáveis

Esta seção descreve os pontos adaptáveis do framework ABCDE. A flexibilidade que estes pontos requerem é avaliada e o metapadrão adequado é aplicado ao modelo de objetos das Figuras 5.1 e 5.2 para tornar o framework flexível. Os pontos adaptáveis são:

- **Modelo de Cooperação**

O modelo de cooperação dos editores instanciados a partir do framework consiste, basicamente, no conjunto de anotações utilizado. Este é o principal ponto adaptável do framework, pois é através deste modelo que editores cooperativos para outras atividades serão criados. Para que isso ocorra, o framework deve ter a capacidade de alterar o conjunto de anotações utilizado no modelo de cooperação. Isto pode ser obtido através de dois metapadrões distintos: Unificação[Pre94a, 124] e 1:1 Conexão[Pre94a, 124]. A escolha de um dos metapadrões baseia-se nas vantagens e desvantagens oferecidas por cada um deles. O metapadrão Unificação não permite a modificação do modelo de anotações em tempo de execução, pois os métodos template e hook estão situados na mesma classe. Apesar disso, ele possui uma implementação mais eficiente, uma vez que não precisa delegar responsabilidades a outros objetos. Isto é feito pelo metapadrão 1:1 Conexão que,

no entanto, permite a modificação em tempo de execução do modelo de anotações. O metapadrão escolhido foi o metapadrão Unificação, pois o autor define o modelo de cooperação que o editor irá apoiar durante o processo de construção do editor. Desta forma, não é necessária a modificação do modelo em tempo de execução. Figura 5.3 apresenta o metapadrão Unificação aplicado a classe Editor. O método *constructor* da classe Editor faz uma chamada ao método *criarModeloDeAnotações()* que retorna uma instância da classe que define o modelo de cooperação utilizado.

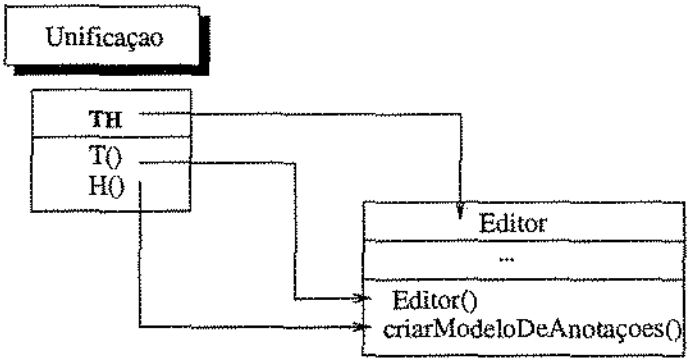


Figura 5.3: Metapadrão Unificação[Pre94a, 124] aplicado na classe Editor

• **Metodologia de Desenvolvimento de Software**

O metapadrão Unificação também é utilizado no projeto do framework visando permitir que a metodologia de desenvolvimento de software instanciada nos editores possa ser modificada. Assim, o usuário do framework ao instanciar um editor cooperativo a partir do ABCDE pode definir uma metodologia diferente da metodologia originalmente utilizada (UML). Esta metodologia também é definida pelo usuário do framework durante o processo de construção do editor, por isso ela também não precisa ser modificada em tempo de execução. Figura 5.4 apresenta o metapadrão Unificação[Pre94a, 124] aplicado a classe Editor. O método *constructor* da classe Editor faz uma chamada ao método *criarMetodologia()* que retorna uma instância da classe que define a metodologia utilizada.

• **Filtros de Anotações**

As anotações apresentadas na coluna de anotações são selecionadas através de filtros que restringem estas anotações. Pode-se imaginar vários tipos de filtros, como por exemplo, filtros de autoria, que apresentam somente anotações criadas por um determinado coautor; filtros temporais, que apresentam anotações inseridas após uma determinada data, etc. No entanto, novos filtros podem ser necessários dependendo do modelo de anotações utilizado, logo este aspecto também deve ser mantido flexível. Além disso, filtros podem ser combinados para restringir ainda mais as anotações a serem apresentadas. Por exem-

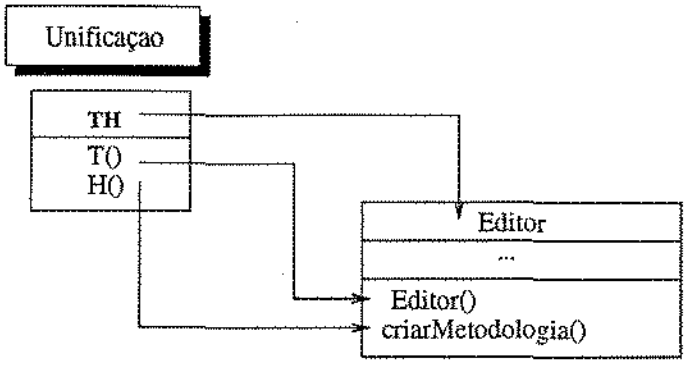


Figura 5.4: Metapadrão Unificação[Pre94a, 124] aplicado na classe Editor

plo, um usuário poderia aplicar um filtro de autoria e sobre o seu resultado aplicar um filtro temporal. Isto pode ser obtido através do metapadrão 1:N Conexão[Pre94a, 124], que também permite que os filtros utilizados sejam modificados em tempo de execução. Figura 5.5 apresenta este metapadrão aplicado no projeto do framework.

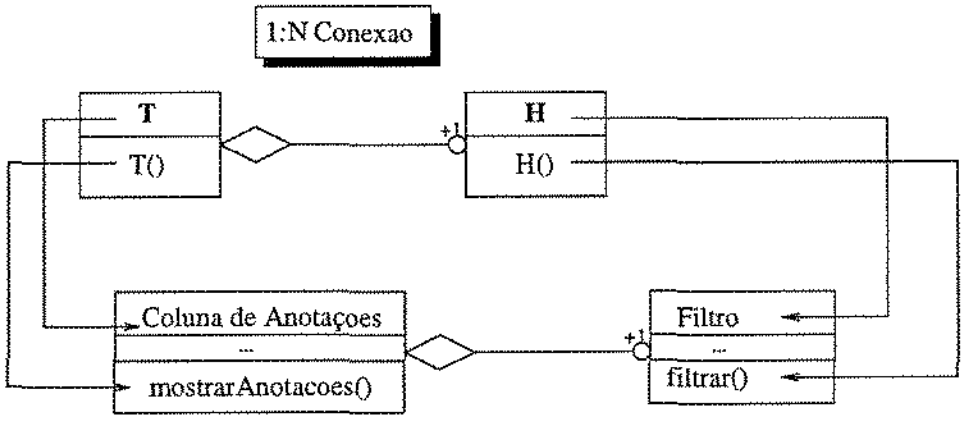


Figura 5.5: Metapadrão 1:N Conexão[Pre94a, 124] aplicado nas classes Filtro e Coluna de Anotações

Pontos Fixos do Framework

A metodologia de projeto orientado a pontos adaptáveis de Pree enfoca principalmente os pontos adaptáveis do framework, uma vez que serão estes pontos que fornecerão ao framework a flexibilidade necessária para que ele possa se moldar à diferentes aplicações. Entretanto, os pontos fixos de um framework também são importantes, pois correspondem às partes comuns das aplicações do domínio do framework. Estas partes devem ser implementadas diretamente no framework, de modo a permitir que elas sejam reutilizadas

por usuários do mesmo sem qualquer esforço adicional. Desta forma, é possível oferecer automaticamente uma série de serviços aos usuários, sem que os mesmos precisem se preocupar com sua implementação. A identificação e implementação destes pontos fixos foi feita e encontra-se sumarizada abaixo, na forma de uma lista de serviços oferecida automaticamente pelo framework ABCDE:

- Informações sobre percepção (seção 2.4). Como por exemplo, o momento em que ocorreu a última modificação de um diagrama, o último usuário que acessou cada diagrama, quais as modificações ocorridas em um diagrama após o último acesso de cada usuário, etc. Este tipo de informação ajuda a coordenação das atividades dos usuários, evitando o trabalho redundante ou conflitante[DB92];
- Tratamento de eventos para a apresentação de anotações;
- Gerência de arquivos e da interface com o usuário para o armazenamento e recuperação de diagramas;
- Gerência de arquivos e da interface com usuário para o armazenamento e recuperação de informações sobre diagramas, como descrição, data de criação, autor, etc; e
- Controle de acesso dos usuários. Somente usuários autorizados pelo gerente do diagrama podem acessá-lo.

5.3.3 Modelagem utilizando Padrões de Projeto

O projeto do editor também pode ser flexibilizado com a utilização de padrões de projeto. A utilização destes padrões não implica em um projeto menor, pelo contrário, freqüentemente a utilização destes padrões torna o projeto maior[GHJV94]. No entanto, esta desvantagem pode ser tolerada se considerarmos o grau de flexibilidade e reutilização obtido.

Abaixo são descritos alguns aspectos do projeto do framework que devem ser mantidos flexíveis, assim como os respectivos padrões de projeto utilizados:

• Algoritmos para Desenho Automático de Diagramas

A utilização de algoritmos para o redesenho de diagramas resolve o problema da visualização de anotações discutido no capítulo 4. No entanto, alguns desses algoritmos modificam o diagrama inteiro causando uma desorientação no usuário. Além disso, a implementação de tais algoritmos freqüentemente é custosa e demorada. Assim, deseja-se permitir que tais algoritmos possam ser facilmente modificados. Isto pode ser obtido

através da utilização do padrão Estratégia[GHJV94, 315] que cria uma classe para representar cada algoritmo utilizado pelo editor.

Figura 5.6 apresenta o modelo de objetos deste padrão aplicado ao projeto do framework. A classe Editor mantém uma referência para uma instância de uma subclasse da classe abstrata Algoritmo. Esta instância será responsável pelo redesenho do diagrama, pois os métodos relacionados a este serviço lhe são delegados pela classe Editor. Este padrão também permite que os algoritmos sejam modificados em tempo de execução, oferecendo ao usuário a possibilidade de escolher o algoritmo mais adequado. Em outras palavras, o usuário pode escolher o algoritmo que cause a menor desorientação possível.

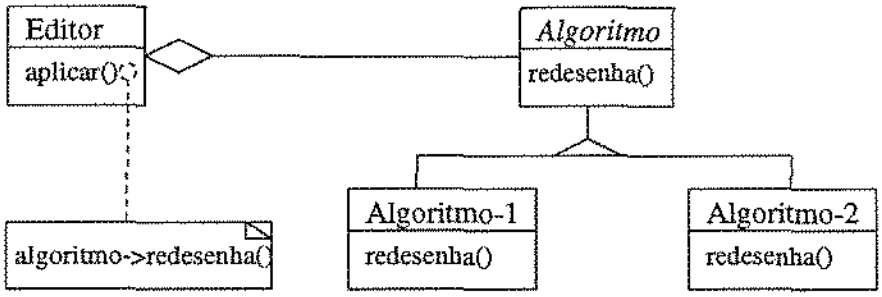


Figura 5.6: Utilização do padrão Estratégia[GHJV94, 315] para Algoritmos de Redesenho de Diagramas

• **Separação entre Modelo e Visão**

A separação entre modelo e visão em uma aplicação é desejável, uma vez que permite que classes representativas do modelo e/ou da visão possam ser reutilizadas independentemente. O modelo e a visão são dois aspectos de uma mesma aplicação que precisam estar sempre consistentes, ou seja, no momento em que ocorre uma mudança nos dados do modelo, a visão precisa ser notificada para que ela possa refletir tal modificação. O padrão Observador[GHJV94, 293] permite que isto seja feito, pois encapsula esses dois aspectos em objetos diferentes. Por exemplo, teremos uma classe chamada ClasseUML que registra os dados de uma classe criada no editor. Esta classe possui uma visão que é implementada pela classe VClasseUML (*Visual* ClasseUML), que é responsável por apresentar a ClasseUML segundo a notação da metodologia UML. Com esta separação, pode-se modificar a forma de apresentação dos conceitos da metodologia pela implementação de novas visões. Isto permite que o framework possa ser estendido para apoiar outras notações para desenvolvimento de software. Figura 5.7 ilustra o padrão Observador aplicado a classe Elemento. Cada Elemento possui no mínimo um VElemento equivalente que será responsável pela sua apresentação. Um Elemento também pode possuir mais de um VElemento associado. Isto é desejável, pois permite que diferentes visões do mesmo modelo possam ser criadas e todas elas são mantidas consistentes.

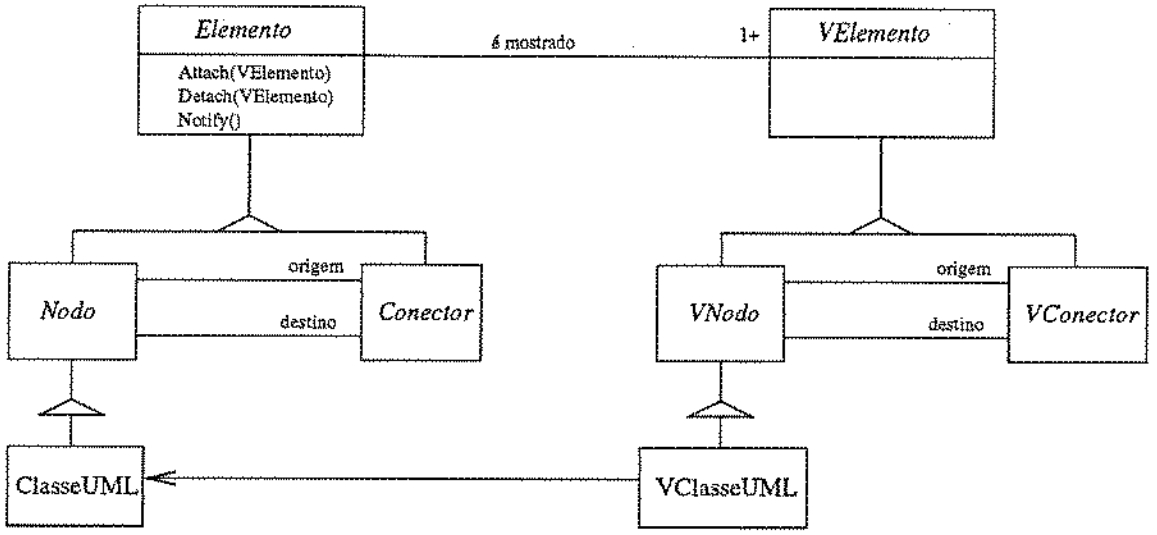


Figura 5.7: Padrão Observador[GHJV94, 293] associando Elemento a VElemento

O projeto do ABCDE foi feito visando oferecer condições de modelar qualquer notação utilizada em uma metodologia de desenvolvimento de software, desde que ela possa ser modelada como um grafo. Para isto, a classe Elemento possui as subclasses Nodo e Conector. Conector é uma classe abstrata que modela o comportamento de classes que representam relacionamentos entre dois objetos, que por sua vez, são instâncias de subclasses da classe Nodo. Por exemplo, uma classe da metodologia UML é modelada como uma subclasse da classe Nodo, enquanto que os relacionamentos existentes na UML são modelados como subclasses da classe Conector.

É necessário manter a consistência entre nodos e conectores, ou seja, a remoção de um nodo implica *necessariamente* na remoção de seus conectores, pois geralmente uma metodologia não permite que um relacionamento seja criado sem que ele esteja associado a pelo menos duas entidades. Outra consistência que precisa ser mantida ocorre quando um nodo é movido pelo diagrama. Neste caso, os conectores associados também devem ser movidos. Estas dependências podem ser mantidas através da utilização do padrão *Dependents*[GHJV94, 315][2]. Neste caso, qualquer objeto que é instância de uma subclasse da classe Nodo possuirá uma lista de objetos Conectores associados, tal que, quando o Nodo é movido(removido), este objeto notifica os seus Conectores associados movendo-os(removendo-os) também. Figura 5.8 ilustra o padrão *Dependents* aplicado as classes Nodo e Conector.

[2]Na verdade, este é o mesmo padrão Observador utilizado na resolução de um problema diferente, porém similar.

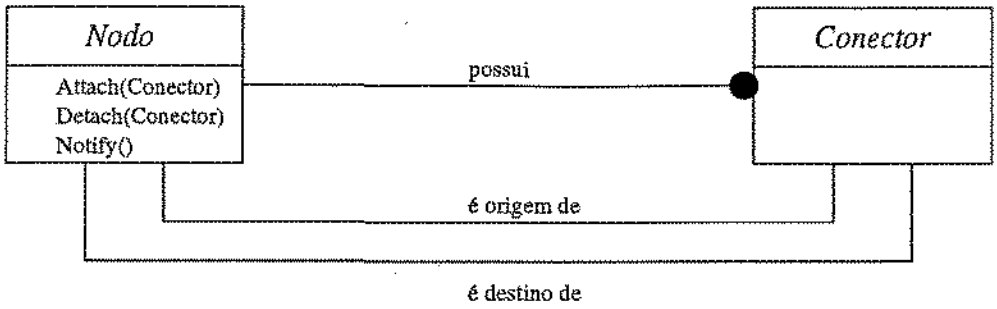


Figura 5.8: Padrão *Dependents*[GHJV94, 293] associando Nodo a Conector

• **Consistência entre Modelo e Visão**

A hierarquia de classes da Figura 5.7 pode ser dividida em duas hierarquias paralelas onde uma classe de uma hierarquia delega responsabilidades a uma classe da outra hierarquia. Por exemplo, a ClasseUML delega a responsabilidade de ser apresentada à classe VClasseUML. Para que as duas hierarquias possam estar devidamente conectadas utilizou-se o padrão *Factory Method* [GHJV94, 107]. Este padrão define na ClasseUML um método *factory* chamado *criarVisão()* que é responsável por instanciar um objeto que corresponderá à visão da classe. Este objeto deve ser obrigatoriamente instanciado a partir de uma das subclasses concretas existentes na hierarquia originada na classe abstrata VElemento. Figura 5.9 apresenta a estrutura deste padrão instanciada no framework ABCDE. Quando o método *criarVisão()* é chamado, este padrão garante que o modelo estará associado à sua visão correspondente.

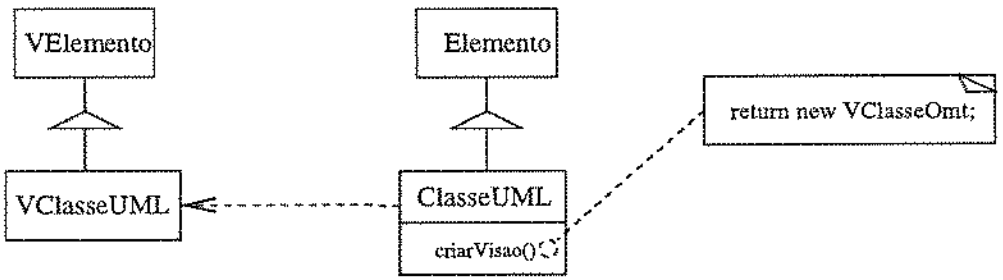


Figura 5.9: Padrão *FactoryMethod*[GHJV94, 107] utilizado para conectar as classes do Modelo às classes da Visão.

• **Representação das Anotações**

A descrição do modelo apresentada no capítulo 4, define uma proposta como sendo uma anotação composta por um conjunto de outras anotações. O editor deve permitir que o usuário trate este conjunto de anotações de maneira uniforme, de tal modo que se o usuário deseja, por exemplo, visualizar uma proposta, todas as anotações que a compõem também serão visualizadas. O padrão *Composite*[GHJV94, 163] permite que componentes

simples possam ser unidos para formar agregados de componentes, adequando-se perfeitamente à hierarquia de anotações. O modelo de objetos descrevendo a hierarquia de anotações é apresentado na Figura 5.10. É interessante observar que a modelagem feita utilizando a metodologia *OMT* obteve o mesmo modelo de objetos que o da Figura 5.10. Isto ocorreu devido à existência de uma especificação precisa do modelo. No entanto, a modelagem feita utilizando-se o catálogo de padrões de projeto, ao formar um vocabulário comum entre projetistas, é mais informativa que a modelagem utilizando-se a OMT. Desta forma, as vantagens e desvantagens deste padrão são compreendidas pelos usuários do framework sem que seja necessário que eles tenham um conhecimento detalhado do modelo de cooperação. Por exemplo, a utilização do padrão *Composite* permite que novas anotações possam ser criadas e tratadas de maneira uniforme pelo usuário.

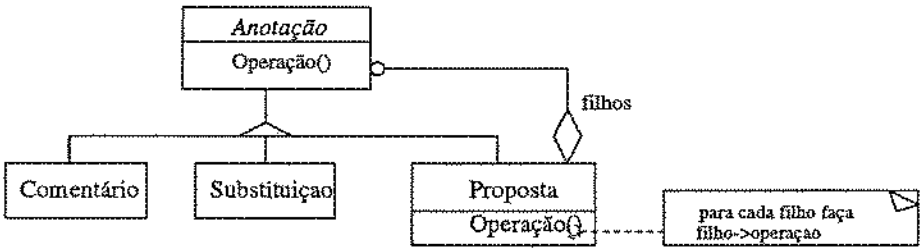


Figura 5.10: Padrão *Composite*[GHJV94, 163] na hierarquia de Anotações

- **Modificação do comportamento segundo a Ferramenta selecionada**

Geralmente, aplicações para edição de diagramas fornecem ferramentas (*tools*) para a execução de suas operações. Por exemplo, uma ferramenta para criação de relacionamentos permite que o usuário selecione a classe inicial, arraste o *mouse* até a classe final e então crie o relacionamento entre as duas classes[3]; uma ferramenta de seleção permite que os elementos do diagrama sejam selecionados para que operações possam ser aplicadas diretamente a todos eles; etc. Estas ferramentas normalmente são colocadas em *palletes* para que o usuário possa acessá-las.

A execução correta das atividades de cada uma das ferramentas requer que o editor seja capaz de modificar seu comportamento em tempo de execução conforme a ferramenta atualmente selecionada. Assim, elementos serão inseridos se uma ferramenta de criação está selecionada, elementos serão selecionados se a ferramenta de seleção está ativa, e assim por diante. Para permitir esta modificação de comportamento, o padrão Estado[GHJV94, 305] foi utilizado. Este padrão é utilizado no projeto do framework Unidraw[VL90] sendo chamado de *Tool*(Ferramenta). Uma vez que este nome é mais adequado ao contexto de edição de diagramas, ele será utilizado no restante desta dissertação.

[3]Supondo que este relacionamento seja permitido pelo metamodelo da metodologia.

A Figura 5.11 apresenta o modelo de objetos do framework com a utilização do padrão Ferramenta. Uma classe abstrata Ferramenta foi criada para modelar o comportamento comum das várias ferramentas, como por exemplo sua apresentação na *pallete* de elementos, sua seleção através de botões do *mouse*, etc. O objeto da classe Diagrama mantém uma referência para a Ferramenta atualmente selecionada e delega as operações de tratamento de eventos para ela. Por exemplo, se o botão esquerdo do *mouse* é pressionado sobre a área de desenho do Diagrama, a operação executada depende da Ferramenta atualmente selecionada. Quando o usuário seleciona outra Ferramenta, o objeto da classe Diagrama substitui a Ferramenta atual pela nova Ferramenta selecionada.

O comportamento específico das ferramentas é obtido através da criação de subclasses da classe Ferramenta que redefinirão os métodos de tratamento de eventos. Em especial, duas subclasses da classe Ferramenta foram criadas. A primeira, chamada de FerramentaDeSeleção, permite selecionar conjuntos de elementos do diagrama para que sejam aplicadas operações sobre todo este conjunto. Ela foi criada porque as diversas ferramentas que permitem anotar elementos do diagrama devem permitir a seleção de conjuntos de elementos. Desta forma, este comportamento comum é definido uma única vez e reutilizado por estas ferramentas que são subclasses desta classe. A segunda, chamada FerramentaParaAresta, é uma classe abstrata que modela o comportamento comum das ferramentas que possuem o objetivo de criar relacionamentos entre dois elementos do diagrama. Todas as ferramentas para manipulação dos relacionamentos da metodologia UML são exemplos destas ferramentas. O comportamento comum destas ferramentas é a apresentação de mensagens de erro quando o ponto inicial do relacionamento não contém nenhuma classe, a manipulação do *mouse* quando este é arrastado até a segunda classe, etc.

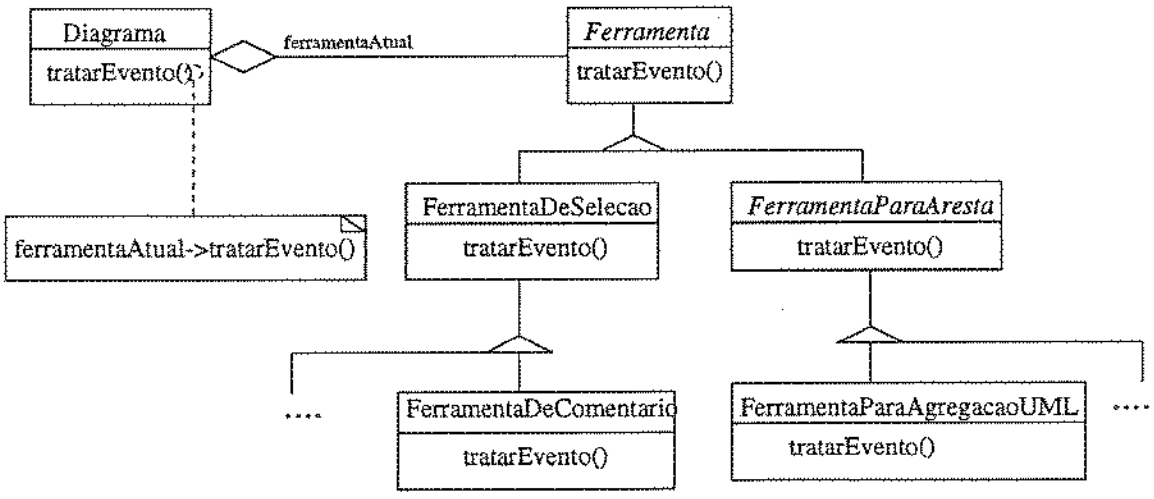


Figura 5.11: Padrão Ferramenta[GHJV94, 305] aplicado ao projeto do framework.

5.4 Implementação e Utilização do Framework

Um protótipo inicial do framework ABCDE foi desenvolvido. Ele consiste de cerca de oitenta classes, distribuídas entre classes de aplicação e classes auxiliares, implementadas utilizando a linguagem de programação Java[Fla97]. As classes auxiliares são as classes construídas de modo a apoiar o projeto do framework, tais como, armazenamento e recuperação de diagramas, controle de acesso dos usuários, etc. Classes de aplicação são classes de alto nível que fatoram a estrutura comum do domínio do framework, ou seja, elas fatoram o comportamento comum de editores de diagramas cooperativos. Estas classes definem o modelo abstrato de execução dos editores construídos utilizando o ABCDE e juntas criam um editor cooperativo genérico.

A implementação atual do framework ABCDE não permite o acesso síncrono aos diagramas. Isto significa que dois usuários não podem acessar o mesmo diagrama ao mesmo tempo.

Figura 5.12 apresenta as principais classes do framework. Classes em *itálico* correspondem a classes abstratas e classes em **negrito** correspondem as classes que o usuário necessita conhecer para utilizar o framework. Estima-se que para a utilização do ABCDE o usuário necessite estender cerca de 10 a 12 classes e 30 métodos. Estes métodos seguiram o padrão de projeto para linguagens de programação proposto por Pree[Pre94a] (seção 3.2.3) que consiste em convenções no modo de atribuir nomes a classes, variáveis, métodos, etc. A utilização destas convenções visa facilitar o aprendizado e utilização do ABCDE.

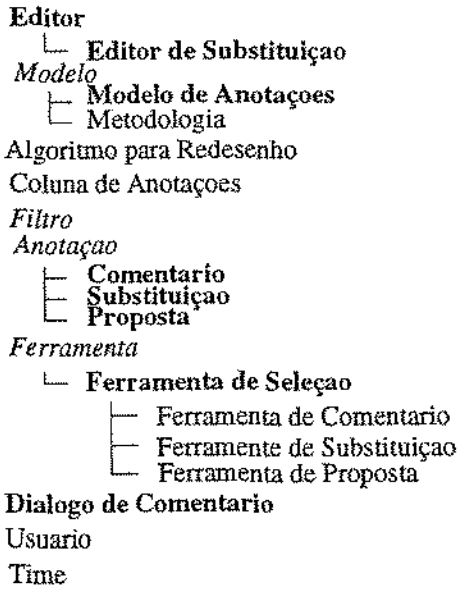


Figura 5.12: Classes de aplicação do framework ABCDE.

Figura 5.13 apresenta uma visão geral do framework ABCDE. Toda aplicação instanciada a partir do ABCDE possui exatamente um objeto da classe *Editor*. O passo inicial da execução de uma aplicação consiste em criar uma instância de uma subclasse de Editor e executar o método *show()*[4] neste objeto para que o fluxo de eventos seja iniciado. A configuração da metodologia e do modelo de cooperação implementados pelo Editor é feita através do metapadrão Unificação. Além das classes relacionadas ao modelo de cooperação, outras classes auxiliares também devem ser redefinidas.

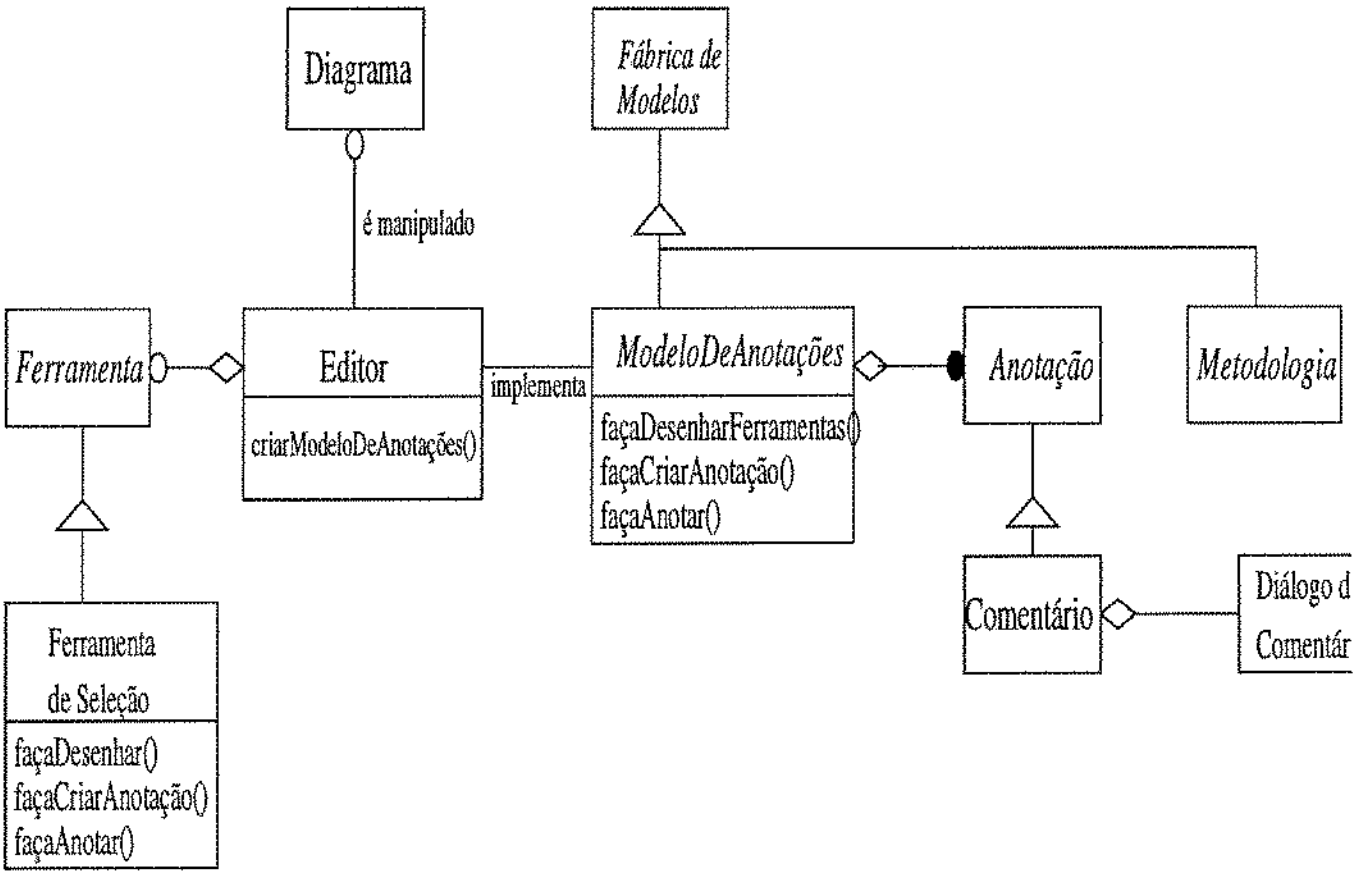


Figura 5.13: Instanciação de um Editor para Inspeção de Software a partir do framework ABCDE.

[4]Este método corresponde ao método da classe *Frame* do framework *AWT* que faz parte do conjunto de *packages* disponibilizados juntamente com a linguagem de programação Java.

5.5 ABCDE-Web: Edição cooperativa de diagramas na Internet

A utilização da Internet como meio para permitir a autoria cooperativa de artefatos não é uma abordagem nova. Por exemplo, Symposia[QV95] é um editor de hiper-textos cooperativo desenvolvido no INRIA que usa servidores WWW configurados para aceitar a operação **PUT** do protocolo HTTP como infra-estrutura para permitir a cooperação. Esta abordagem permite cooperação entre grupos dispersos, cujos membros podem estar em diferentes organizações, e até mesmo, diferentes países.

Nos últimos anos, alguns sistemas para o *desenvolvimento de software* na Internet começaram a surgir, como por exemplo, ICARO[LF96] e HiperDev[MXR97]. No entanto, ainda não existem muitos editores cooperativos de diagramas que possam ser utilizados na Internet. Esta seção apresenta o sistema ABCDE-Web que é um editor de diagramas cooperativo que pode ser utilizado na Internet e que foi instanciado a partir do framework ABCDE.

5.5.1 Descrição e Implementação do ABCDE-Web

ABCDE-Web é um sistema que apóia o desenvolvimento cooperativo de software através da Internet. O sistema apóia as fases de análise e projeto de sistemas através da edição cooperativa de diagramas. Ele foi desenvolvido a partir do framework ABCDE em uma arquitetura cliente-servidor utilizando recursos de comunicação existentes no JDK 1.1.

O modelo de cooperação do ABCDE-Web não foi modificado pois o objetivo desta implementação era testar critérios como usabilidade, operacionalidade e funcionalidade das aplicações que seriam instanciadas a partir do ABCDE. Além disso, este sistema contribui para o avanço do conhecimento científico na área de aplicações cooperativas na Internet, pois implementa um modelo de cooperação bastante flexível. Isto não é obtido pelos outros editores de diagramas cooperativos discutidos no capítulo 2, seção 2.6.

O cliente e o servidor do ABCDE-Web são descritos abaixo:

O servidor

O servidor atua como uma aplicação WWW padrão, servindo como um repositório para um conjunto de diagramas que foram criados cooperativamente. Ele gerencia o acesso dos usuários aos diagramas através de um esquema simples de autenticação usuário/senha. Uma vez que um usuário tenha sido identificado, todos os diagramas que ele pode acessar são apresentados. No entanto, ele não pode acessar a todos, pois alguns destes diagramas podem estar sendo acessados por outros usuários. Neste caso, é apresentada uma indicação do usuário que está acessando o diagrama e o momento inicial do acesso. O usuário que

criou o diagrama no servidor é o seu gerente. Ele especifica a lista de usuários que podem acessar o diagrama, ou pode permitir acesso irrestrito ao diagrama. O servidor também coleta as informações sobre percepção de cada diagrama, e é responsável por armazená-las e apresentá-las aos usuários.

Quando o usuário escolhe um diagrama, o servidor envia o cliente para o browser do usuário, envia o diagrama para este cliente e faz um *check out* do diagrama. Quando o usuário termina de editar o diagrama, o cliente envia o diagrama de volta para o servidor. O servidor atualiza a base de dados e faz o *check in* do diagrama, tornando-o disponível para os outros usuários que tenham permissão para acessá-lo. Figura 5.14 apresenta um exemplo de um acesso de um usuário usando o browser HotJava 1.1[5]. Neste exemplo existe um único diagrama no servidor. Este diagrama representa o padrão de projeto Observador[GHJV94] que foi utilizado no projeto do ABCDE.

Applet para Edição de Diagramas Cooperativa Baseada em Anotações

A^{2}BCDE (*Applet Annotation Based Cooperative Diagram Editor*) implementa o cliente da aplicação. Ele foi desenvolvido a partir do framework ABCDE e permite o acesso assíncrono aos diagramas. Optou-se pelo acesso assíncrono aos diagramas por duas razões. A primeira, porque o framework ABCDE, em sua implementação atual, não permite o acesso síncrono a diagramas. A segunda, porque em situações assíncronas o acoplamento entre o servidor e o cliente pode ser perdida: o servidor envia todo o diagrama para o cliente e recebe de volta todo o diagrama do cliente após a edição do usuário. Em situações síncronas todas as operações efetuadas por um usuário precisam ser imediatamente transferidas para o servidor que as executa, e então propaga-as para todos os outros clientes. Nós não estamos certos sobre como lidar com os atrasos comuns na Internet e nem com a perda de pacotes em uma arquitetura que requer que um grande volume de mensagens sejam enviadas entre cliente e servidor.

Figura 5.15 apresenta um exemplo de uma instância do editor cooperativo. O lado esquerdo da figura mostra a *pallete* de construção e de anotação onde os elementos da metodologia e as anotações definidas no modelo de cooperação são apresentados. Ao centro é apresentado o diagrama de classes do padrão de projeto Observador[GHJV94, 293] construído segundo a notação UML. O lado direito da figura apresenta a coluna de anotações que apresenta as anotações feitas pelos usuários que tem permissão de acesso ao diagrama. Cada tipo de anotação é apresentada com uma cor diferente. Dependência forte entre anotações é apresentada através de identação.

[5]No momento em que este trabalho estava sendo escrito, os browsers Netscape Navigator e Microsoft Internet Explorer não estavam oferecendo apoio completo ao JDK 1.1.

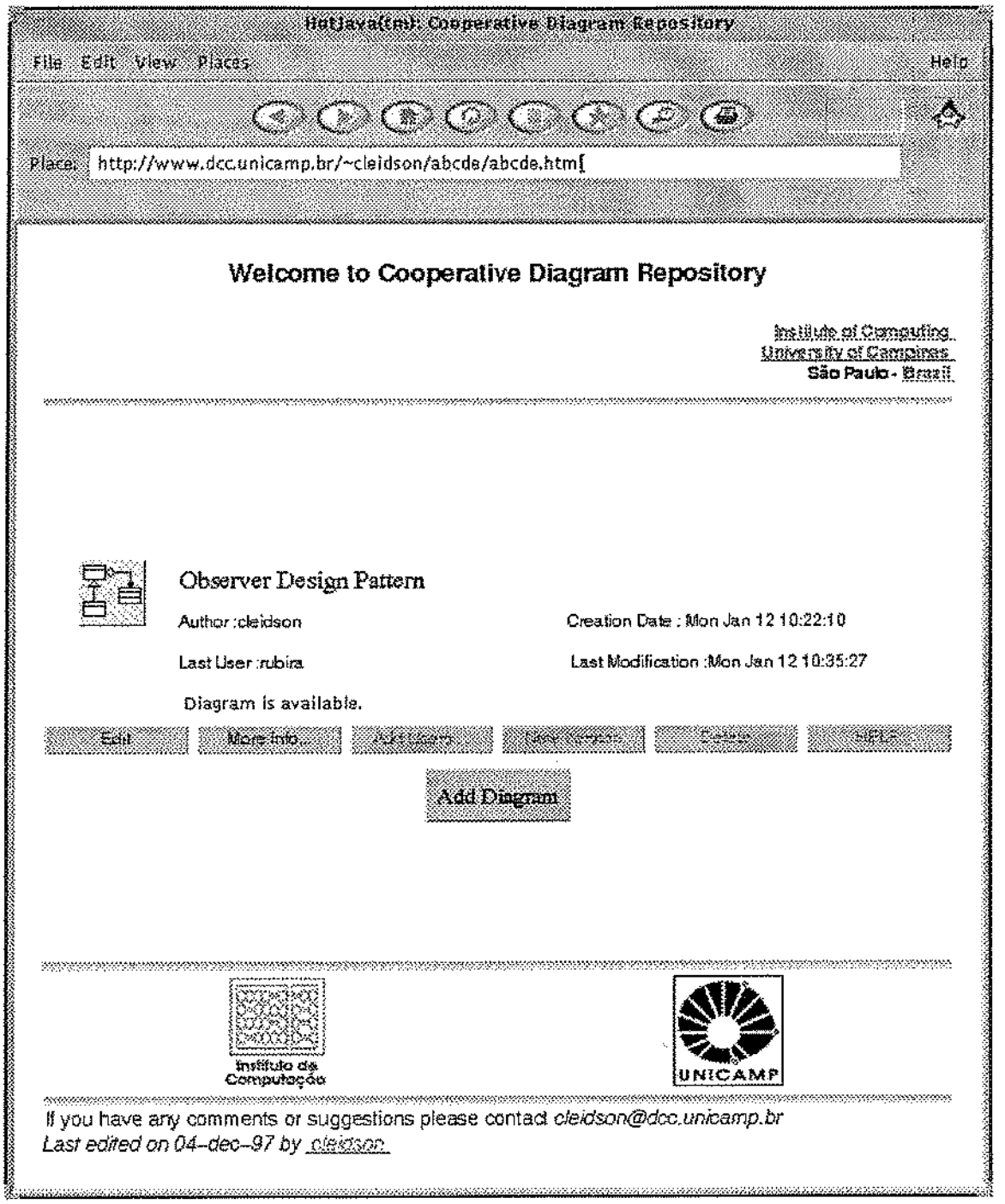


Figura 5.14: Exemplo de um usuário acessando o servidor através do browser HotJava 1.1.

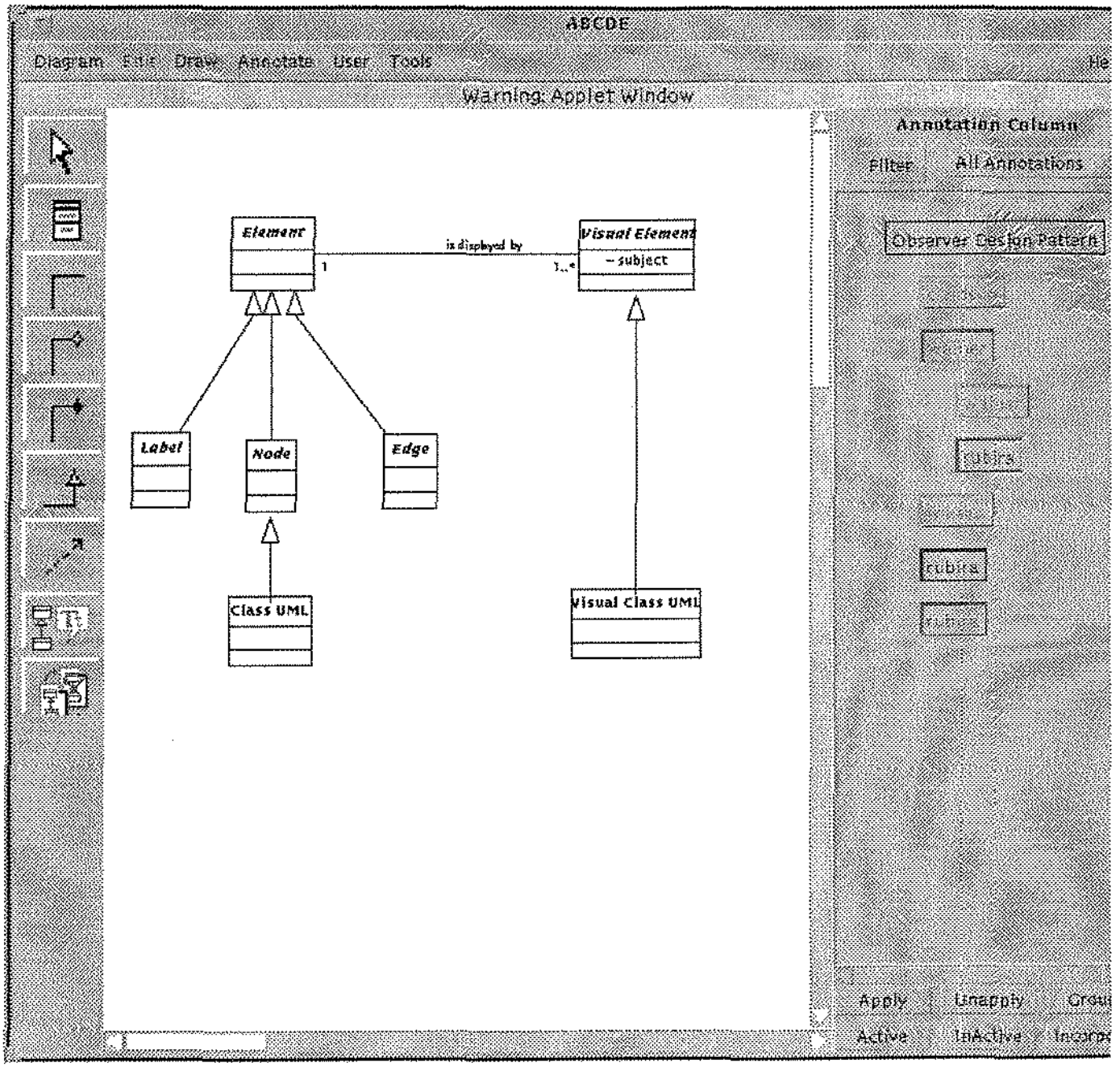


Figura 5.15: Exemplo do A²BCDE.

5.6 Resumo

Nós mostramos como um framework para editores de diagramas cooperativos pode ser construído através da utilização conjunta de padrões de projeto e metapadrões. O framework ABCDE (*Annotation Based Cooperative Diagram Editor*)[SRW98] foi obtido a partir da utilização da metodologia de projeto orientado a pontos adaptáveis[Pre94a] para desenvolvimento de frameworks. Esta metodologia propõe a utilização de metapadrões para tornar o projeto do framework flexível, no entanto, optou-se por utilizar também os padrões de projeto de Gamma et al.[GHJV94] pois os mesmos são mais concretos, fáceis de manusear e entender, e provêem um guia mais concreto para compreensão dos pontos adaptáveis de um framework[Sch96].

Mais especificamente, o ABCDE foi construído a partir de uma série de transformações feitas sobre o projeto de um editor de diagramas cooperativo. Este projeto foi obtido a partir das funcionalidades de um editor cooperativo qualquer de diagramas baseado em anotações. As transformações foram feitas utilizando-se metapadrões e padrões de projeto para se obter a flexibilidade necessária. A implementação do framework também foi descrita, assim como o sistema ABCDE-Web[SWR98] que é um editor de diagramas cooperativo que pode ser utilizado na Internet. Este sistema foi instanciado a partir do framework ABCDE. Ele foi desenvolvido visando avaliar critérios como usabilidade, operacionalidade e funcionalidade do framework ABCDE.

Capítulo 6

Conclusões e Extensões

Esta dissertação propôs um modelo de cooperação para o desenvolvimento cooperativo de software que baseia-se na utilização de anotações em diagramas. As anotações são uma forma de comunicar idéias ou opiniões sobre um documento, servindo como um mecanismo de comunicação entre os seus autores. O modelo proposto oferece apoio às duas principais formas de cooperação que ocorrem durante a atividade de autoria cooperativa: revisão e co-autoria. Além disso, ele é independente da metodologia utilizada para o desenvolvimento de software e pode ser facilmente estendido para apoiar outras atividades do processo de desenvolvimento de software. Ratificando tal afirmativa, apresentou-se duas extensões ao modelo para as atividades de inspeção de software e desenvolvimento de software com registro das argumentações sobre o projeto. As extensões foram apresentadas através da especialização das anotações do modelo original.

A segunda contribuição desta dissertação é o framework ABCDE (*Annotation Based Cooperative Diagram Editor*) que permite a construção de editores de diagramas cooperativos para a notação UML[BRJ97], pois implementa o modelo de cooperação definido. A construção do framework ABCDE foi feita utilizando-se a abordagem de projeto orientado a pontos adaptáveis proposta por Pree [Pre94a] em conjunto com os padrões de projeto de Gamma et *al.* [GHJV94] e sua implementação utilizando a linguagem de programação Java[1][Fla97]. O principal ponto adaptável do framework é o modelo de anotações implementado. Além disso, o sistema ABCDE-Web para edição cooperativa de diagramas na Internet foi desenvolvido a partir do framework. Ele contribui para o avanço do conhecimento científico na área de aplicações cooperativas na Internet, pois implementa o modelo de cooperação previamente definido que é, conforme mostrado, bastante flexível.

O resto do capítulo está organizado da seguinte forma. Seção 6.1 apresenta os resultados preliminares da aplicação de testes de utilização de um editor instanciado a partir do framework ABCDE. Seção 6.2 apresenta algumas limitações existentes na dissertação

[1]Java é uma marca registrada da Sun Microsystems, Inc.

e a seção 6.3 apresenta algumas sugestões de extensões que podem ser propostas a esta dissertação.

6.1 Resultados Preliminares

Alunos do curso de Bacharelado em Ciência da Computação da Universidade Federal do Pará (UFPa) foram submetidos a testes de utilização de um editor que foi instanciado a partir do ABCDE. O grupo consistia em cinco alunos sem experiência na utilização de ferramentas CASE e desenvolvimento de software (orientado a objetos ou estruturado). O editor foi utilizado de maneira convencional, ou seja, *sem* considerar seus aspectos cooperativos (anotações), por isso, não foi necessário redefinir seu modelo de cooperação.

A tarefa dos alunos era construir diagramas de classes que correspondiam à análise orientada a objetos de dois problemas diferentes, cuja especificação foi fornecida. O objetivo do teste era avaliar os pontos fixos do framework, pois estes pontos serão herdados pelos editores instanciados a partir do ABCDE. Funcionalidade, usabilidade e operacionalidade do editor foram testadas, atribuindo-se notas entre um (1) e cinco (5) a critérios como por exemplo, adequação, corretude, robustez, disponibilidade de recursos para aprendizado, adequação da interface, etc. O formulário completo de avaliação utilizado pelos alunos encontra-se no Apêndice A.

Os principais resultados obtidos são sumarizados abaixo:

- Aumento na área de edição de diagramas. Isto ocorre devido a dificuldade observada pelos usuários em movimentar-se na mesma através das barras de rolagem, pois elas fazem com que o desempenho seja consideravelmente prejudicado.

- Diminuição da coluna de anotações. Uma vez que as anotações não foram utilizadas neste teste, não existe a necessidade de apresentar a coluna de anotações. Isto sugere que o usuário deve poder escolher, durante a utilização da ferramenta, se deseja ou não que a coluna de anotações seja apresentada. Desta forma, evitaríamos possíveis distrações do usuário e permitiríamos um aumento na área de edição de diagramas.

- Implementação de outros conceitos da notação UML que não foram implementados; em especial, atributos de ligação.

- Maior utilização do teclado. Permitir que operações realizadas unicamente através do *mouse* também possam ser realizadas com auxílio do teclado.

6.2 Limitações

A abordagem proposta nesta dissertação apresenta algumas limitações, tanto no aspecto conceitual, quanto na atual implementação do framework.

As limitações no aspecto conceitual referem-se ao modelo de cooperação e são:

- A falta de uma avaliação efetiva do modelo de cooperação. Até o presente momento o modelo de cooperação ainda não foi avaliado para verificar se ele é adequado ao desenvolvimento cooperativo de software. Espera-se que a observação de várias situações de projeto cooperativo, assim como o *feedback* dos usuários, sirvam como auxílio para o entendimento das características da atividade de desenvolvimento cooperativo de software.

- A falta de uma avaliação precisa sobre algumas decisões tomadas durante o projeto do framework. Durante o projeto do framework algumas decisões de projeto tiveram de ser tomadas para que sua implementação pudesse ser concluída. Por exemplo, quando uma anotação torna-se inativa, o que deve ser feito com as anotações que dependem fortemente dela? Elas devem tornar-se inativas também ou simplesmente devem adicionar esta anotação à sua própria estrutura? A solução adotada foi adicionar a anotação que se tornou inativa às anotações que dependem fortemente dela. No entanto, esta questão somente poderá ser respondida corretamente a partir da observação de usuários em situações reais de projeto cooperativo.

- O modelo de cooperação parece se adequar à diversas famílias de diagramas, não apenas a diagramas de análise e projeto orientado a objetos. No entanto, para diagramas que apresentem elementos internos a outros elementos, como por exemplo os *statecharts* de Harel[Har87], o modelo parece não ser adequado. Isto ocorre porque nestes diagramas se uma anotação é feita sobre o elemento mais externo, não é evidente se a anotação também deve ser válida para os elementos mais internos, mesmo que, freqüentemente, nestes diagramas outras características do elemento mais externo possam ser propagadas aos elementos internos.

A implementação atual do framework ABCDE ainda precisa ser melhorada para que possa ser distribuída a usuários em geral. Essencialmente, o framework deve ser modificado para implementar completamente o modelo de cooperação descrito no capítulo 4. Algumas idéias deste modelo ainda não se encontram implementadas, embora a atual implementação do ABCDE tenha sido feita levando-as em consideração. Como por exemplo, algoritmos para redesenho automático de diagramas, controle de versões, mecanismos de *Undo* e possibilidade de criar diferentes configurações para um diagrama. Em outras

palavras, muitas classes desenvolvidas para a versão atual do framework ainda precisam ser modificadas para corresponder completamente ao projeto final descrito no capítulo 5.

Além disso, esta implementação utiliza a linguagem Java, que é uma linguagem interpretada, e baseia-se na utilização de padrões de projeto e meta-padrões, que se baseiam na utilização maciça do mecanismo de delegação de mensagens. Estes dois fatores impõem um desempenho relativamente pobre[2]. No entanto, esta ineficiência pode ser tolerada se considerarmos que a utilização de padrões de projeto e meta-padrões oferecem um alto grau de flexibilidade ao framework, e que Java é uma linguagem independente de plataforma, o que fornece um alto grau de portabilidade ao framework.

6.3 Extensões

Diversas extensões podem ser propostas a esta dissertação. A principal seria a definição de modelos de cooperação específicos para atividades do desenvolvimento de software, em especial, inspeção de software e decisões de projeto. Modelos de cooperação para estas atividades foram parcialmente definidos na seção 4.5, entretanto estas atividades precisam ser estudadas mais detalhadamente para a especificação de modelos de cooperação mais adequados.

As extensões podem ser divididas em extensões ao modelo de cooperação e extensões ao framework. As extensões ao modelo de cooperação são:

- Avaliação do Modelo para Diagramas tipo *Statecharts*
  Uma avaliação mais precisa, e se necessário, a adequação do modelo de cooperação para famílias de diagramas que apresentem elementos internos a outros elementos como os *statecharts* de Harel[Har87]. Desta forma, poder-se-ia construir ferramentas cooperativas para a construção de outros tipos de diagramas que ainda não são apoiados.

- Criação de Páginas de Argumentação
  Uma possível extensão a ser feita ao modelo é a criação de páginas de argumentação a partir das anotações existentes no diagrama. Estas páginas seriam criadas a partir das anotações inativas por uma pessoa responsável pela documentação da discussão ocorrida durante o projeto. Estas páginas poderiam conter informações textuais, assim como as partes do diagrama que foram anotadas, permitindo que a discussão fosse armazenada de maneira estruturada. Elas poderiam ser armazenadas em um banco de dados, de tal forma que os usuários pudessem executar consultas visando reutilizar a experiência obtida em projetos anteriores. A criação de páginas de argumentação a partir das anotações, foi feita inicialmente no sistema XNetwork[RS92].

[2]Esta característica foi observada pelos usuários durante a realização dos testes.

- Criação de relacionamentos entre anotações
  Uma possível extensão do modelo de cooperação é a criação de relacionamentos entre as anotações. Desta forma, a etapa de incorporação poderia ser simplificada e decisões relacionadas poderiam ser expressas mais adequadamente. Por exemplo, poder-se-ia utilizar um mecanismo de generalização-estruturação para relacionar duas anotações. Supondo que uma anotação A generalize outra anotação B. Se a anotação B tornar-se inativa, isto significa que a anotação A também deve se tornar inativa. Um outro exemplo de relacionamento entre duas anotações seria o de substituição. Este relacionamento justifica-se porque assim como podem existir anotações que devem ser aplicadas em conjunto, também podem existir anotações que expressam idéias contraditórias. Se uma destas anotações é aplicada, então as outras automaticamente tornam-se inativas. Estes relacionamentos são criados por qualquer um dos co-autores do diagrama.

- Criação de Alternativas
  Durante a fase de incorporação, o autor pode definir *alternativas*[MM93] de modelagem do diagrama. Cada alternativa expressa um conjunto de decisões de modelagem que se relacionam, porém não possuem uma anotação do tipo proposta para reuni-las. Em outras palavras, uma alternativa consiste em um conjunto de anotações aplicadas ao diagrama. Estas alternativas seriam armazenadas temporariamente, porém não alterariam o diagrama original. Elas serviriam como um mecanismo para auxiliar o processo de exploração do espaço de possíveis diagramas existentes quando diversas anotações podem ser aplicadas simultaneamente. Ao final, o autor pode escolher uma das alternativas existentes tornando-a o novo diagrama.

As possíveis extensões ao framework ABCDE são:

- Reestruturação do framework ABCDE
  O desenvolvimento de um framework geralmente é uma atividade iterativa que envolve várias iterações sobre o projeto original. Em cada iteração a estrutura de classes do framework é reorganizada, sendo criadas novas superclasses abstratas a partir de conjuntos de classes concretas, métodos e/ou variáveis são acrescentados ou reposicionados em classes diferentes, etc[Cam97].

  No projeto atual do framework ABCDE, a forma como são indicados os elementos do diagrama que foram anotados é um ponto fixo, ou seja, esta é uma característica que não pode ser modificada. Esta indicação é feita através da modificação das cores dos elementos anotados. Entretanto, percebeu-se que esta característica deve ser flexível, pois os usuários do framework podem desejar diferentes formas de fazer esta indicação. Logo, esta característica deve tornar-se um ponto adaptável, tal que, cada usuário do framework possa modificar a forma como esta indicação é feita.

Sugere-se a utilização de um sistema de gerência de banco de dados orientados a objetos (SGBDOO) para a implementação do controle de versões. SGBDOO's fornecem um mapeamento direto entre o modelo de objetos utilizado na implementação e o modelo de dados do banco de dados. A utilização de um SGBDOO também permite que outros serviços possam ser oferecidos pelos editores construídos a partir do framework, como atomicidade das transações, recuperação, controle de concorrência, controle de acesso, etc.

Além disso, conforme descrito na seção 6.2, o framework também deve ser modificado para implementar completamente o modelo de cooperação descrito no capítulo 4.

- Suporte a padrões de projeto e meta-padrões
  Assim como o projeto do framework ABCDE foi desenvolvido utilizando-se padrões de projeto e meta-padrões, outros frameworks, toolkits ou aplicações[GHJV94] podem se beneficiar de sua utilização. Desta forma, uma extensão natural do framework seria o apoio a padrões de projeto e meta-padrões em seu conjunto de elementos de construção. Em outras palavras, assim como os editores instanciados a partir do ABCDE possuem um conjunto de elementos de notação correspondente à notação que está sendo utilizada, estes editores também conteriam um catálogo de padrões de projeto e meta-padrões que poderiam ser utilizados pelo usuário.

  Alguns problemas a serem resolvidos para a correta implementação desta extensão são descritos em [FMvW97], mas além disso, sugerimos também que este editor deve apoiar também a criação de padrões de projeto a partir de buscas efetuadas na base de dados, conseqüentemente, o catálogo de padrões de projeto deve permitir que estes novos padrões possam ser incorporados a ele.

  Um editor de diagramas com suporte a padrões de projeto foi desenvolvido por Florijn et *al.*[FMvW97], no entanto, este editor não suporta meta-padrões. Um editor ao apoiar a utilização de meta-padrões pode ser considerado um avanço no estado da arte de pesquisa em frameworks uma vez que seria uma ferramenta de auxílio a construção dos mesmos. Além disso, uma vez que os meta-padrões e padrões de projeto foram utilizados no projeto do framework e estarão devidamente documentados, poder-se-ia gerar parte da documentação do framework de maneira automática.

Apêndice A

Formulário de Avaliação

Este apêndice descreve o formulário utilizado pelos alunos no processo de avaliação do framework ABCDE. Este formulário foi desenvolvido pela profa. Dra. Cecília Rubira para ser utilizado na disciplina MO620 - Engenharia de Software II do Instituto de Computação da UNICAMP, a partir de um modelo fornecido pela Fundação Centro Tecnológico para Informática (CTI).

---

1. Número do Avaliador:
2. Nome do Software Avaliado:
   (a) Software recebido na forma de:
   [ ] disquettes [ ] ftp [ ] página Web [ ] Outros Especifique:
   (b) Manuais Presentes
   [ ] Sim Quantos? [ ] Não
   (c) Ambiente Recomendado:

   Nas questões abaixo, indique o tempo gasto na avaliação. Dê uma nota na escala de 1 a 5, onde 1 é a nota mais baixa e 5 é a nota mais alta, no lugar reservado pelos colchetes "[ ]".
3. Instalação
   (a) Instruções [ ]
   (b) Execução [ ]
   (c) Tempo Gasto:

4. Documentação:

   (a) Organização [ ]

   (b) Apresentação Visual [ ]

   (c) Utilidade [ ]

   (d) Redação [ ]

   (e) Facilidade de Encontrar Informações [ ]

   (f) Tempo Gasto:

5. Funcionalidade (Verificação Funcional)

   (a) Adequação (software faz o que se propõe?) [ ]

   (b) Correção (resultados obtidos estão de acordo com o especificado?) [ ]

   (c) Robustez (software opera de maneira razoável em situações não especificadas?) [ ]

   (d) Tempo Gasto:

6. Usabilidade (Interface com o Usuário)

   (a) Adequação da Interface (a apresentação da interface e de suas funções é adequada aos objetivos a que se propõe?) [ ]

   (b) Operacionalidade da Interface (funções providas pela interface estão implementadas a contento sob o ponto de vista funcional?) [ ]

   (c) Disponibilidade de Recursos para Aprendizado e Execução (help on-line, conjunto de exemplos, mensagens informativas em tempo de execução, etc.) [ ]

   (d) Apresentação Visual da Interface (Inteligibilidade) (A apresentação visual é agradável e facilita a leitura e identificação das informações?) [ ]

   (e) Tempo Gasto:

7. Operacionalidade do Sistema (Nas funções verificadas, informa ao usuário o andamento das tarefas ou possíveis falhas ocorridas?) [ ] Tempo Gasto:

8. Originalidade e Criatividade

   (a) Relevância Científica, Tecnológica ou Educacional (avalie a contribuição do software) [ ]

(b) Criatividade [ ]

(c) Tempo Gasto:

9. Observações Gerais e Conclusões:

10. Familiaridade com ferramentas CASE [ ].

11. Experiência no desenvolvimento de software [ ].

Bibliografia

[ABL89] A. F. Ackerman, L. S. Buchwald, and F. H. Lewski. Software Inspections: An Effective Verification Process. *IEEE Software*, 6(2), May 1989.

[BJ94] Kent Beck and Ralph Johnson. Patterns generate architectures. In Mario Toroko and Remo Pareschi, editors, *European Conference on Object Oriented Programming - ECOOP'94*, number 821 in Lecture Notes on Computer Science, pages 139–149, Bologna, Italy, July 1994. Springer Verlag.

[BMR+96] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, and M. Stal. *Patterns Oriented Software Architecture: A System of Patterns.* John Wiley & Sons Ltd, Chichester, UK, second edition, 1996.

[BNPM93] R.M. Baecker, D. Nastos, I.R. Posner, and K.L. Mawby. The user-centred iterative design of collaborative writing software. In S. Ashlund, K. Mullet, A. Henderson, E. Hollnagel, and T. White, editors, *Proc. ACM INTERCHI'93 Conf. on Human Factors in Computing Systems*, pages 399–405, Amsterdam, The Netherlands, April 1993. New York: ACM SIGCHI.

[Boo94] Grady Booch. *Object-Oriented Design with Applications.* Benjamin Cummings, 2 edition, 1994.

[BRJ97] Grady Booch, James Rumbaugh, and Ivar Jacobson. Unified Modeling Language - versão 1.0. Technical report, Rational Software Corporation, jan 1997.

[BRSS94] Richard Bentley, Tom Rodden, Pete Sawyer, and Ian Sommerville. Architectural Support for Cooperative Multiuser Interfaces. *IEEE Computer*, 27(5):37–46, May 1994.

[BSM90] L. Brothers, V. Sembugamoorthy, and M. Muller. ICICLE: Groupware for Code Inspection. In *Proceedings of Conference on Computer Supported Collaborative Work*, pages 169–181, Los Angeles, CA, oct 1990.

[CAB+94] Derek Coleman, Patrick Arnold, Stephanie Bodoff, Chris Dollin, Helena Gilcheist, Fiona Hayes, and Paul Jeremes. *Object-Oriented Development: The Fusion Method.* Prentice-Hall, 1994.

[Cam96] Alda Campos. Design Patterns: o simples está na moda. *ComputerWorld*, 3(153):17, 1996.

[Cam97] Marcelo Ricardo Campo. *Compreensão Visual de Frameworks através da Introspecção de Exemplos.* PhD thesis, Instituto de Informática, Universidade Federal do Rio Grande do Sul, mar 1997.

[CI92] Roy H. Campbell and Nayeem Islam. A technique for documenting the frameworks of an object-oriented system. In *Proceedings of Second International Workshop on Object-Orientation in operating Systems*, pages 288–300, September 1992.

[CIM92] Roy H. Campbell, Nayeem Islam, and Peter Madany. Choices, frameworks and refinement. *Computing Systems*, 5(3):217–257, Summer 1992.

[Coa92] Peter Coad. Object-oriented patterns. *Communications of the ACM*, 35(9):152–159, September 1992.

[Cop91] James O. Coplien. *Advanced C++ Programming Styles and Idioms.* Addison Wesley Publishing Company, September 1991.

[CP95] Marcelo Campo and Roberto T. Price. O uso de técnicas visuais e navegacionais para compreensão de frameworks orientados a objetos. In *IX Simpósio Brasileiro de Engenharia de Software*, pages 175–190. Sociedade Brasileira de Computação, Outubro 1995.

[DB92] Paul Dourish and Victoria Bellotti. Awareness and coordination in shared workspaces. In J. Turner and R.E. Kraut, editors, *Proceedings of the International Confeference on Computer-Supported Cooperative Work*, pages 107–114, Toronto, Canada, November 1992. New York: SIGCHI & SIGOIS ACM.

[Die96] Elton Dietrich. Projeto de um Sistema de Suporte à Autoria Cooperativa de editor de Hiperdocumentos. Master's thesis, Instituto de Informática, Universidade Federal do Rio Grande do Sul, 1996.

[dSBCC95] Marcos Roberto da Silva Borges, Maria Claudia Reis Calvalcanti, and Maria Luiza Machado Campos. Suporte por Computador ao Trabalho Cooperativo.

In *XIV Jornada de Atualização em Informática*, 29 de Julho a 4 de Agosto, Canela - RS, 1995.

[DX97a] Márcio S. Dias and Geraldo Xexéo. Editor Cooperativo para Diagramação de Software OO. In *Anais do XI Simpósio Brasileiro de Engenharia de Software - SBES'97*, pages 499–502, Fortaleza, CE, 1997.

[DX97b] Márcio S. Dias and Geraldo Xexéo. Supporting Software Design with a Multi-Platform Collaborative Editor. In *World Multiconference on Systems, Cybernetics and Informatics - SCI'97*, Caracas - Venezuela, jul 1997.

[EG92] Thomas Eggenschwiler and Erich Gamma. Et++swapsmanager: Using object technology in the financial engineering domain. In *Conference on Object-Oriented Programming, Systems and Languages - OOPSLA'92*, pages 166–177, Vancouver, Canada, 1992. ACM Sigplan Notices.

[EGR91] C. A. Ellis, S. J. Gibbs, and G. L. Rein. Groupware: Some Issues and Experiences. *Communications of the ACM*, 34(1):9–28, January 1991.

[FcdM94] Hugo Fuks and Leonardo Mendonça de Moura. A Document Based Approach for Cooperation. *Journal of the Brazilian Computer Society*, 1(1):36–45, 1994.

[FGNR92] Gerard Fischer, A. Girgensohn, K. Nakakoji, and D. Redmiles. Supporting Software Designers with Integrated, Domain-Oriented Design Environments. *IEEE Transactions on Software Engineering*, 18(6):511–522, 1992. Special Issue on "Knowledge Representation and Reasoning in Software Engineering", A. Borgida and M. Jarke (Editors).

[Fir93] Donald G. Firesmith. Frameworks: the golden path to object Nirvana. *Journal of Object-Oriented Programming*, 6(6):6–8, October 1993.

[FKL88] R.S. Fish, R.E. Kraut, and M.D.P. Leland. Quilt: A collaborative tool for cooperative writing. In Robert B. Allen, editor, *Proceedings of ACM SIGOIS/IEEE TC-OA Conference on Office Information Systems*, pages 30–37, Palo Alto, CA, March 1988. ACM SIGOIS Bulletin, 9:2&3.

[Fla97] David Flanagam. *Java in a Nutshell.* O'Reilly & Associates, 1997.

[FMvW97] Gert Florijn, Marco Meijers, and Pieter van Winsen. Tool support for object-oriented patterns. In Mehnet Aksit and Satushi Matsuoka, editors, *11 th European Conference on Object-Oriented Programming - ECOOP'97*, number 1241 in Lecture Notes on Computer Science, pages 472–495, Finland, June 1997. Springer-Verlag.

[FP95] Edgardo Fabres and Jose A. Pino. SHADOW: Applications and Advances. In *Proceedings of CYTED-RITOS International Workshop on Groupware*, pages 139–148, Lisbon - Portugal, September 1995.

[FS97a] Mohamed E. Fayad and Douglas C. Schmidt. Object-oriented applciation frameworks. *Communications of the ACM*, 40(10):32–38, october 1997.

[FS97b] Martin Fowler and Kendal Scott. *UML Distilled: Applying the Standard Object Modeling Language*. Addison-Wesley, 1997.

[GHJV93] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. Design Patterns: Abstraction and Reuse of Object-Oriented Design. In Oscar Nierstrazs, editor, *European Conference on Object Oriented Programming*, pages 406–431, Kaiserslautern, Germany, 1993. Springer-Verlag.

[GHJV94] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison Wesley Publishing Company, April 1994.

[Gru94] Jonatahn Grudin. Groupware and Social Dynamics: Eight Challenges for Developers. *Communications of the ACM*, 37(1):92–105, January 1994.

[Har87] D. Harel. Statecharts: A Visual Formalism for Complex Systems. *Science of Computer Programming*, 8(3):231–274, jun 1987.

[HHG90] Richard Helm, Ian M. Holland, and Dipayan Gangopadhyay. Contracts: Specifying behavioral compositions in object oriented systems. In Norman Meyrowitz, editor, *European Conference on Object-Oriented Programming / Conference on Object-Oriented Programming, Systems and Languages*, volume 25, pages 169–180, Ottawa, Canada, October 1990. ACM SIGPLAN Notices.

[HJE95] Hermann Huni, Ralph E. Johnson, and Robert Engel. A framework for network protocol software. In *Conference on Object-Oriented Programming, Systems and Languages - OOPSLA '95*, volume 30, pages 358–369, Austin, Texas, October 1995. ACM Sigplan Notices, ACM SIGPLAN Notices.

[Hol92] Ian M. Holland. Specifying reusable components using contracts. In O. Lehrmann Hadson, editor, *European Conference on Object-Oriented Programming*, pages 287–308. Springer-Verlag, 1992.

[HW92] J.M. Haake and B. Wilson. Supporting collaborative writing of hyperdocuments in sepia. In J. Turner and R.E. Kraut, editors, *Proc. 4th Int. Conf.*

on Computer-Supported Cooperative Work, pages 138–146, Toronto, Canada, November 1992. New York: SIGCHI/SIGOIS ACM.

[JF88] Ralph E. Johnson and Brian Foote. Designing reusable classes. *Journal of Object-Oriented Programming*, 1(2):22–35, June/July 1988.

[Joh92] Ralph E. Johnson. Documenting frameworks using patterns. In Andreas Paepcke, editor, *Proceedings of Conference on Object-Oriented Programming, Systems and Languages - OOPSLA'92*, volume 27, pages 63–76, Vancouver, Canada, 1992. ACM SIGPLAN Notices.

[Joh93] Ralph E. Johnson. How to design frameworks. In Andreas Paepcke, editor, *Tutorial Notes of Conference on Object-Oriented Programming, Systems and Languages - OOPSLA'93*, Washington, DC, 1993.

[Kar96] Laurent Karsenty. An Empirical Evaluation of Design Rationale Documents. In *Proceedings of Conference on Human Interaction*, pages 150–156, Vancouver, BC, Canada, April 1996.

[LF96] G. Licea and J. Favela. ICARO: a web-based environment for collaborative software. In *Proceedings of II CRIWG - CYTED-RITOS International Workshop on Groupware*, September 1996.

[MDTR93] V. Mashayekhi, J. M. Drake, W. Tsai, and J. Riedl. Distributed, Collaborative Software Inspection. *IEEE Software*, 10(5), September 1993.

[MGL+87] T. W. Malone, L. R. Grant, K. Y. Lay, R. Rao, and D. Rosenblitt. Semistructured messages are surprinsingly useful for computer-supported coordination. *ACM Transaction on Office Information Systems*, 5:115–131, 1987.

[Mic97] Microsoft Corporation, Estados Unidos. *Microsoft Office 97*, 1997.

[MM93] Sten Minör and Boris Magnusson. A model for semi-(a)synchronous collaborative editing. In G. de Michelis, C. Simone, and K. Schmidt, editors, *Proc. of the 3rd European Conf. on Computer-Supported Cooperative Work ECSCW'93*, pages 219–231, Milan, Italy, September 1993. Dordrecht,Boston,London: Kluwer Academic Publishers.

[MXR97] C. Maidantchik, G. B. Xexeo, and A. R. C. Rocha. A WWW Software Development Environment to Support Cooperative and Spread Working Groups. In *Proceedings of the Computing in High Energy Physics' 97 Conference*, April 1997.

[NKCM90] Christine M. Neurwith, David S. Kaufer, Ravinder Chandhok, and James H. Morris. Issues in the Design of Computer Support for Co-authoring and Commenting. In G. de Michelis, C. Simone, and K. Schmidt, editors, *Proceedings of Computer Supported Cooperative Work*, pages 537–549. ACM Press, October 1990.

[Orf95] Edwards Hankly Orfaly. *The Essential Distributed Objects Survival Guide*, chapter 12, Object Frameworks: An Overview, pages 221–238. John Wiley & Sons, 1995.

[PB92] Ilona R. Posner and Ronald M. Baecker. How People Write Together. In *Proceedings of the Twenty-Fifth International Conference on the Systems Sciences*, pages 239–250, January 1992.

[Pre94a] Wolfgang Pree. *Design Patterns for Object-Oriented Software Development.* Addison Wesley Publishing Company, 1994.

[Pre94b] Wolfgang Pree. Meta Patterns- A Means for Capturing the Essentials of Reusable Object-Oriented Design. In Mario Toroko and Remo Pareschi, editors, *European Conference on Object Oriented Programming - ECOOP'94*, number 821 in Lecture Notes on Computer Science, pages 150–162, Bologna, Italy, July 1994. Springer Verlag.

[Pre95] Roger S. Pressman. *Engenharia de Software.* McGraw-Hill, 3 edition, 1995.

[QV95] Vincent Quint and Irene Vatton. Symposia - User Documentation. Available in http://symposia.inria.fr/symposia/userdoc/en/index.html, 1995.

[RBP+91] James Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, and W. Lorensen. *Object-Oriented Modeling and Design.* Prentice Hall International, 1991.

[RD92] B. Ramesh and V. Dhar. Supporting systems development by capturing deliberations using requirement engineering. *IEEE Transactions on Software Engineering*, 18(6):498–509, June 1992.

[RS92] Brent Reeves and Frank Shipman. Supporting Communication between Designers with Artifact-Centered Evolving Information Spaces. In J. Turner and R.E. Kraut, editors, *Proc. 4th Int. Conf. on Computer-Supported Cooperative Work*, pages 394–401, Toronto, Canada, November 1992. New York: SIGCHI/SIGOIS ACM.

[Rum95] James Rumbaugh. What is a method ? *Journal of Object-Oriented Programming*, pages 10–16,26, October 1995.

[SBF+86] M. Stefik, G. Bobrow, G. Foster, S. Lanning, and D. Tatar. WYSIWIS Revised: Early Experiences with Multiuser Iinterfaces. In *Proceedings of Conference on Computer-Supported Cooperative Work*, pages 276–290. ACM, New York: ACM, 1986.

[Sch95a] Hans Albrecht Schmid. Creating the architecture of a manufacturing framework by design patterns. In *Conference on Object-Oriented Programming, Systems and Languages - OOPSLA'95*, volume 30, pages 370–384, Ocotber 1995.

[Sch95b] Doug Schmidt. Using design patterns to develop reusable object-oriented communication software. *Communications of the ACM*, 38(10):65–74, October 1995.

[Sch96] Hans Albrecht Schmid. Design patterns for constructing the hot spots of a manufacturing framework. *Journal of Object-Oriented Programming*, 9(3), jan 1996.

[SHL+92] Norbert Streitz, Jörg Hannemann, Andreas Lemke, Wolfgang Schuler, Helge Schütt, and Manfred Thüring. SEPIA: A Cooperative Hypermedia Authoring Environment. In *Proceedings of European Conference on Hypertext*, pages 11–22, Milano, Italy, Nov. 30 - Dec. 4 1992. ACM.

[SP96] Ricardo Pereira Silva and Roberto Tom Price. Em direção a uma metodologia para o desenvolvimento de frameworks de aplicação orientados a objetos. In *Anais do X Simpósio Brasileiro de Engenharia de Software*, 15 a 18 de Outubro,São Carlos - SP, 1996.

[SRW98] Cleidson R. B. Souza, Cecília M. F. Rubira, and Jacques Wainer. Um Framework para Editores de Diagramas Cooperativos Baseados em Anotação. In Ernesto Pimentel and Carlos A. Heuser, editors, *Workshop Iberoamericano de Engenharia de Requisitos e Ambientes de Software*, pages 38–49, Torres, RS,Brasil, 1-3 Abril 1998.

[SS95] Doug Schmidt and Paul Stephenson. Experience using design patterns to evolve communication software across diverse os platforms. In Wlater Olthoff, editor, *European Conference on Object-Oriented Programming*, number 952 in Lecture Notes on Computer Science, pages 399–423, Aarhus, Denmark, August 1995. Springer-Verlag.

[Str91] B. Stroustrup. *The C++ Programming Language.* Addison Wesley, 2nd edition, 1991.

[SWR97] Cleidson R. B. Souza, Jacques Wainer, and Cecília M. F. Rubira. Um Modelo de Anotações para o Desenvolvimento Cooperativo de Software. In *III Workshop on Hypermedia and Multimedia Applications*, pages 143–154, São Carlos - SP, 23 - 25 Maio 1997.

[SWR98] Cleidson R. B. Souza, Jacques Wainer, and Cecília M. F. Rubira. Cooperative diagram editing on the web. In *Software Engineering over the Internet Workshop, International Conference on Software Engineering*, Kyoto, Japan, April 1998.

[Tal95] Taligent. Building Object-Oriented Frameworks. Disponivel em *http : //http.taligent.com*, 1995.

[Tut96] Renato Tutumi. Plataforma para ferramentas de edição de diagramas. Master's thesis, Departamento de Ciência da Computação, Universidade Estadual de Campinas, 1996.

[VL90] John M. Vlissides and Mark A. Linton. Unidraw: A Framework for Building Domain-Specific Graphical Editors. *ACM Transactions on Information Systems*, 8(3):237–268, July 1990.

[VS95] Iris Vessey and Ajay Paul Sravanapudi. CASE Tools as Collaborative Support Technologies. *Communications of the ACM*, 38(1):83–95, January 1995.

[WBJ90] Rebecca Wirfs-Brocks and Ralph E. Johnson. Surveying Current Research in Object-Oriented Design. *Communications of the ACM*, 33(9):04–124, 1990.

[WGM88] André Weinand, Erich Gamma, and Rudolf Marty. Et++ - An Object-Oriented Application Framework in C++. In *Conference on Object-Oriented Programming, Systems and Languages - OOPSLA'88*, pages 46–57. ACM Sigplan Notices, September 1988.

[WGM89] André Weinand, Erich Gamma, and Rudolf Marty. Design and Implementation of ET++, a seamless object-oriented application framework. *Structured Programming*, 10(2), 1989.

[You92] Edward Yourdon. *Análise Estruturada Moderna*. Editora Campus, Rio de Janeiro, 1992.