UNIVERSIDADE FEDERAL DE SANTA CATARINA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Dário Lissandro Beutler

UM FRAMEWORK PARA A CRIAÇÃO DE FERRAMENTAS DE AUTORIA PARA DOCUMENTOS MULTIMÍDIA

Dissertação submetida à Universidade Federal de Santa Catarina como parte dos requisitos para a obtenção do grau de Mestre em Ciência da Computação

Ricardo Pereira e Silva
Orientador

Florianópolis, outubro de 2003

UM FRAMEWORK PARA A CRIAÇÃO DE FERRAMENTAS DE AUTORIA PARA DOCUMENTOS MULTIMÍDIA

Dário Lissandro Beutler

Esta dissertação foi julgada adequada para a obtenção do título de Mestre em Ciência da Computação Área de Concentração Sistemas de Computação e aprovada em sua forma final pelo Programa de Pós-Graduação em Ciência da Computação.

_____
Prof. Fernando A. O. Gauthier, Dr
Coordenador

_____
Prof. Ricardo Pereira e Silva, Dr.
Orientador

Banca Examinadora

_____
Prof. Roberto Willrich, Dr.

_____
Prof. Raul Sidnei Wazlawick, Dr.

_____
Prof. Leandro José Komosinski, Dr.

AGRADECIMENTOS

Agradeço a Deus, por ter me concedido saúde, capacidade e discernimento para a conclusão deste importante empreendimento.

Ao meu orientador, professor Ricardo Pereira e Silva, por partilhar comigo seu precioso tempo e infindável competência. Fico extremamente grato e feliz por ter tido a oportunidade de trabalharmos juntos.

A minha namorada, Cleonice, pelo companheirismo, pelas opiniões e colaborações neste trabalho. O melhor desse trabalho foi fazê-lo estando junto com você.

Ao meu pai Martim e a minha mãe Iria por terem sempre me educado e incentivado para o estudo. Aos meus irmãos Diego e Carine por compreenderem a minha ausência durante o período desse trabalho.

Ao Instituto Cenecista de Ensino Superior de Santo Ângelo, especialmente ao diretor Dr. José Barcaro, que sempre depositou confiança em nosso trabalho, permitindo conciliar a atividade docente na instituição com o desenvolvimento do presente trabalho.

A Universidade Federal de Santa Catarina – em particular ao Departamento de Informática e Estatística por ter viabilizado esse mestrado.

SUMÁRIO

LISTA DE ABREVIATURAS

- GUI – Graphical User Interface
- HTML - Hypertext Markup Language
- ISO ODA -Office Document Architecture (Arquitetura oficial de documento)
- OMT – Object Modelling Technique (Técnica de Modelagem de Objetos)
- OOA – Object-Oriented Analysis (Análise orientada a objetos)
- OOAD - Object-Oriented Analysis and Design (Análise e Projeto Orientados a Objetos)
- UML – Unified Modelling Language (Linguagem de Modelagem Unificada)
- WWW – World Wide Web

LISTA DE FIGURAS

RESUMO

As ferramentas de autoria multimídia são sistemas que possibilitam a produção e apresentação de diferentes objetos de mídia inter-relacionados. Existe atualmente uma série dessas aplicações, porém, elas constituem uma área de desenvolvimento de software que precisa ser mais explorada. Desenvolver novas ferramentas de autoria, mais específicas, mais intuitivas e de menor custo representa um grande desafio. Desenvolver software com recursos multimídia constitui-se em uma atividade complexa e que demanda grandes custos em termos de tempo e esforço. A reutilização de uma arquitetura de software pré-elaborada vem contribuir para a criação de tais aplicações.

Este trabalho apresenta a análise, projeto e desenvolvimento do HyperToolBuilder, um framework orientado a objetos, voltado para a criação de ferramentas de autoria de documentos multimídia.

O framework HyperToolBuilder é voltado para a produção de ferramentas de autoria específicas para a criação de documentos multimídia, como por exemplo: apresentações, livros, enciclopédias, tutoriais, programas de exercício e prática e jogos educativos, todos com recursos multimídia. O objetivo é então fornecer suporte de reutilização em alta granularidade, tanto de código como de projeto de software para a produção dessas aplicações. Pretende-se auxiliar futuros desenvolvedores na criação destas ferramentas para que haja uma maior utilização das mesmas por profissionais das mais diversas áreas.

Palavras-chave*:* reuso de software, framework orientado a objetos, documentos multimídia, ferramentas de autoria

ABSTRACT

The multimedia authoring tools are systems that enable the production and presentation of different inter-related media objects. There are many products of multimedia authoring nowadays, however, they constitute an area of software development that needs to be explored over and over. Developing new more specific, intuitive and cheaper authoring tools represents a great challenge. Developing software with multimedia resources is a complex activity that demands great costs in terms of time and effort. The reuse of a previous elaborated software architecture comes to contribute to the creation of such applications.

This work presents the analysis, project and development of HyperToolBuilder, an object-oriented framework, destined for the creation of authoring tools for multimedia documents.

The framework HyperToolBuilder is destined for the production of authoring tools specific for the creation of multimedia documents, that includes presentations, books, encyclopedias, tutors, drill-and-practice programs and games, all of them with multimedia resources. The goal is then to supply reuse support in high granularity, so much of code as of software project for the production of these applications. It intends to aid future developers in the creation of these tools so that happens a larger use of the same ones by professionals of different areas.

Key-words: *software reuse, object-oriented framework, multimedia documents, authoring tools*

1. INTRODUÇÃO

O maior desafio enfrentado pelos desenvolvedores de software é aumentar a produtividade de suas equipes, mantendo a qualidade de seus produtos. As aplicações tornam-se cada vez mais complexas e precisam ser desenvolvidas com velocidades cada vez maiores. Neste paradoxo em que os desenvolvedores precisam ser cada vez mais produtivos, torna-se relevante a noção de reutilização de software. A idéia da reutilização é aproveitar o esforço já realizado não somente de código, mas principalmente de análise e projeto (FACH, 2001).

A crescente utilização da programação orientada a objetos contribui para aumentar a busca de teorias e artefatos de software reutilizáveis. Mas apenas fazer uso da programação orientada a objetos não é a garantia da obtenção eficiente da reusabilidade. Artefatos de software devem ser projetados para promover reuso. Se os mesmos forem projetados com essa finalidade eles provêem não apenas a diminuição do tempo e custo de desenvolvimento, mas principalmente decréscimo do custo de manutenção. Artefatos de software reutilizáveis simplificam a criação de novos sistemas e de novas versões de sistemas existentes (JOHNSON, 1988).

Existem diversas técnicas de projeto que dão ao software orientado a objetos um maior grau de reusabilidade. Uma das abordagens mais importantes para prover reuso são os frameworks, que consistem em projetos abstratos orientados a objetos (JOHNSON, 1988). A abordagem de frameworks objetiva proporcionar a reutilização de artefatos de alta granularidade, abrangendo tanto reuso de código como de projeto. O desenvolvimento de um framework consiste em conhecer várias aplicações de um domínio específico e criar uma estrutura de classes que as generalize. Essa estrutura deve ter partes completamente definidas, comuns a todas as aplicações, e partes flexíveis para proporcionar a possibilidade de implementação dos elementos específicos das aplicações do domínio tratado. O objetivo de um framework é proporcionar uma estrutura que torne o mais eficiente possível o desenvolvimento de aplicações em um domínio (SILVA, 2000).

Uma das áreas que traz grandes desafios ao desenvolvimento de software são as aplicações que manipulam recursos multimídia. Aplicações multimídia são aquelas

que produzem e manipulam ducumentos que utilizam e integram texto, imagem, áudio, vídeo, animação e gráficos em computadores. Para que se alcance um avanço no potencial das aplicações multimídia devem ser desenvolvidos sistemas que habilitem a produção e apresentação de objetos de mídia complexos e inter-relacionados. Esses sistemas são genericamente chamados de ferramentas de autoria multimídia (BULTERMAN, 1995).

Este trabalho tem como objetivo projetar e desenvolver o HyperToolBuilder, um framework orientado a objetos que fornece uma infra-estrutura para facilitar o desenvolvimento de ferramentas de autoria multimídia. Utilizando o framework HyperToolBuilder, o desenvolvedor terá à disposição um projeto abstrato e um protótipo implementado em linguagem Java, que servirá de subsídio para a produção de ferramentas de autoria, baseadas no paradigma de páginas ou cartões, para a criação de documentos multimídia. As futuras ferramentas de autoria deverão ser aplicações cada vez mais complexas e exigirão cada vez mais recursos do que as ferramentas tradicionais e o framework HyperToolBuilder proporcionará uma maior facilidade para o desenvolvimento dessas aplicações.

1.1 Motivação

A utilização de tecnologias multimídia constitui-se em uma das grandes possibilidades e necessidades da atualidade. Existe carência de soluções sob a forma de artefatos de software reutilizáveis para o modelo de domínio das ferramentas de autoria. A disponibilidade de um arcabouço de projeto e de código, sob a abordagem de frameworks para ferramentas de autoria constitui-se em um recurso importante para engenharia de software. Faz-se necessário também proporcionar reuso de software em um nível de granularidade elevado, o que é possível através da abordagem de frameworks.

1.2 Contexto da pesquisa

O tema do trabalho situa-se na área de Engenharia de Software, onde são estudadas e aplicadas técnicas de reutilização de software. Visa aplicar a abordagem de frameworks orientados a objetos na área de produção de ferramentas de autoria multimídia.

A partir do framework HyperToolBuilder poderá ser desenvolvida uma ferramenta de autoria A, por exemplo, para criar apresentações multimídia; uma ferramenta de autoria B para criação de livros eletrônicos; uma ferramenta de autoria C para criação de tutoriais multimídia e assim por diante.

O desenvolvimento de software multimídia é muito recente se comparado com outros tipos de sofware e por isso a preocupação em termos de reutilização deve acontecer o mais cedo possível. O que acontece atualmente é que geralmente apenas as grandes empresas (Microsoft, Macromedia, etc...) conseguem investir no desenvolvimento de software nessa área.

1.3 Objetivo do trabalho

O objetivo desse trabalho é projetar, desenvolver e aplicar o framework HyperToolBuilder, que dá suporte à produção de ferramentas de autoria para manipulação de documentos multimídia. O uso de um framework orientado a objetos para o desenvolvimento de ferramentas de autoria não apenas promoverá o reuso em alta escala, mas também reduzirá o nível de complexidade de desenvolvimento de ferramentas de autoria que incluem características complexas como: vídeo, música, gráfico e hipertexto. O framework proverá recursos de projeto e código de software para a futura criação de novas ferramentas de autoria, permitindo que desenvolvedores não necessitem produzir software do ponto zero (fazer com que os mesmos já possuam alguns recursos para o desenvolvimento de software).

Apesar de já existirem muitas ferramentas de autoria para documentos multimídia, existem atualmente poucos recursos disponíveis para a reutilização no desenvolvimento dessas aplicações, não somente de software, mas também de conhecimento, para elaboração de outros sistemas. Por isso o framework se mostrará bastante útil para reutilização de software no desenvolvimento dessas aplicações.

Uma solução reusável que dê subsídios para que profissionais de programação e pequenas empresas possam desenvolver softwares mais atraentes, personalizados, eficientes e adequados às diferentes situações de uso da multimídia. Uma das expectativas com o desenvolvimento do HyperToolBuilder é proporcionar artefatos de software (código) e conhecimento (projeto) para que “pequenos” desenvolvedores de sofware possam criar novas e diferentes ferramentas de autoria.

1.4 Abordagem utilizada

A forma de trabalho consistiu em inicialmente obter conhecimento e experiência em frameworks orientados a objetos e em ferramentas de autoria. Nesta pesquisa foi realizada uma análise extensiva da literatura pertinente disponível. Realizou-se uma coleta e leitura de artigos e livros sobre assuntos das seguintes áreas:

- Análise, projeto e programação orientados a objetos;
- Frameworks orientados a objetos;
- Padrões de projeto e metapadrões;
- Ferramentas de autoria;
- Linguagem de programação Java.

O passo seguinte foi obter conhecimento sobre os conceitos relacionados às ferramentas de autoria multimídia. Também foram analisadas várias aplicações do domínio de ferramentas de autoria, verificando os elementos que são comuns a várias destas aplicações e os que se diferenciam de uma aplicação à outra.

O terceiro passo foi construir iterativamente através dos elementos comuns e não comuns aos vários tipos de ferramentas de autoria um framework orientado a objetos, que pudesse futuramente ser reutilizado no desenvolvimento destas aplicações. Após, ocorreu o teste do framework, que consistiu em desenvolver duas aplicações – HyperBook e HyperTest - com características distintas (utilização do framework) para verificar a sua viabilidade. Finalmente foram realizadas as análises, interpretações e conclusões finais.

1.5 Escopo do trabalho

Como foi descrito na seção 1.3, dentre os objetivos desta pesquisa estão projetar e desenvolver um framework para ferramentas de autoria para documentos multimídia, a fim de fornecer aos desenvolvedores um artefato reusável para facilitar a criação dessas ferramentas.

O framework proporcionará um arcabouço em termos de projeto e código para ferramentas de autoria. Ele servirá como base para futuros trabalhos para criar estas ferramentas com mais recursos. Sabe-se que existe a necessidade de novas gerações

deste tipo de software. As futuras ferramentas de autoria deverão ser aplicações muito complexas, pois exigirão ainda mais recursos do que as ferramentas atuais. Este framework poderá servir futuramente como um módulo ao qual podem ser adicionados novos recursos para se chegar aos requisitos das novas gerações de ferramentas de autoria.

1.6 Estrutura da dissertação

O presente trabalho é constituído por seis capítulos, cujos conteúdos estão descritos a seguir.

O capítulo 2 apresenta conceitos e características no desenvolvimento de software baseado em frameworks. Abrange estudos a respeito de desenvolvimento de frameworks, bem como o desenvolvimento de aplicações a partir de frameworks. Este estudo é de extrema importância neste trabalho, pois se faz necessária a aquisição de conhecimentos e experiência, principalmente na construção e utilização de frameworks, já que a proposta do trabalho é neste sentido.

O capítulo 3 apresenta o estado da arte das ferramentas de autoria multimídia. Ele traz a definição dos termos multimídia. Apresenta também ferramentas de autoria multimídia como softwares necessários para a produção de documentos multimídia. O objetivo do capítulo é realizar um estudo das ferramentas de autoria, pois a proposta deste trabalho é um framework para este domínio de aplicações. O capítulo descreve também etapas e requisitos para a criação de documentos multimídia, bem como os paradigmas existentes para a autoria de documentos multimídia.

O capítulo 4 apresenta aspectos da análise e projeto do framework HyperToolBuilder, voltado para o desenvolvimento de ferramentas de autoria para documentos multimídia. Ele apresenta a modelagem estática, através dos diagramas de classes que representam os conceitos envolvidos no framework e seus relacionamentos para a representação das soluções de projeto. Traz também a modelagem dinâmica do framework, para ilustrar que todas as aplicações desenvolvidas sob o mesmo devem seguir ao protocolo de controle estabelecido pelas classes do mesmo.

O capítulo 5 apresenta a experiência da utilização do framework HyperToolBuilder através do desenvolvimento de aplicações (ferramentas de autoria) a partir do mesmo. São mostrados os aspectos funcionais da ferramenta de autoria para

livros eletrônicos chamada HyperBook e da ferramenta de autoria para construção de exercícios chamada HyperTest, ambas desenvolvidas baseadas no framework. Apresenta uma discussão de como as aplicações foram geradas a partir do HyperToolBuilder, mostrando a quantidade do reuso de classes ocorrido no desenvolvimento das duas aplicações a partir do mesmo. Traz também resultados obtidos a partir da utilização prática de uma das aplicações desenvolvidas sobre o framework, bem como aspectos relacionados ao desenvolvimento de outras ferramentas de autoria a partir do mesmo.

No capítulo 6 são apresentados os resultados obtidos do presente trabalho, são mostradas as limitações existentes e são propostos os futuros trabalhos e linhas de pesquisas utilizando os resultados do trabalho desenvolvido.

2. FRAMEWORKS ORIENTADOS A OBJETOS

Um framework orientado a objetos é uma estrutura de classes[1] inter-relacionadas, que corresponde a uma implementação incompleta para um conjunto de aplicações de um determinado domínio. A construção das aplicações se dará então através da adaptação dessa estrutura de classes (SILVA, 2000).

Este capítulo tem por objetivo conceituar e trazer características envolvidas na abordagem de desenvolvimento de software visando a reutilização, denominada framework orientado a objetos. Visa também tratar aspectos relacionados ao projeto e construção de frameworks, bem como sua utilização na criação de aplicações.

2.1 Conceito de framework

Segundo WIRFS-BROCK (1991), framework é um esqueleto de implementação de uma aplicação ou de um subsistema de aplicação, em um domínio de problema particular. É composto de classes abstratas e concretas e provê um modelo de interação ou colaboração entre as instâncias de classes definidas pelo framework. Um framework é utilizado através de configuração ou conexão de classes concretas e derivação de novas classes concretas a partir das classes abstratas do mesmo.

ORFALY (1995) define framework como 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. Já WIRFS-BROCKS (1990) define framework como um projeto orientado a objetos abstrato que pode ser adaptado segundo as necessidades da aplicação.

Apesar das inúmeras metodologias e paradigmas de desenvolvimento existentes atualmente, o desenvolvimento de software continua sendo um processo caro e demorado. Um grande problema enfrentado por equipes de desenvolvimento de software é de que forma aumentar a produtividade das equipes de desenvolvimento, mantendo-se a qualidade do produto. Esse problema torna-se ainda maior atualmente, pois ao mesmo tempo em que as aplicações tornam-se cada vez mais complexas[2], precisam ser desenvolvidas com velocidades cada vez maiores. Nesse paradoxo em que os engenheiros de software se encontram de

[1] Utilizando o paradigma de desenvolvimento orientado a objetos.
[2] Aplicações mais complexas envolve complexidade em todos os sentidos, como tamanho e número de informações a serem manipuladas, por exemplo.

desenvolver aplicações cada vez mais complexas em menos tempo (produtividade) surge o conceito de reutilização de software (FACH, 2001).

A idéia da reutilização é "não reinventar a roda", ou seja, aproveitar o esforço já realizado não somente de código, mas também em termos de planejamento e projeto. Com isso, não acontece só o ganho de tempo, mas principalmente a redução de erros por se estar utilizando artefatos já construídos e bem testados.

Existem várias abordagens para se alcançar a reutilização:

- reuso de funções (por exemplo, biblioteca de funções C) e classes de biblioteca (por exemplo, VisualWorks);
- através de componentes;
- utilizando padrões de projeto;
- frameworks orientados a objetos;
- através de geradores de aplicações e outros.

Fazendo uma analogia entre a técnica de frameworks e um projeto de casas pré-fabricadas, onde as casas de um conjunto, por exemplo, possuem a mesma planta baixa, são construídas com bases iguais, com paredes do mesmo tipo, telhado, portas e janelas básicas, e a partir delas pode-se mudar o telhado, as portas e as janelas, construir outros cômodos, aumentá-los ou mesmo remover algum deles. De forma semelhante um framework constitui-se de uma estrutura básica, na qual classes podem ser adicionadas, removidas ou estendidas para se adequar à aplicação em desenvolvimento.

A diferença entre a reutilização de classes de uma biblioteca e de um framework é que no primeiro caso geralmente se faz uma reutilização em baixa granularidade, pois se reutiliza apenas algumas classes isoladas que precisam ser interligadas para compor a aplicação. No segundo acontece a reutilização em alta granularidade, pois além de reutilizar um conjunto de classes se reutiliza também o seu inter-relacionamento planejado no desenvolvimento do framework (SILVA, 2000). As figuras 2-1, 2-2 e 2-3 mostram essa diferença (áreas sombreadas representam classes e associações que são reutilizadas).

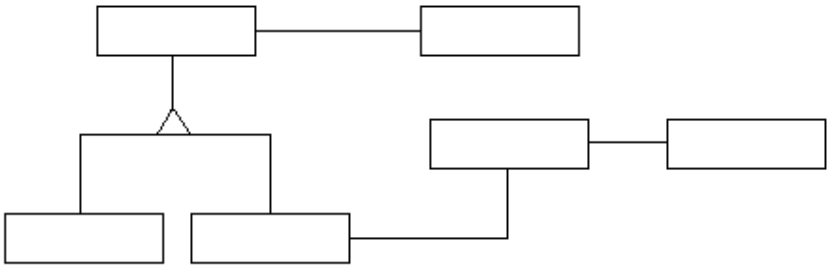


Figura 2-1 Aplicação desenvolvida totalmente

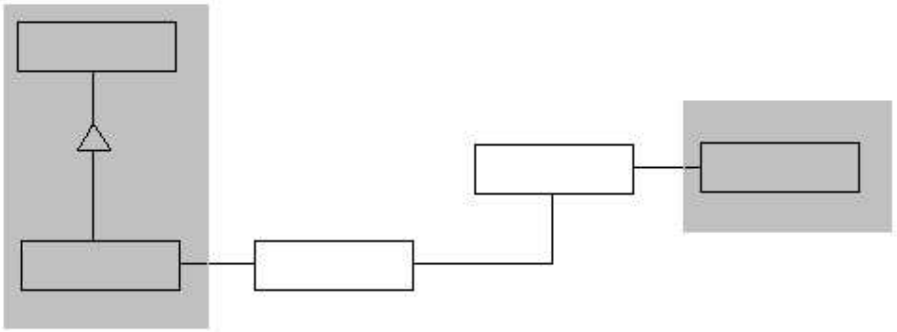


Figura 2-2 Aplicação desenvolvida reutilizando classes de biblioteca (SILVA, 2000)

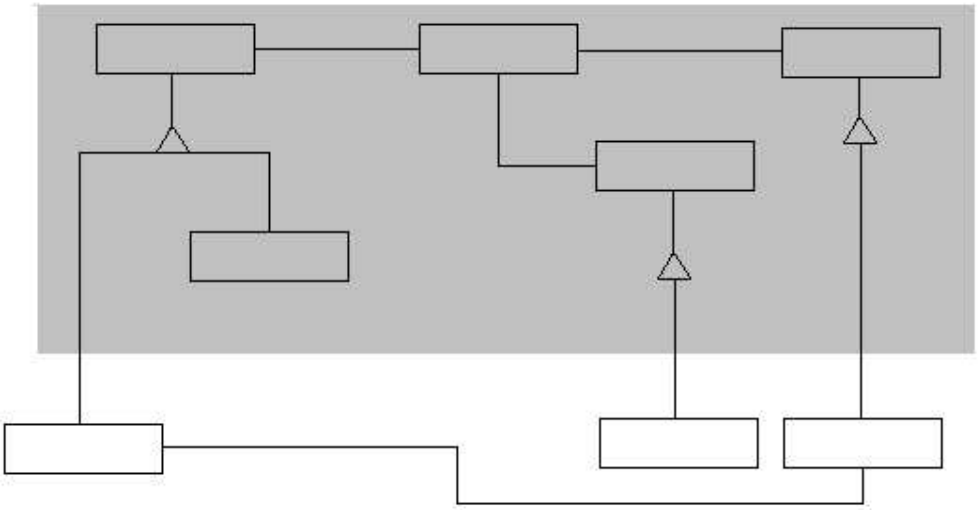


Figura 2-3 Aplicação desenvolvida reutilizando um framework (SILVA, 2000)

O foco da reutilização de software através de frameworks está justamente na reutilização não somente de código, mas também de projeto que, sem dúvida, é uma das tarefas mais importantes e complexas dentro do ciclo de desenvolvimento de software. A fase de projeto do framework é importante, pois inclui a definição do fluxo de controle das futuras

aplicações. A fase de projeto define, essencialmente, a sua divisão em partes, as interfaces entre elas e como estas partes cooperam para implementar uma determinada funcionalidade, que envolve um conhecimento profundo do domínio da aplicação. Subentende-se então que o projeto e o desenvolvimento de um framework, que necessite representar um equilíbrio adequado entre todos estes aspectos, não são tarefas triviais a serem realizadas por projetistas inexperientes. Neste sentido, ressalta-se a importância da reutilização de projetos.

As classes abstratas de um framework são os repositórios dos conceitos gerais do domínio de aplicação. No contexto de um framework, um método de uma classe abstrata pode ser deixado propositalmente incompleto para que a sua definição seja acabada na geração de uma aplicação. Apenas os atributos que são comuns a todas as aplicações de um domínio são incluídos em classes abstratas (SILVA, 2000).

Uma outra característica é que os frameworks podem incluir utilitários adicionais para auxiliar nas aplicações. Um utilitário pode ser, por exemplo, um gerador de código (ROGERS, 1997).

Quanto a granularidade, os frameworks podem ser encontrados com diferentes quantidades de classes, sendo assim mais ou menos complexos. O presente trabalho abordará frameworks independentemente da granularidade, porém, sempre supondo que a estrutura contenha mais de uma classe.

Um framework objetiva gerar diferentes aplicações para um domínio que pode ser bem específico ou mais abrangente. Quanto à abrangência existem duas categorias de frameworks, segundo ROGERS (1997): horizontal e vertical.

2.1.1 Frameworks horizontais

Os horizontais são mais gerais que os verticais e por isso podem ser usados por mais tipos de aplicações. *Toolkits* GUI são exemplos de frameworks horizontais. Eles podem ser usados para construir interfaces de usuário para uma grande faixa de aplicações. O framework horizontal por ser mais geral é mais utilizado, em maior quantidade, mas seu usuário deve se preocupar com a introdução de muitas características não desejadas.

2.1.2 Frameworks verticais

Os verticais são mais específicos para um domínio particular. Por exemplo, um framework para executar análise estatística de dados econômicos é específico para aplicações

financeiras. Um framework vertical é menos geral, mas usualmente é melhor utilizado e provê uma solução mais completa.

A distinção desses dois tipos de frameworks não é radical. De fato há uma grande área intermediária. Pode-se procurar misturar e combinar o que as classes têm de melhor nesses dois conjuntos, podendo-se chegar a uma boa solução para grande parte das aplicações.

2.2 Vantagens da utilização de frameworks

A utilização de frameworks oferece uma série de vantagens no desenvolvimento de software, como por exemplo:

- O desenvolvimento de novas aplicações torna-se mais rápido e mais barato, já que se utilizam componentes pré-fabricados e pré-testados. O desenvolvedor basicamente reescreverá alguns métodos específicos de determinadas classes. Já que a estrutura do programa e o fluxo de execução já estão definidos.

- Os frameworks permitem a reutilização de código e projeto. Ressalta-se aqui a importância do reuso de projeto que apresenta uma solução genérica para uma 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 à herança, quando um erro em um framework é corrigido, ou uma característica adicionada, os benefícios são imediatamente estendidos às novas classes (ORFALY, 1995).

- Possibilidade de desenvolvimento de aplicações mais complexas. A criação de frameworks baseados em outros permite o desenvolvimento de aplicações cada vez mais complexas. Como, por exemplo, o *ET++ SwapsManager* (EGGENSCHWILER, 1992) que é um framework para aplicações comerciais, desenvolvido a partir do framework *ET++* (WEINAND, 1988).

2.3 Desenvolvimento de aplicações a partir de frameworks

Os frameworks invertem a ótica do reuso de classes, da abordagem bottom-up para a abordagem top-down: o desenvolvimento inicia com o entendimento do sistema contido no projeto do framework, e segue no detalhamento das particularidades da aplicação específica, o que é definido pelo usuário do framework. Assim, o desenvolvimento de uma aplicação a

partir de um framework consiste em entendê-lo e a seguir adaptar sua estrutura de classes, fazendo com que esta inclua as particularidades da aplicação (TALIGENT, 1995).

A arquitetura de um framework deve ser representada por classes genéricas escritas usualmente em alguma linguagem de programação orientada a objetos. Cada classe essencialmente representa um dos componentes funcionais em que o domínio é dividido. As interfaces entre os componentes são definidas através das mensagens que são trocadas. As interfaces são implementadas por métodos que podem ser parte do framework (comportamento geral) ou podem ser parte de cada aplicação específica (comportamento específico) (CAMPO, 1997).

Diferentes aplicações podem ser instanciadas a partir de um framework. Esta instanciação pode ser feita de duas formas, segundo JOHNSON (1988):

2.3.1 – Frameworks caixa-branca (dirigidos à arquitetura)

Os frameworks caixa-branca (JOHNSON, 1992) ou também chamados de dirigidos à arquitetura são aqueles em que a aplicação é gerada a partir da criação de subclasses das classes do framework. Eles são mais difíceis de usar, pois exigem profundo conhecimento do framework e um certo esforço no desenvolvimento de código. Em resumo, o framework é reutilizado através da especialização de seus componentes. Essa forma de reutilização dos componentes do framework é chamada de reutilização baseada em herança.

Os frameworks acima referidos requerem uma documentação explícita 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 hiperdocumentos a partir do framework *ET++* (WEINAND, 1988), 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 (PREE, 1995).

2.3.2 – Frameworks caixa-preta (dirigidos a dados)

Os frameworks caixa-preta (JOHNSON, 1992) ou dirigidos a dados são aqueles em que as aplicações são geradas a partir de diferentes combinações de objetos, instâncias das classes do framework. Sua customização é feita através da disponibilização de um conjunto de componentes que fornecem o comportamento de aplicações específicas. Cada um desses 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 mesmos. Assim, o framework é reutilizado através da instanciação de seus componentes. São mais fáceis de usar, porém são menos flexíveis (FAYAD, 1997; JOHNSON, 1988; TALIGENT, 1994). Essa forma de reutilização dos componentes do framework é chamada de reutilização baseada em composição.

2.3.3 Frameworks caixa-cinza

Esses frameworks possuem uma base dirigida à arquitetura e uma camada dirigida a dados. Possibilita a criação de aplicações a partir da combinação de objetos e também permite a geração de subclasses (TALIGENT, 1994). Alguns dos componentes dos frameworks são reutilizados através de herança e outros através de composição. Johnson (JOHNSON, 1992) denomina esses frameworks de caixa-cinza.

2.3.4 Comparação e análise das abordagens de instanciação das aplicações

Como convenção se utilizará o termo caixa-branca para indicar que este framework baseia-se mais freqüentemente em herança do que em composição como mecanismo de reutilização. O inverso é válido para o termo framework caixa-preta.

Geralmente um framework recém-desenvolvido é do tipo caixa-branca, pois o domínio das aplicações ainda não é suficientemente compreendido para que seja possível parametrizar o seu comportamento. A partir de testes do framework e conseqüente reorganização de sua estrutura de classes, o domínio das aplicações passa a ser bem compreendido e os aspectos que devem ser flexíveis no framework são definidos. Nesse momento a tendência é que o framework se torne um caixa-preta.

A vantagem da reutilização baseada em herança é principalmente a facilidade de utilização do framework. As classes do framework implementam a maior parte das funcionalidades desejadas, o que permite propagar às subclasses as mudanças realizadas nas classes do framework. No entanto, isso torna-se uma desvantagem se considerarmos que as subclasses precisam ser modificadas caso as superclasses o sejam. Uma outra questão é que os métodos são determinados estaticamente, razão pela qual não se pode modificar dinamicamente os serviços executados.

A reutilização baseada em composição tem 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 podem manter referências para outros. Através destas referências, um objeto pode delegar a responsabilidade de executar um dado serviço a outro. Este, que realizará efetivamente o serviço solicitado, depende da configuração da instância criada.

A reutilização baseada em composição é mais desejável que a baseada em herança, pois permite maior flexibilidade ao framework. Entretanto, isso requer um conjunto de classes que implemente todas as funcionalidades do domínio. Como isso é muito difícil de ser obtido, freqüentemente surge a necessidade de se desenvolver novas classes que implementem serviços não disponíveis. Essas classes são implementadas reutilizando as funcionalidades de classes existentes através da reutilização baseada em herança. Assim sendo, composição e herança são técnicas complementares que, quando utilizadas adequadamente, contribuem para o desenvolvimento de sistemas flexíveis e reutilizáveis (CAMPO, 1997; TALIGENT, 1995).

No contexto de frameworks as aplicações são desenvolvidas especializando-se as classes do framework para fornecer a implementação de alguns métodos, embora a maior parte da funcionalidade da aplicação seja herdada. A diferença com relação a outros métodos de reutilização é que no caso dos frameworks , não são estes que são “chamados” pela aplicação, mas ao contrário eles é que “chamam” os métodos da aplicação que implementam suas especificidades. O controle da aplicação está nas classes do framework e não nas classes específicas da aplicação.

O projeto do framework é que vai definir a sua flexibilidade colocando as possibilidades e restrições a serem consideradas pelos usuários. Um framework deve apresentar uma documentação para que os usuários possam gerar aplicações.

O ideal é o que framework seja capaz de abranger todos os conceitos gerais de um domínio de aplicação, deixando apenas os aspectos particulares para serem definidos nas aplicações. O usuário não deve necessitar criar classes que não sejam subclasses de classes abstratas do framework. O objetivo disto é o framework conseguir generalizar o domínio modelado (SILVA, 2000).

2.4 Análise de domínio para desenvolvimento de frameworks

Construir um framework não é uma tarefa simples. Particionar um problema de forma correta e definir suas classes e interfaces, de modo que representem abstrações corretas do domínio, é uma atividade trabalhosa.

Idealmente o projeto de um framework é o resultado de um processo de análise de domínio (JOHNSON, 1993). Esta no contexto de frameworks se mostra de vital importância, pois consiste num estudo profundo para a generalização de um conjunto de aplicações com características comuns – este conjunto de aplicações com características comuns é chamado de domínio de aplicações. A análise de domínio se mostra importante no que consiste a capacidade futura de uma efetiva reutilização no desenvolvimento de aplicações deste domínio, ou seja, essa atividade representa substancialmente o maior esforço por parte do desenvolvedor, pois ela representará a capacidade real de reutilização em alta granularidade. Quanto mais bem conhecido for o domínio, mais êxito se obterá na produtividade de software neste domínio.

Segundo ARANGO (1991), análise de domínio é o processo de identificação e organização de conhecimento a respeito de uma classe de problemas – um domínio de aplicações – para suportar a descrição e solução desses problemas (ARANGO, 1991).

ARANGO (1991) situa a análise de domínio em um meta-nível em relação ao desenvolvimento de uma aplicação – ao invés de desenvolver uma aplicação, é desenvolvida uma meta-aplicação, que será reusada no desenvolvimento de aplicações específicas (ARANGO, 1991). Ele também propõe uma abordagem prática para a análise de domínio que consistiria nas etapas:

- Identificar as fontes de conhecimento;
- Adquirir o conhecimento existente;
- Representar o conhecimento, através de um modelo de domínio.

Segundo ROGERS (1997) existem alguns passos que podem ser seguidos para se alcançar uma especificação completa e concisa de um framework:

> *1) Categorizar o problema – existem duas razões para categorizar o problema – Primeiro: para se criar um repositório de frameworks eles devem ser categorizados pelo tipo de problema que representam. Segundo: quando se determina que um problema já está em uma categoria existente no repositório, o framework desta categoria pode ser reusado.*
>
> *2) Pesquisar problemas similares – pode-se pesquisar problemas similares através do conhecimento dos usuários que solicitam a aplicação ou através da própria experiência do desenvolvedor.*
>
> *3) Preparar uma descrição do domínio – Aqui se deve formalizar as anotações feitas nos passos anteriores com uma descrição do domínio. O problema será descrito numa visão semi-estruturada. Neste processo, pode-se identificar facetas do domínio que são comuns*

a outros domínios. Para preparar a descrição, deve-se primeiro preparar um glossário contendo definições das entidades domínio-específicas. Assim, não haverá confusão quanto ao seu significado. A terminologia domínio-específica é muitas vezes interpretada de forma diferente pelas pessoas envolvidas no projeto (usuários e desenvolvedores). Este glossário servirá para unificar as idéias. Em segundo lugar, deve-se criar uma lista de características que descrevem o domínio de problema. Esta lista será refinada por um procedimento de sete passos:

Passo 1) Criar uma lista de características descrevendo o domínio de problema. Cada característica deve ser brevemente escrita, para que seja expressa a sua essência. Não devem ser criadas mais que cinqüenta características. Se isto acontecer, deve-se verificar se o domínio de problema não está com os limites muito abrangentes e, assim, ajustar estes limites e refazer o passo.

Passo 2) Reagrupar as características para que as mais relacionadas entre si fiquem agrupadas. Após, reescrever as características para parecerem mais como requisitos. Isto trará mais consistência às mesmas. Uma dica é colocar o tempo verbal no futuro. Exemplo: em "os pedidos são distribuídos entre os vendedores" altera-se para "os pedidos deverão ser distribuídos entre os vendedores", ficando como um requisito.

Passo 3) Separar as características domínio-específicas das genéricas. Deve-se criar duas listas: características que são específicas do domínio e aquelas que não são.

Passo 4) Particionar graficamente as características domínio-especificas das genéricas. Deve-se começar com uma folha em branco e uma linha divisória vertical. Á esquerda desenha-se um círculo para cada característica específica. Colocando-se o número da característica dentro do círculo correspondente. Á direita, desenha-se um círculo para cada característica genérica, também colocando o número da mesma no centro.

Passo 5) Para cada característica da lista domínio especifica, tenta-se criar novas características reescrevendo-a com termos que não sejam domínio-específicos. Em outras palavras, tenta-se criar versões domínio-independentes. Então coloca-se as novas características genéricas na lista domínio-independente e adiciona-se círculos correspondentes à direita do diagrama. Cria-se uma seta do círculo representando a característica domínio-especifica original para as generalizadas que surgiram dela.

Passo 6) Identificar facetas por agrupamento gráfico das características genéricas. Agrupa-se as características genéricas circulando-as em grupos, isto é, criando partições. As regras para criação destas partições são as seguintes:

- *Características que são conceitos fechados não devem aparecer em nenhuma partição e sim sozinhas;*
- *Características mutuamente dependentes devem aparecer na mesma partição;*
- *Características com um tema em comum devem aparecer na mesma partição;*

Passo 7) Criar listas separadas de características para cada partição. Cada partição criada é uma faceta. Também remove-se os círculos individuais que estão dentro da partição. Qualquer faceta reconhecida contendo uma implementação no repositório deve ser rotulada com o nome existente, colocando o nome dentro do circulo, As novas facetas devem receber um nome apropriado ao tema. Exemplos

de facetas: Estoque, Contabilidade, Compras. Finalizando este passo, tem-se uma "proposta" para a descrição do domínio. "Proposta" porque muitas revisões deverão ser feitas e confirmadas com as pessoas experientes no domínio do problema.

8) Analisar facetas genéricas do domínio – consiste nos seguintes passos:

- *Nomear a faceta;*
- *Pesquisar a faceta sob a perspectiva de domínios diferentes – devem ser pesquisados vários domínios, para acumular evidências que a faceta se estende entre os domínios;*
- *Preparar uma descrição da faceta – cria-se um glossário contendo definições das entidades específicas da faceta. Depois cria-se uma lista numerada de características que descrevem a faceta, usando as instruções dos passos 1 e 2 vistos anteriormente. Não deverão existir mais de vinte e cinco características e cada uma deverá ter no máximo quatro frases.*

9) Repetidamente: revisar e refinar a análise – Estes procedimentos servem para reunir informações suficientes para iniciar o projeto de um framework. Quando o projeto for iniciado, pode-se esperar vários retornos à análise de domínio para refiná-lo até que os limites do domínio sejam bem estabelecidos.

Através dessas etapas pode-se concluir que a atividade da análise de domínio consiste numa das atividades mais importantes no desenvolvimento de um framework. O modelo de domínio em frameworks consiste em uma estrutura de classes. Esta estrutura contém a arquitetura e a definição da estrutura de controle das aplicações a serem geradas a partir dela, o que promove a reutilização do projeto. A implementação das classes do framework é reusada na geração de aplicações e, com isso, verifica-se também a reutilização de código. Assim, a reutilização promovida pela abordagem de frameworks abrange todo o ciclo de vida de uma aplicação, desde a etapa de análise até a implementação. Com isso, mostra-se clara a importância da técnica de frameworks em termos de promoção de reutilização (SILVA, 2000).

2.5 Desenvolvimento de frameworks

O desenvolvimento de um framework consiste em muitas iterações no seu projeto, o que não é algo que se invente de uma hora para outra (LEWIS, 1995). Desenvolver uma aplicação, construindo primeiramente um framework, levará mais tempo do que simplesmente resolver um problema específico. Isso ocorre porque é difícil separar os aspectos genéricos dos específicos (ROGERS, 1997). Entretanto, a incorporação de novas características e o desenvolvimento das aplicações seguintes será muito mais rápida.

A principal característica que se busca ao desenvolver um framework é alcançar a generalidade em relação a conceitos e funcionalidades do domínio e também produzir uma estrutura flexível, promovendo a alterabilidade e extensibilidade (SILVA, 2000).

A alterabilidade significa a capacidade do framework alterar suas funcionalidades. No desenvolvimento do framework deve se fazer uma diferenciação entre os aspectos que são comuns a um domínio e os conceitos específicos das aplicações. O framework deverá implementar então os conceitos comuns do domínio, prevendo a alterabilidade requerida na estrutura de classes.

Extensibilidade é a capacidade de agregação de novos recursos ao framework. No momento em que se desenvolvem novas aplicações, o framework, através de iterações, poderá agregar novos recursos, inclusive até podendo estender os limites do domínio tratado.

O desenvolvedor precisa ter clara a idéia das diferentes aplicações para justamente ter a capacidade de diferenciar os aspectos gerais do domínio (comuns a todas as aplicações do domínio) e os aspectos específicos (particulares de cada aplicação).

O ciclo de vida de um framework difere do ciclo de vida de uma aplicação convencional, pois nunca é um artefato de software isolado, mas a sua existência está sempre relacionada à existência de outros artefatos (aplicações) originadores do framework, originados a partir dele ou que exercem alguma influência na definição da estrutura de classes do framework. A Figura 2-4 a seguir apresenta aspectos relacionados ao ciclo de vida de frameworks: mostra inicialmente um framework gerado por um desenvolvedor ou uma equipe a partir de um conjunto de aplicações, após, a geração de aplicações sob este framework e juntamente neste processo as possíveis alterações do framework. Exibe finalmente a possibilidade de alteração do framework com a possível inclusão de características de novas aplicações.

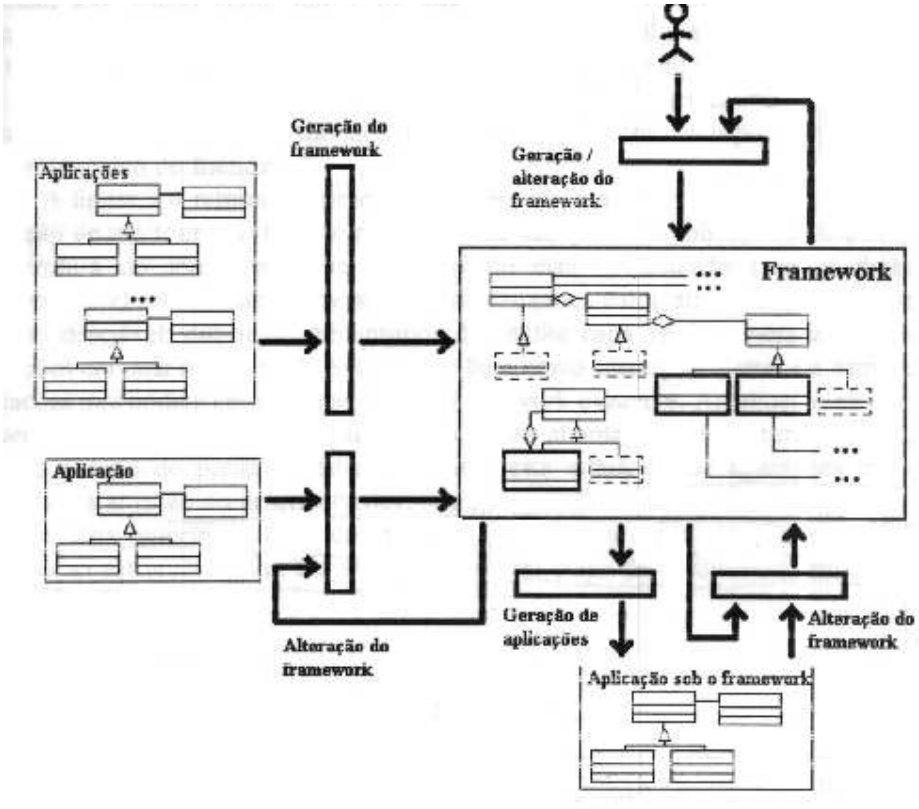


Figura 2-4 Ciclo de vida dos frameworks (SILVA, 2000)

O processo de análise de domínio consiste na construção de uma abstração de uma realidade tratada. Logo, é inevitável que o framework seja incapaz de conter todas as informações do domínio. Um framework consegue ser uma descrição aproximada do domínio, construída a partir das informações até então disponíveis (SILVA, 2000). Logo, verifica-se que o ciclo de vida do framework está diretamente ligado às possíveis aplicações do domínio e que, por isso, pode sofrer manutenções.

Existem algumas metodologias de desenvolvimento de frameworks, ou seja, propostas de procedimentos para obter informações de um domínio, a construção e o teste da estrutura do framework, as quais serão descritas brevemente:

2.5.1 Projeto dirigido por exemplo

Nesta metodologia (Example-Driven Design (JOHNSON, 1996)) os frameworks são projetados utilizando-se exemplos de aplicações prontas, que vão sendo generalizadas num processo de várias iterações. A evolução é sucessiva, com as tentativas de se adaptar o framework às necessidades de todas as aplicações-exemplo escolhidas.

Segundo JOHNSON (1996), através dos exemplos concretos, o desenvolvedor irá verificar aspectos semelhantes de diferentes aplicações, criando classes abstratas que agrupam as semelhanças, ficando para as classes concretas de nível inferior a especialização para satisfazer cada caso. A generalização (Figura 2-5) ocorre através da busca de elementos com nomes diferentes, mas são a mesma coisa; particionando os exemplos (parametrização) para obter elementos similares e agrupando esses elementos em classes.

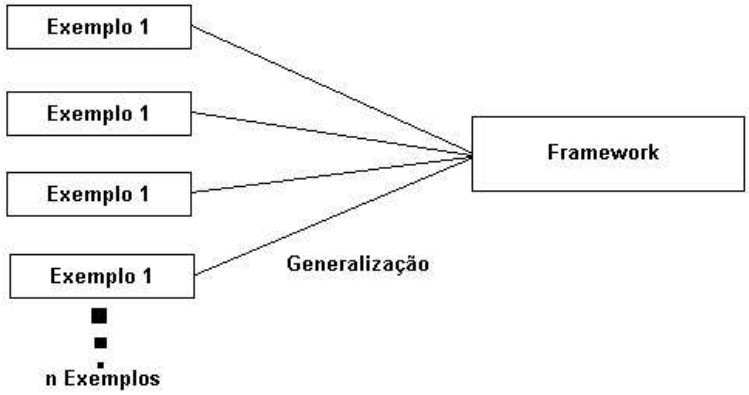


Figura 2-5 Projeto dirigido por exemplo

O processo de generalização visto na Figura 2-5 consiste na fatoração dos componentes dos exemplos, envolvendo três etapas(JOHNSON, 1996):

- *Identificação e criação de classes abstratas*: consiste em identificar o comportamento genérico dos exemplos envolvidos e criar classes abstratas que representem este comportamento geral, deixando para subclasses a definição do comportamento específico.

- *Criação de subclasses para eliminar condicionais*: O objetivo é mover para subclasses códigos que tratam diferentes casos específicos. Assim, as classes são fatoradas em novas subclasses (uma para cada caso específico), eliminando as sentenças condicionais correspondentes na classe origem.

- *Identificação de agregações e criação de componentes*: Em muitos casos o relacionamento de herança é usado quando na realidade se tem uma relação de agregação, ou composição, entre os objetos. Isso resulta nas hierarquias de classes não serem organizadas em função dos conceitos, ou da semântica que as classes representam, dificultando tanto a compreensão do projeto quanto a reutilização de código, o qual poderia ser encapsulado em outros componentes.

O processo apontado como ideal para o desenvolvimento de frameworks, segundo o projeto dirigido por exemplo, é composto pelas seguintes etapas:

1. Analisar o domínio;
2. Assimilar as abstrações já conhecidas;
3. Coletar um número mínimo de quatro aplicações como exemplo;
4. Avaliar a adequação de cada exemplo;
5. Criar a hierarquia de classes que possa ser especializada para abranger os exemplos;
6. Testar o framework usando-o para desenvolver os exemplos.

2.5.2 Projeto dirigido por *hot spot*

Os *hot spots* são as partes do framework mantidas flexíveis. A essência desta metodologia é justamente identificar as partes flexíveis (os *hot spots*) na estrutura de classes de um domínio e, com isso, desenvolver o framework. PREE (1995) propõe a seqüência abaixo (Figura 2-6) para desenvolvimento de um framework: em síntese esta abordagem consiste no desenvolvimento inicial de uma estrutura de classes e, após, a realização da identificação dos partes flexíveis de forma iterativa.

A seqüência de procedimentos da figura abaixo é proposta em (PREE, 1995):

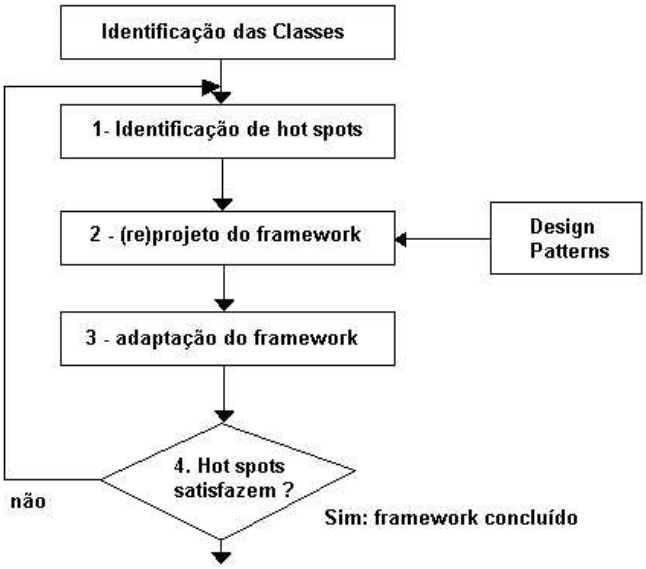


Figura 2-6 Etapas do projeto dirigido por hot spot

1 - Identificam-se aspectos que diferem de aplicação para aplicação e define-se o grau de flexibilidade para cada caso (Figura 2-6– etapa 1). Quando os aspectos diferentes são

os dados, classes abstratas agruparão os dados (atributos) comuns e classes concretas, específicas para as aplicações, agruparão os dados específicos. Já quando os aspectos diferentes forem as funções, devem existir nas classes métodos base (que implementam completamente a operação – representam o comportamento comum), métodos abstratos (em classes abstratas – quando a implementação deverá ser realizada por uma subclasse da classe onde o método está definido) que definem apenas o protocolo do método, métodos *template*, que chamam outros (*hook*) para completar seu processamento e métodos *hook*, que flexibilizam o comportamento dos *template*.

2 – Após, modifica-se a estrutura de classes para comportar a flexibilidade requerida (Figura 2-6 – etapa 2).

3 - Outra etapa é refinar a estrutura de classes a partir de novos conhecimentos do domínio. (Figura 2-6 – etapa 3)

4 - E finalmente se o framework for avaliado como satisfatório (os *hot spots* dão ao framework o grau de flexibilidade para gerar as aplicações requeridas?), se sim, a elaboração do framework está encerrada. (Figura 2-6– etapa 4)

2.5.3 Metodologia de desenvolvimento da empresa Taligent

A idéia de construção de frameworks da empresa já extinta Taligent (TALIGENT, 1994) é a de produzir, ao invés de um framework, um conjunto de frameworks estruturalmente menores e mais simples, que usados conjuntamente, darão origem às aplicações. A idéia dessa empresa é que pequenos frameworks são mais flexíveis e podem ser reutilizados mais freqüentemente. Assim, cada pequeno framework é voltado para aspectos específicos do domínio.

Essa metodologia propõe os seguintes passos para o desenvolvimento:

1. Identificar e caracterizar o domínio do problema – analisar o domínio e identificar os frameworks necessários, utilizando frameworks existentes e criando novos, bem como identificar os limites de responsabilidade de cada um dos frameworks;
2. Definir a arquitetura e o projeto – refinar a estrutura de classes do passo anterior, analisando como os usuários interagem com o framework – que

classes o usuário instancia? Que classes o usuário produz? Que métodos o usuário chama?

3. Implementar o framework – implementar as classes principais, e testar o framework, gerando aplicações;
4. Desdobrar o framework - criar documentações na forma de diagramas, receitas (como usar o framework para resolver determinada aplicação) e exemplos de aplicações simples, e realizar, se necessário, a manutenção do framework (corrigindo erros eventuais, adicionando novas características, mudando pouco as interfaces - recomendável adicionar novas classes ao invés de alterar hierarquia das classes existentes, e também adicionar novos métodos ao contrário de alterar métodos existentes).

Fazendo uma análise das três metodologias, SILVA (2000) ressalta que as seguintes características não são mutuamente exclusivas, ou seja, no desenvolvimento de um framework pode-se seguir determinadas características de uma metodologia sem ferir as características de outra:

- Todas comportam a busca de informações em aplicações já existentes, como no projeto dirigido por exemplo;
- Todos podem conter a ênfase na busca de partes flexíveis, como na proposta do projeto dirigido por *hot spots*;
- Todas podem ter a ênfase em minimizar a quantidade de código que o usuário deve desenvolver, como na proposta da empresa Taligent;
- Desenvolver frameworks extensos que cubram várias aplicações de um domínio ou pequenos frameworks para gerar as aplicações, como propõe a empresa Taligent. É uma decisão de projeto e pode ser adotada por qualquer metodologia.

SILVA (2000) também destaca as etapas que são comuns para as três metodologias e que devem ser observadas por todo projetista de um framework:

1. Aquisição de conhecimento do domínio.
2. Construção da estrutura de classes do framework (modelagem do framework).

3. Implementação do framework.
4. Avaliação do framework, com o desenvolvimento de aplicações.
5. Refinamento do framework a partir da aquisição de novos conhecimentos do domínio.
6. Uma outra característica comum entre as metodologias de desenvolvimento é a recomendação do uso de padrões de projeto (apresentados na próxima seção).

2.5.4 Padrões de projeto

Os padrões de projeto (*design patterns*) surgiram da observação de que diferentes partes dos frameworks possuíam estrutura de classes semelhantes. Um padrão de projeto corresponde a uma microarquitetura, em que são definidas as classes envolvidas, suas responsabilidades e a forma como cooperam. O uso de padrão de projeto consiste em incluir a microarquitetura escolhida no sistema em desenvolvimento de modo a incorporar ao sistema a funcionalidade desejada (SILVA, 2000). Em geral, padrões de projeto auxiliam a reduzir a complexidade em muitas situações da vida real (PREE, 1995). Existem padrões de comportamento para, por exemplo, dirigir um carro, realizar uma refeição, descrever boas combinações de roupa, etc. Também os frameworks bem desenvolvidos, usualmente, reusam diversos padrões de projeto.

Os padrões de projeto, diferentemente dos frameworks que são implementados em uma linguagem de programação, são experiências de projetos que descrevem como realizar certas tarefas de desenvolvimento de software. Essas experiências de projeto (hierarquia de classes, protocolos e distribuição de responsabilidades às classes) foram observadas durante o projeto de aplicações, sendo amadurecidas com o tempo (JOHNSON, 1996). A definição de GAMMA (1994) é que os padrões de projeto são organizações de hierarquia de classes, protocolos e distribuição de responsabilidades entre classes, que caracterizam construções elementares de projeto orientado a objetos. O conceito dado por CAMPO (1997) é que o padrão é uma estrutura que aparece repetidamente nos projetos orientados a objetos, o qual é utilizado para resolver um problema determinado de forma flexível e adaptável dinamicamente.

Projetistas experientes em orientação a objetos catalogaram padrões de projeto. Isto representa um recurso valioso, pois conhecer estes exemplos é muito útil para o

desenvolvimento de frameworks. Bons projetistas conhecem vários padrões (JOHNSON, 1996).

No desenvolvimento de software os programadores tendem criar programas imitando partes de programas escritos por outros, o que não significa copiar. Essa imitação envolve a percepção de um padrão dentro de um programa que pode ser adaptado.

O conceito de padrão pode ser visto como uma abstração da atividade de imitação. Por exemplo, livros de algoritmos também caem na categoria de padrões. Algoritmos de ordenação, por exemplo, são padrões (padrões de codificação) que descrevem como ordenar de forma eficiente, dependendo do contexto.

A utilização de padrões de projeto apresenta as seguintes vantagens (GAMMA, 1993; GAMMA, 1994; PREE, 1995; SCHMIDT, 1995; SCHMIDT & STEPHENSON, 1995):

- Os padrões formam um vocabulário padrão, possibilitando aos projetistas se comunicarem e documentarem projetos;
- Reduzem a complexidade do sistema, pois identificam abstrações que estão a um nível de abstração acima das classes e instâncias;
- Permitem larga reutilização de arquitetura de software, mesmo que a reutilização de algoritmos, implementações, interfaces ou projetos detalhados não seja permitida;
- São uma base de experiência reutilizável para a construção de software. 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 biblioteca de classes e frameworks. Uma vez que um usuário tenha aprendido uma biblioteca, ele poderá reutilizar esta experiência para aprender novas bibliotecas;
- Atuam como "blocos de construção" que podem ser utilizados par a criar aplicações mais complexas.

As seguintes notações ou combinações são aplicadas para abordagens de padrões de projeto:

- notação textual informal – uma descrição. Por exemplo, um resumo de situações onde o *pattern* pode ser aplicado;

- notação textual formal, isto é, utilizando um formalismo ou uma linguagem de programação;
- notação gráfica – utiliza diagramas classe/objeto e complementa as outras notações;

GAMMA (1993) apresenta um esquema de classificação que agrupa os padrões em categorias que facilitam a compreensão e utilização. Este sistema divide o conjunto de padrões em função de dois critérios ortogonais, descritos abaixo (CAMPO, 1997):

- Escopo – é o domínio sobre o qual o padrão pode ser aplicado. Os padrões cujo escopo é classe tratam sobre relacionamentos entre classes e subclasses. O escopo objeto trata relacionamentos entre objetos individuais, enquanto o escopo composto abrange as estruturas de objetos recursivos.
- Caracterização – identifica a função do padrão. Os criacionais envolvem o processo de criação de objetos, os estruturais tratam sobre a forma em que classes e objetos são compostos para formar estruturas de nível superior. Os comportamentais caracterizam a forma em que classes e objetos interagem e distribuem responsabilidades.

BUSCHMANN (1996) classifica os padrões em três categorias, segundo os seus níveis de abstração:

- Padrões de arquitetura: expressam um esquema da organização global da estrutura do sistema. Os mesmos provêem um conjunto de subsistemas pré-definidos, especificando os seus 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 granularidade 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 (COAD, 1992).

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

GAMMA (1992) foi o pioneiro a catalogar padrões de projeto em sua tese de Ph.D. Ele foi base para um catálogo mais avançado publicado em 1993 e 1994 (GAMMA, 1993; GAMMA, 1994) com mais de 24 padrões.

Segundo GAMMA (1994) um padrão deve apresentar quatro elementos essenciais:

1) O nome do padrão: usado para descrever um problema, sua solução e suas conseqüências. Assim, o vocabulário dos projetistas é ampliado, facilitando o desenvolvimento de software.

2) O problema: descreve quando aplicar o padrão. Explica o problema em questão e o contexto. Geralmente, 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.

4) As conseqüências: são os resultados e decisões que a utilização de um padrão acarreta. A descrição dessas conseqüências é vital para a avaliação dos custos e benefícios de um padrão e, com isso, auxiliar na decisão sobre sua utilização. As conseqüências de um padrão incluem seu impacto sobre a flexibilidade, portabilidade e expansibilidade do sistema.

Todas as metodologias de desenvolvimento de frameworks recomendam o uso de padrões no desenvolvimento de frameworks, pois os mesmos têm a capacidade de identificar, classificar e descrever soluções existentes, possibilitando o reuso destas soluções.

Através do uso de soluções semiprontas, o desenvolvedor geralmente ganha agilidade no seu desenvolvimento, apesar do tempo necessário para o mesmo conhecer esses padrões na medida em que se tornam disponíveis.

2.5.5 Metapadrões

Existe também o conceito de metapadrão (*metapattern*) (PREE, 1995) que é uma abordagem complementar à abordagem dos padrões de projeto. O referido autor observou e classificou alguns arranjos de funcionalidade sobre métodos, que se repetem em diferentes padrões de projeto – o que denominou metapadrões. Estes trabalham em um nível de abstração superior aos padrões de projeto. Eles definem como manter a flexibilidade das classes, encapsulando a funcionalidade sujeita à alteração em métodos *hook*[3], chamados por métodos *template*[4].

Um framework bem projetado deve conter um conjunto flexível de métodos *template* e *hook*. Métodos *template* implementam os pontos fixos – *frozen spots* (PREE, 1995) do framework, ou seja, as partes comuns das aplicações. Já os métodos *hook* implementam os pontos adaptáveis (PREE, 1995) – *hot spots* - que são as partes que podem ser estendidas para cada aplicação específica.

Um exemplo dos métodos *template* e *hook* em um framework para editores de diagramas constitui-se no seguinte: um método *template* cria a *pallete*[5] com os elementos necessários para a construção de determinados diagramas que este editor apóia. Este método não contém a definição dos elementos da *pallete*, ele apenas é responsável por chamar outros métodos que contêm esta definição. Estes são os métodos *hook*, que contêm a definição dos elementos da notação do editor. Assim, para que um usuário do framework modifique a notação apropriada, bastaria que ele modificasse estes métodos *hook*. O próprio framework através de seus métodos *template* se encarregaria de chamar estes métodos quando fosse necessário. Aí está a grande dificuldade do projeto de frameworks que é justamente a identificação dos seus pontos adaptáveis. O objetivo de um bom framework é que os usuários apenas sobrecarreguem os métodos *hook*, pois são eles que implementam os pontos adaptáveis.

---

[3] O método *hook* sempre será invocado por um método *template*, seu objetivo é proporcionar a flexibilização a um método *template*.

[4] O método *template* ao ser executado invoca pelo menos um outro método – um método *hook*. O método *template* comporta a parte imutável de um procedimento.

[5] Pallete – é uma barra de ferramentas. Possuindo vários botões para possibilitar a seleção das ferramentas disponíveis.

Segundo PREE (1995), os metapadrões complementam a abordagem de Gamma, pois os mesmos permitem a visualização dos pontos adaptáveis de um framework. Isso facilita o entendimento e conseqüentemente a utilização do framework. Cada metapadrão ocorre provavelmente várias vezes em um framework. Os metapadrões podem ser vistos como um meio de capturar e classificar o projeto de um framework.

2.5.6 Comparação entre padrões e metapadrões

Tanto padrões de projeto como metapadrões podem ser utilizados no projeto de um framework. As duas abordagens possuem o objetivo comum de tornar o framework flexível. Mas comparativamente poderíamos ressaltar alguns aspectos:

- Metapadrões podem contribuir para documentar o projeto de qualquer framework, independente de seu domínio (PREE, 1995).
- 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, como os padrões de projeto são mais concretos, são também mais fáceis de manusear e entender.
- Padrões de projeto são um guia mais concreto para a compreensão dos pontos adaptáveis de um framework (SCHMIDT, 1995).
- 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 (SCHMID, 1996).
- O catálogo de padrões de projeto, como um complemento para as metodologias de análise e projeto orientado a objetos, é insuficiente para apoiar o desenvolvimento de frameworks (PREE, 1995).
- O catálogo de padrões de projeto tem se tornado cada vez mais completo (SCHMID, 1996).

Em outras palavras, cada uma das abordagens apresenta vantagens e desvantagens, sendo mais adequado, no caso de frameworks, utilizar uma combinação das duas (SCHMID, 1996; SCHMIDT, 1995).

2.5.7 Etapas para construção da estrutura de classes de um framework

SILVA (2000) apresenta as seguintes etapas para a construção da estrutura de classes de um framework, observando que estas etapas devem ser seguidas iterativamente:

1. Etapa de generalização – identificação de estruturas idênticas entre as aplicações. O objetivo é chegar numa estrutura de classes que generalize o domínio tratado. Isto se dá através da confrontação das estruturas de classes das diferentes aplicações. Buscam-se também elementos prontos que podem ser reutilizados, frameworks e outros componentes.

2. Etapa de flexibilização – busca-se o que deve ser mantido flexível na estrutura de classes, de modo que a estrutura possa ser especializada, gerando diferentes aplicações. Localização de *hot spots* (partes flexíveis) na estrutura de classes e de soluções de projeto para modelá-los. Os *hot spots* são identificados quando se determinam situações de processamento comuns às aplicações(*frozen spots*), com variações de uma aplicação específica para outra.

3. Etapa de aplicação de metapadrões – quando de identifica um *hot spot* deve-se verificar o requisito de flexibilização por ele imposto com os casos tratados por padrões existentes. Se um padrão é adequado para a solução de um problema, deve ser incorporado ao framework. No caso dos metapadrões, o seu uso consiste em transformar um procedimento geral em método *template*, cujo comportamento é flexibilizado através de um método *hook*.

4. Etapa de aplicação de padrões de projeto – incorporar padrões de projeto consiste em incluir as classes de um padrão selecionado na estrutura de classes em desenvolvimento, ou seja, fazer com que classes já existentes assumam as responsabilidades correspondentes às classes do padrão de projeto.

5. Etapa de aplicação de princípios práticos de orientação a objetos – é necessário seguir princípios de projeto orientado a objetos, como o uso de herança para reutilização de interfaces, ao invés de herança para reutilização de código; reutilização de código através da composição de objetos;

preocupação em promover polimorfismo, na definição de classes e métodos, para possibilitar acoplamento dinâmico, etc.

2.6 Uso de frameworks

A tarefa de utilizar frameworks consiste em produzir aplicações a partir desse framework. De acordo com JOHNSON (1996), o usuário de um framework é o programador de aplicações. Quando este usuário utiliza um framework, seu trabalho é prover apenas as partes específicas do seu problema. Toda a infra-estrutura da aplicação deve estar no framework e esta infra-estrutura é que consome o maior esforço de desenvolvimento (ROGERS, 1997).

O objetivo dos frameworks é o aumento da produtividade e qualidade no desenvolvimento de software, em função da reutilização promovida. Porém, quando se desenvolve software utilizando um framework, existe a necessidade de se aprender a desenvolver aplicações sob este framework. E a utilização de framework obviamente só terá sentido se o esforço em aprender a usar o framework for menor que o necessário para desenvolver toda uma aplicação sem uso desta metodologia (SILVA, 2000).

Pode-se ressaltar com isso, então, que quem está projetando e desenvolvendo um framework deve estar preocupado em fornecer subsídios para minimizar o esforço para compreender o framework a ser utilizado. Algumas informações que devem ficar claras aos usuários de um framework são: que tipo de aplicações podem ser desenvolvidas a partir de um framework, informações de como desenvolver aplicações elementares sob o framework e informações dos detalhes do projeto do framework.

Algumas questões-chave para usar frameworks do tipo caixa branca ou caixa cinza (SILVA, 2000):

- Quais classes? Que classes devem ser desenvolvidas pelo usuário e que classes concretas do framework podem ser reutilizadas ?
- Quais métodos? Ao gerar uma classe concreta para uma aplicação, que métodos devem ser definidos e que métodos herdados podem ser reutilizados?

- O que os métodos fazem? Quais as responsabilidades dos métodos e em que situações de processamento eles atuam e como eles implementam a cooperação entre diferentes objetos?

2.6.1 Documentação de frameworks

Um framework deve apresentar uma documentação em que os usuários possam usar para gerarem aplicações. Pois se no desenvolvimento convencional a documentação é importante, ela torna-se mais importante ainda no contexto de frameworks. Afinal, se os usuários não entenderem o framework, eles não o utilizarão (TALIGENT, 1995). Outro detalhe é que geralmente a complexidade de se documentar um framework é maior do que a de documentar uma aplicação convencional, devido à natureza abstrata dos frameworks.

Quem desenvolve um framework deve fornecer informações aos usuários principalmente de quais recursos são disponibilizados, para que eles evitem esforços desnecessários e também para que encontrem respostas para suas dúvidas. Pois, muitas vezes, se o usuário não conhecer bem o framework poderá dispender vários esforços desenvolvendo o que poderia ser reutilizado.

Existem basicamente dois tipos de descrição de frameworks (SILVA, 2000):

- Descrição do projeto – é baseada em metodologias OOAD (metodologias de análise e projeto orientados a objetos, como a de Coad e Yourdon (COAD, 1993), OMT (RUMBAUGH, 1994), OOSE (JACOBSON, 1992), Fusion (COLEMAN, 1994), de Martin e Odell (MARTIN & ODELL, 1995) e UML – que é apenas de análise (OMG, 2001). Também pode possuir descrição textual, referências ao código fonte e descrição de aspectos isolados do projeto do framework;

- Descrição de como usar – são os passos a serem seguidos para se desenvolver uma aplicação.

Vários autores têm proposto diferentes técnicas especialmente projetadas para auxiliar a documentação de frameworks, como, por exemplo, *cookbooks* (CAMPO, 1997), contratos (HOLLAND, 1992), padrões de projeto (GAMMA, 1994) e metapadrões (PREE, 1995).

Cookbooks são um conjunto de receitas textuais que estabelecem de forma mais breve possível como usar um framework para produzir aplicações.

A técnica de contratos baseia-se em uma linguagem semiformal que descreve os mecanismos de colaboração entre as classes do framework. CAMPBELL & ISLAM (1992) utilizam esta técnica para descrever parte do framework de um sistema operacional.

Os padrões de projeto (*design patterns*), vistos na seção 2.5.4, são utilizados como mecanismo de documentação de duas maneiras distintas: (i)como mecanismo para descrever a arquitetura do framework, o que permite que o usuário entenda o projeto e conseqüentemente entenda o funcionamento e as restrições; e (ii) como mecanismo para descrever a utilização do framework.

A abordagem de metapadrões, vistos na seção 2.5.5, é utilizada como um mecanismo de auxílio no processo de construção de framework, pois adiciona flexibilidade ao projeto do mesmo. Esta abordagem também pode ser usada como um mecanismo de documentação, pois descreve como utilizar o framework.

Geralmente grande parte das formas de documentação de frameworks (*cookbooks* ou padrões de documentação) não é ao todo suficiente para desenvolver certas aplicações; neste caso o que geralmente se tem disponível é o código fonte do framework, ou exemplos de aplicações desenvolvidas sob o framework. Mas compreender partes de um framework a partir do código fonte é uma atividade de baixo nível e obviamente uma tarefa mais árdua para se aprender a usar um framework. Porém, a disponibilidade de código não deixa de ser um recurso adicional para sanar determinadas dúvidas do desenvolvedor.

2.7 Considerações Finais

A utilização do desenvolvimento de software orientado a objetos possibilita a produção de artefatos de software reutilizáveis através da herança e do polimorfismo, porém, apenas a utilização dessa tecnologia não é suficiente para garantir que se desenvolvam artefatos reutilizáveis.

Para realmente se obter reuso no desenvolvimento de software é necessário que os desenvolvedores estejam disciplinados para isso. Segundo JOHNSON (1988), o desenvolvimento de artefatos reutilizáveis não acontece por acaso, deve existir a intenção da produção desses. Eles devem conhecer e fazer uso das abordagens para se alcançar a reutilização. As principais abordagens para obter artefatos reutilizáveis são: o reuso de classes, de geradores de aplicação, de componentes, de padrões de projeto e de frameworks. Essas abordagens não são mutuamente exclusivas, elas podem ser utilizadas em conjunto.

Frameworks orientados a objetos constituem-se em um modo de reutilizar projetos de alta granularidade. Um framework é um projeto genérico de um domínio que pode ser adaptado a aplicações específicas. Pode-se fazer uma analogia entre o desenvolvimento de software através de frameworks e o preenchimento de um formulário, por exemplo. Desenvolver uma aplicação sem o uso de framework seria comparado a receber um papel totalmente em branco e redigir todas as informações necessárias. Já no desenvolvimento de uma aplicação com o uso de framework seria comparado a receber um formulário pré-elaborado, com lacunas a serem preenchidas. No primeiro caso toda a informação precisa ser pensada e criada do ponto zero e no segundo já existe uma estrutura pré-montada, facilitando muito o preenchimento.

Em função dos frameworks serem uma das mais importantes formas de reuso, pois promovem a reutilização de projeto (JOHNSON, 1988), optou-se por utilizar essa abordagem para pesquisar a viabilidade do desenvolvimento de suporte reutilizável, para facilitar o desenvolvimento de aplicações em um domíno específico. Esse domínio constitui-se em aplicações de ferramentas de autoria para a criação de documentos multimídia, que serão discutidas no próximo capítulo.

3. FERRAMENTAS DE AUTORIA MULTIMÍDIA

A integração e uso de várias mídias em microcomputadores está sendo cada vez mais usual como instrumento para a troca de informações. Porém, para tornar esta forma de comunicação mais fácil, atrativa e eficiente é necessário que sistemas capazes de habilitar a produção e apresentação de objetos de mídia complexos inter-relacionados sejam desenvolvidos. Estes sistemas são geralmente chamados de ferramentas de autoria multimídia (BULTERMAN, 1995).

Neste capítulo são discutidos aspectos das ferramentas de autoria multimídia, é examinado o estado atual da arte e, também, é analisado um conjunto de desafios de pesquisa que precisam ser realizados para que todo o potencial de tecnologia de produção multimídia possa ser utilizado, para compartilhar informação efetivamente.

3.1. Multimídia

A codificação e distribuição de informação foram problemas que mantiveram os seres humanos ocupados desde seus primórdios. Talvez o marco mais significante nessa atividade foi o desenvolvimento da escrita. A tecnologia disponível para produzir documentos escritos permaneceu praticamente constante há cem anos atrás, até quando recursos significantes nos meios para criar e distribuir documentos começaram a acontecer. Alguns exemplos são a imprensa, o serviço postal, o desenvolvimento de jornais, o copiador de xerox e, mais recentemente, a combinação de sistemas de composição de páginas baseados em computador com as redes de computadores e impressão a laser. Cada um desses desenvolvimentos surgiu de uma necessidade específica de um usuário para fazer o processamento de informação baseado em texto mais efetivo, fidedigno, reproduzível e amplamente disponível.

Durante os últimos cinco anos, um aumento enorme na velocidade de processamento dos computadores e uma redução dramática no custo de dispositivos periféricos inteligentes criou o potencial para aumentar, igualar e substituir o texto como a forma padrão de troca de informações. Esses desenvolvimentos consistem no termo genérico, multimídia.

O termo **multimídia** define a utilização de diversas mídias para a transmissão de informações, ou seja, utilizar recursos simultâneos de áudio, vídeo, imagens, texto, animações e etc... para a comunicação (FLUCKIGER, 1995). No cenário mundial, multimídia agrupa

cinco das indústrias que mais crescem: informática, telecomunicações, publicidade, consumidores de dispositivos de áudio e vídeo, indústria de televisão e cinema. Por esse motivo a pesquisa de ferramentas e componentes que possam melhorar a qualidade do universo multimídia é sempre bem aceita e incentivada.

Ao contrário do desenvolvimento voltado ao usuário para facilidades de processamento de textos, muito do crescimento da multimídia foi o resultado do desenvolvimento relativamente independente da tecnologia de produção de periféricos. Isso conduziu a um distanciamento entre a possibilidade do computador processar e apresentar informações e a habilidade do usuário para criar e administrar documentos que exploram o poder da tecnologia disponível.

O problema de criar e administrar informação multimídia não é trivial. Considere a introdução de áudio ao microcomputador: apesar de toda experiência na produção e processamento da informação falada, o "e-mail auditivo", por exemplo, não se desenvolveu como uma alternativa natural ao correio eletrônico baseado em texto. A desvantagem principal não é a falta de familiaridade com a mídia, mas a falta de ferramentas integradas de edição e que permitem a um grande número de usuários ter acesso à tecnologia e criar mensagens que podem ser compartilhadas facilmente com outros. Os problemas de criação e edição incluem a habilidade para visualizar, reorganizar e refinar documentos. Esses problemas são compostos quando devem ser manipuladas juntamente várias mídias e quando o uso dessas mídias requer treinamento especializado, antes que elas possam ser usadas para troca de informação em geral (BULTERMAN, 1995).

A motivação do uso destas aplicações é o aumento da transferência de informações pelo uso simultâneo de um ou mais sensos do usuário. Isso porque os humanos aprendem mais, e mais rapidamente, quando vários dos seus sentidos (visão, audição, etc...) são utilizados.

Outro objetivo de uma aplicação multimídia é emular a comunicação humana face-a-face. Isso tem levado à contínua investigação de sistemas de comunicação e computação que se aproximam da velocidade de transmissão, fidelidade e eficiência das comunicações entre as pessoas.

3.2 Ferramentas de autoria multimídia

Ferramentas de autoria multimídia ou ambientes de desenvolvimento multimídia facilitam e automatizam a criação de documentos multimídia. Há uma grande variedade de tais ambientes, também chamados de sistemas de autoria. Estes são projetados para fornecer ferramentas de criação e organização de uma variedade de elementos de mídia como textos, gráficos, imagens, animações, áudio e vídeo a fim de produzir documentos multimídia[6]. Os usuários desse tipo de software, os autores dos documentos, são profissionais das mais diversas áreas e estudantes que desenvolvem apresentações educacionais, de marketing, artes gráficas e etc... Os mesmos tomam decisões acerca do layout gráfico e estilo de interação que os usuários finais reais (estudantes ou indivíduos no público) vêem e ouvem (BUFORD, 1994).

3.3 Tipos de mídia

Há um conjunto fundamental de tipos de mídia que define o conteúdo que aparece em um documento multimídia. Esses tipos de mídia terão que ser manipulados e apresentados pela ferramenta de autoria. Idealmente eles deveriam ser tipos de dados fundamentais na linguagem de *scripting* da ferramenta (DENNIS, 1995).

FLUCKIGER (1995) classifica as mídias em dois tipos: Mídia Discreta e Mídia Contínua. A **Mídia Discreta** é a representação da informação de forma estática, ou seja, sem movimento. Podendo essa adquirir algum movimento ou animação por meio de propriedades aplicadas às mesmas, pela utilização das ferramentas de autoria.

Um dos tipos de mídia discreta é o texto que permanece como um dos elementos mais importantes nos documentos multimídia. Apesar da manipulação e apresentação de texto já estar em um nível bem avançado, faltam ferramentas integradas para estruturar, indexar e resumir textos. Enquanto as imagens e sons em uma enciclopédia eletrônica são atraentes, a incorporação de grandes textos que trazem muita informação se torna também muito importante (DENNIS, 1995).

A **Mídia Contínua** representa a informação dinâmica, cuja semântica evolui com o tempo e a principal característica é o movimento ou alterações de percepção da mesma. As

---

[6] Um documento multimídia pode ser definido como uma estrutura descrevendo a coordenação e estilo de apresentação de uma coleção de mídias estáticas e dinâmicas (WILLRICH, 1996).

principais representantes de mídia contínua são: o vídeo digitalizado, som digitalizado e as animações.

Autores de documentos multimídia já têm ferramentas de edição de conteúdo que eles entendem e fazem bom uso. Um sistema de autoria multimídia deve ser aberto à interface com tais ferramentas mais conhecidas. Por exemplo, editores de vídeo digital de alta qualidade estão disponíveis para profissionais de vídeos. O sistema de autoria deve procurar aceitar o maior número possível de formatos padrões de tipos de mídias de muitas indústrias (DENNIS, 1995).

3.4 Documentos multimídia

Um documento multimídia pode ser visto como uma especificação de atividade que pode ser usada para coordenar a execução (*runtime*) da apresentação de uma coleção de objetos de mídia, em que cada objeto pode conter um ou mais itens de mídia, de um ou mais tipos (BULTERMAN, 1995). WILLRICH (1996) define um documento multimídia como uma estrutura descrevendo a coordenação e estilo de apresentação de uma coleção de mídias estáticas e dinâmicas.

Um exemplo de documento multimídia real pode ser a apresentação dos pontos turísticos de uma cidade, constituída de textos, imagens e vídeos apresentados em uma ordem pré-definida e possivelmente com alguns mecanismos de interação. Quando o autor cria um documento multimídia, ele define a orquestração da apresentação dos componentes, a partir da definição dos instantes de início e fim das apresentações, das relações condicionais e temporais entre e no interior dos componentes. Esta descrição da ordem temporal relativa da apresentação dos componentes é chamada de cenário multimídia ou tela de apresentação.

Documentos multimídia podem ser interativos. A interação é o componente de autoria multimídia que difere um documento multimídia de um "mero" filme. Através da interação o usuário pode controlar o fluxo de eventos em vez de receber os dados passivamente – é a grande vantagem que a utilização dos computadores adiciona ao conteúdo. Autores precisam de ferramentas novas para protótipo, implementação, reuso e avaliação de mecanismos de interação. Para muitos sistemas isso implica em ferramentas para lidar com interfaces gráficas para o usuário.

3.5. Documentos hipertexto e hipermídia

Um documento hipertexto é uma estrutura de informação organizada de maneira não linear: os dados são armazenados em uma rede de nós conectados por ligações ou *links* (WILLRICH, 1996):

- Os **nós** contêm as unidades de informação compostas por texto e outras informações gráficas. Em geral, um nó representa um conceito ou uma idéia expressa de uma maneira textual ou gráfica.
- Os **links** definem as relações lógicas (ou semânticas) entre os *nó*s, isto é, eles definem relações entre conceitos e idéias. A noção de *âncor*a permite a especificação de uma parte da informação que será fonte ou destino de um *link.*

Como os textos e imagens são informações estáticas (as informações não evoluem no tempo), a noção de tempo não é utilizada quando da especificação de documentos hipertextos clássicos. No hipertexto, o tempo de apresentação é determinado pelo leitor do documento.

Um **documento hipermídia** é uma combinação de documentos hipertexto e multimídia (WILLRICH, 1996) (como ilustrado na Figura 3-1).

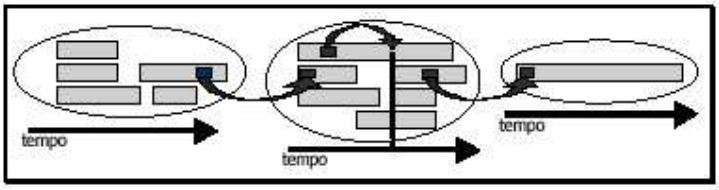


Figura 3-1 Documento hipermídia (WILLRICH, 1996)

Em muitos trabalhos, não há a distinção entre documentos multimídia e hipermídia. Nesse caso o termo documento multimídia engloba os documentos multimídia e os documentos hipermídia. A fim de simplificar o tratamento dos termos, documento multimídia será utilizado para especificar tanto os documentos multimídia quanto os hipermídia.

3.6 Criação de documentos multimídia

Embora diferentes documentos multimídia possam impor diferentes requisitos, há um conjunto de passos relativamente consistentes que o autor deve seguir quando ele desenvolve um documento. Esses passos, apresentados na Figura 3-2, guiam o autor da inspiração inicial ao documento acabado. Como apresentado nesta figura, o processo de autoria pode ser dividido em quatro estágios (WILLRICH, 1996):

- *Análise e projeto preliminare*s, em que os requisitos para o documento, seu conteúdo e suas interfaces são especificados;
- *Aquisição de material*, em que os materiais que formarão o documento são coletados, criados ou digitalizados;
- *Composição do documen*to, em que é realizada a composição lógica (através de links), temporal e espacial dos componentes do documento;
- *Avaliação e liberaçã*o, em que o documento é testado, refinado e, finalmente, distribuído para sua audiência.

Pode-se observar na Figura 3-2 que há realimentações importantes no processo de autoria (por exemplo na composição do documento pode-se verificar a necessidade da aquisição de novos materiais). Ainda nesta figura, estes quatro estágios não são sempre realizados em seqüência. Certos estágios podem ser realizados em paralelo (por exemplo, a aquisição de material pode ocorrer simultaneamente com a composição do documento). Além disso, a criação do documento pode ser feita via prototipagem, onde as etapas de aquisição de material, composição e avaliação podem ser repetidas até a conclusão do documento. Mesmo na prototipagem, a etapa de análise e projeto preliminar deve ser a mais completa possível a fim de aumentar as chances de se alcançar as metas definidas.

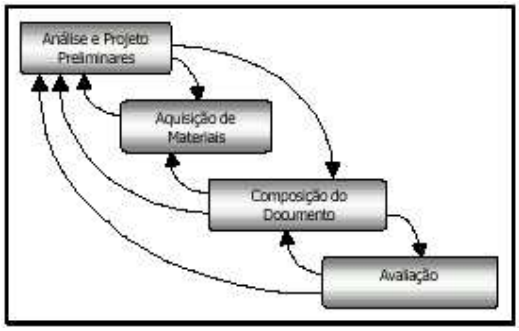


Figura 3-2. Passos na construção de documentos multimídia (FLUCKIGER, 1995)

Apesar de apresentar os tópicos básicos, este modelo não oferece recursos o suficiente para guiar todo o processo de autoria, pois identifica os passos, porém não oferece ferramentas ou modelos práticos para o desenvolvimento do objetivo multimídia.

3.7. Requisitos para um modelo multimídia

Baseado na ISO ODA *(Office Document Architecture*) (ISO 8613, 1998), várias abordagens dividem a descrição de documentos em três partes:

- **Estrutura lógica**, que descreve as diferentes partes lógicas de um documento (ou componentes) e suas relações lógicas;
- **Estrutura de apresentação**, que descreve como, onde e quando os diferentes componentes serão apresentados;
- **Estrutura de conteúdo** descreve as informações que constituem os componentes.

FLUCKIGER (1995) afirma que a partição da descrição do documento nestas três estruturas oferece várias vantagens. A separação entre a estrutura lógica e de apresentação, por exemplo, permite a reutilização da estrutura lógica quando da apresentação do documento em dispositivos diferentes, onde somente uma modificação simples ou adaptação da estrutura de apresentação deve ser realizada; e a separação entre a estrutura de apresentação e do conteúdo é necessária, pois os documentos podem utilizar os mesmos dados em diferentes contextos ou por diferentes documentos (SCHLOSS, 1994).

A separação entre a semântica pura (a estrutura lógica) e a definição dos instantes de aparição dos componentes (o quando, definido pela estrutura de apresentação) é contestada por vários especialistas. Isto porque um documento multimídia não é somente aquilo que ele contém, mas também quando seus conteúdos são apresentados. Assim, baseado na arquitetura proposta por KLAS (1990), WILLRICH & DIAZ & SAQUI-SANNES (1996) consideram que um modelo multimídia ideal deveria permitir uma descrição multinível incluindo as três seguintes estruturas:

> ***a)Estrutura de Conteúdo*** *– descreve as informações que constituem os componentes.*
>
> *Uma das primeiras etapas da realização de um documento multimídia é a geração ou captura das mídias (textos, imagens, áudios, vídeos, etc.) que vão ser utilizados para compor o documento. Por exemplo, o autor pode usar uma câmera de vídeo para registrar uma seqüência de vídeo, criar uma informação gráfica a partir de um editor. Esses são os chamados* ***dados primitivos****.*
>
> *A estrutura do conteúdo é responsável pela descrição dos dados primitivos. Por exemplo, ela pode ser constituída por um conjunto de dados primitivos e seus descritores. O par de dados primitivos e descritores destes dados é chamado de* ***objeto multimídia****.*
>
> *No nível da estrutura do conteúdo, um modelo multimídia deve permitir, entre outros, a especificação das informações de acesso e de manipulação dos*

dados primitivos e os valores originais das características espaciais, sonoras e temporais de apresentação.

***b)Estrutura de Apresentação** - descreve como e onde os diferentes componentes serão apresentados;*

A estrutura de apresentação de um documento descreve as características espaciais, sonoras e temporais de cada apresentação que compõem o documento (Existem outros tipos de tratamento dos componentes que suas apresentações, como a preparação dos dados e a execução de executáveis. Algumas vezes, o autor pode personalizar o tratamento de informações. P.e., um script pode conter parâmetros de entrada cujos valores podem ser definidos pelo autor. Não será considerado este tipo de caracterização de um documento multimídia). A este nível, o autor deve especificar as características de apresentação de cada componente e a composição espacial destas apresentações em um dado instante. A especificação das características de apresentação inclui a descrição da maneira como o componente será visto (descrição das características espaciais) e/ou ouvido (descrição das características sonoras) pelo leitor do documento. A composição espacial descreve também as relações espaciais entre as apresentações.

Especificação das Características de Apresentação

A estrutura conceitual define os componentes do documento e suas relações. A partir disto, o autor deve definir o conteúdo dos componentes (isto é, associar um objeto multimídia ao componente) e particularizar e/ou definir suas características de apresentação. P.e., o autor pode trocar as características originais de apresentação dos objetos multimídia (como o volume sonoro, velocidade de apresentação, etc.), ou ele pode definir novas características, como a posição espacial de apresentação de uma imagem. A fim de modelar uma apresentação, um modelo multimídia deve permitir a especificação das seguintes informações:

- *Características temporais de apresentação das informações dinâmicas, como a velocidade, posição de início e de fim de um vídeo e o número de repetições.*
- *Características espaciais de apresentação de informações visuais, como o tamanho, a posição e o estilo de apresentação.*
- *Características das apresentações sonoras, como o volume de apresentação.*
- *Dispositivos de saída, chamados aqui de canais, nos quais as informações serão apresentadas e vista pelo leitor (por exemplo, uma janela, um canal de áudio).*
- *Apresentações alternativas podem ser definidas a fim de repor uma apresentação principal se ela não puder ser apresentada em um certo sistema (permitindo a criação de documentos adaptáveis aos recursos disponíveis), se existirem problemas de acesso, ou restrições temporais não satisfeitas.*

***c) Estrutura Conceitual** - descreve as diferentes partes lógicas do documento ou componentes, suas relações lógicas, e em que instante os componentes serão apresentados;*

A estrutura conceitual especifica os componentes e os grupos de componentes de um documento, e a composição lógica e temporal destes componentes.

Os componentes e grupos de componentes

A estrutura conceitual contém as definições dos componentes semânticos de um documento, permitindo a divisão do documento em, por exemplo, capítulos e parágrafos, ou uma série de seqüências de vídeo e de títulos. Um exemplo desta estruturação é apresentado na Figura 3-3**.**

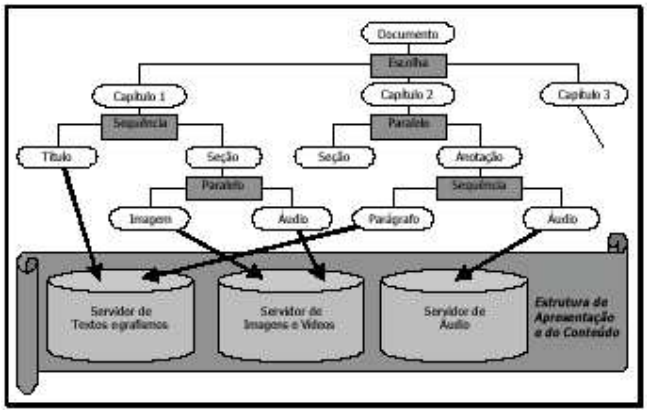


Figura 3-3 Componentes e grupos de componentes

A tarefa de especificação de Documentos Multimídia se torna delicada e complexa com o aumento de tamanho do documento. A estrutura conceitual é utilizada para a construção de apresentações complexas a partir de pequenos grupos. Assim, a descrição do documento pode ser dividida em seções, cada seção podendo ser criada de maneira independente das outras seções (HARDMAN, 1995): as apresentações que exprimem juntas, uma idéia ou um conceito podem ser agrupadas para posterior reutilização. Além disso, esses mecanismos de estruturação permitem a composição do documento a partir de técnicas top-down e ou bottom-up. Eles permitem também a definição de relações lógicas e temporais entre estes grupos e a definição de restrições temporais associadas a um grupo de apresentações. Concluindo, a estrutura conceitual introduz o benefício da modularidade, da encapsulação e mecanismos de abstração.

Independência dos componentes e seus conteúdos

A definição do componente do documento deve ser realizada independentemente das informações acerca da representação do seu conteúdo (KLAS, 1990), que são descritos pelas estruturas de apresentação e do conteúdo (como ilustrado na Figura 3-3*). Assim, os componentes da estrutura conceitual devem fazer unicamente menção às informações que estes componentes representam. A fim de simplificar a especificação do documento, esta menção deve ser independente do tipo de informação (mídia ou não-mídia) ou de sua localização geográfica (local ou remota).*

Quando do armazenamento e da transmissão do documento, certas partes da estrutura conceitual e/ou do conteúdo dos componentes podem ser incluídas explicitamente ou implicitamente na estrutura do documento:

- ***Inclusão explícita****: a estrutura do documento contém os dados primitivos. Assim, a estrutura do documento e os conteúdos incluídos explicitamente são armazenados no mesmo arquivo e transferidos conjuntamente.*

- ***Inclusão implícita**: a estrutura do documento faz referência aos dados primitivos ou outras partes da estrutura conceitual. Assim, no modo de transmissão em tempo real e de telecarga somente uma parte do documento é transferida. Quando da apresentação do documento o sistema cliente deve acessar de maneira remota as partes incluídas implicitamente no documento.*

A única diferença entre esta classificação e a primeira é que o "quando" da apresentação dos componentes é transferido da estrutura de apresentação para a estrutura lógica.

Esse tipo de mecanismo de interação é suportado por ferramentas especializadas. Um modelo multimídia ideal deveria permitir a especificação de todos esses tipos de interação. Além disso, a fim de simplificar a tarefa de estruturação conceitual de um documento multimídia, é necessário que o modelo possa fornecer uma abordagem uniforme de representação dos componentes, das relações lógicas e temporais e dos mecanismos de interação.

Assim, contrariamente à afirmação de FLUCKIGER (1995), quando da apresentação do documento em diferentes sistemas clientes, a estrutura lógica e o comportamento temporal (a estrutura conceitual) podem ser reutilizados. As apresentações (a estrutura de apresentação e do conteúdo) não suportadas pelo sistema cliente devem ser adaptadas e o "quando" da apresentação destes componentes deve ser respeitado. Por exemplo, se um componente constituído de uma apresentação de áudio e o sistema cliente não suporta a apresentação deste tipo de mídia, o conteúdo deste componente deve ser substituído por uma informação textual. Mas os comportamentos temporais dessas apresentações devem ser equivalentes.

Além da descrição dessas três estruturas, um modelo para essas aplicações deve permitir, também, a especificação dos possíveis métodos de interação com o usuário. Dessa forma, um modelo deve permitir a definição de âncoras, de links hipermídia e de outros métodos de interação de documentos multimídia.

3.8. Paradigmas para Autoria de Documentos Multimídia

Existem atualmente várias ferramentas de autoria. Uma lista completa com links para as diversas ferramentas de autoria comerciais e de domínio público pode ser encontrada em FAQ (1997). Além destas, existem outras ferramentas que são encontradas na literatura acadêmica que exploram abordagens mais inovadoras.

Vários paradigmas básicos de autoria são suportados pelos sistemas de autoria multimídia. Bulterman e Hardman (BULTERMAN, 1995) dividem os paradigmas de autoria existentes em quatro grandes grupos:

3.8.1. Autoria baseada em gráfico

Este paradigma usa um diagrama esquemático (gráfico) das interações de fluxo de controle entre os objetos multimídia. Os gráficos podem ser informais ou formais. São usados gráficos informais para dar uma ilustração global do ordenamento rudimentar (esboço) das interações dos objetos. Os gráficos formais também ilustram as interações de objetos, mas eles provêem um mecanismo adicional para provar ou determinar propriedades da apresentação baseadas na semântica do gráfico.

A autoria baseada em gráfico pode prover descrições de apresentações muito poderosas. Para determinados documentos, porém, gráficos informais e formais ficam mais complexos na medida que o tamanho da apresentação cresce. Para administrar esses complexos e sofisticados sistemas de interface com o usuário é necessário que se permita a ele ver mais de longe ou mais de perto (zoom-in e zoom-out) a apresentação. Isto limita a utilidade da abordagem, especialmente no caso de gráficos informais, pois, neste caso, não há nenhum valor somado ao próprio gráfico além de um mecanismo de ordenamento; uma vez que a apresentação se torna tão complexa que enche múltiplas páginas de tela e a visão integrada de uma apresentação é perdida. Para gráficos formais, esta complexidade somada é compensada pelos resultados gerados do formalismo gráfico, mas também neste caso a complexidade da descrição tipicamente limita o uso de tais modelos para necessidades de autoria de propósitos especiais, como a análise de sincronização (BULTERMAN, 1995).

Um exemplo deste paradigma de autoria informal é a **abordagem baseada em ícones** que é similar a programação (por exemplo, utilizando a linguagem C), mas com a ajuda de uma interface gráfica. Esta fornece ícones de alto nível, a fim de visualizar e manipular a estrutura do documento.

A abordagem de composição de documentos multimídia baseada em ícones é adotada, por exemplo, pelos softwares AimTech IconAuthor, Eyes M/M (EYES, 1992), Authorware Professional (AUTHORWARE, 1989), mTropolis e HSC Interactive.

Outro exemplo da utilização do paradigma baseado em gráfico é a **composição hierárquica de documentos multimídia**, em que os componentes de um documento são organizados na forma de uma árvore.

As relações temporais são definidas por operadores de sincronização. Os principais operadores de sincronização são os operadores série e paralelo. Estes definem que um conjunto de ações será executado em série ou paralelo. Essas ações podem ser atômicas ou compostas (BLAKOWSKI, 1996): uma ação atômica manipula a apresentação de uma informação, a interação com o utilizador ou um atraso; as ações compostas são combinações de operadores de sincronização e ações atômicas.

Um exemplo do paradigma de autoria baseado em gráficos formais é a **técnica de Redes de Petri** que é muito utilizada na engenharia de protocolos, automação industrial e muitas outras áreas. Ela permite a realização de uma especificação formal de um sistema e permite também a aplicação de técnicas de análise sobre o sistema a fim de verificar certas propriedades (MURATA, 1989).

As Redes de Petri também podem ser utilizadas para a modelagem de documentos multimídia, isso devido a sua representação gráfica, a facilidade de modelagem dos esquemas de sincronização e a possibilidade de analisar propriedades importantes do comportamento lógico e temporal do documento. Existem vários modelos multimídia baseados em redes de Petri. Uma lista não exaustiva inclui: OCPN (LITTLE, 1990), TSPN (DIAZ, 1993), RTSM (YANG, 1996), HTSPN (SÉNAC, 1996), MORENA (BOTAFOGO, 1995), PHPN (WANG, 1995).

Nesses modelos, geralmente um lugar da rede de Petri representa uma apresentação; as transições e arcos definem relações lógicas e temporais.

3.8.2 Autoria baseada em linha do tempo

Os sistemas de autoria baseados em linha do tempo provêem uma linha temporal para representar o fluxo de interações entre objetos multimídia. Ao contrário dos gráficos de fluxo de controle discutidos previamente, a maioria dos ambientes de autoria baseados em linha secular usa apenas descrições informais.

A natureza da linha secular é semelhante a uma pontuação musical: vários eventos são mostrados em paralelo a um eixo comum. Este paradigma é especialmente apropriado a apresentações multimídia, com o seu uso inerente de eventos paralelos, eventos baseados no

tempo. A autoria é levada a cabo colocando ícones que representam os itens de mídia em uma das trilhas em um tempo específico. Itens podem ser estirados por vários quadros de tempo (manual ou automaticamente, dependendo do sistema de autoria). Alguns sistemas baseados em linha temporal permitem transições de efeitos (como enfraquecer, obscurecer ou transposição espacial de objetos) serem especificadas como propriedades da linha do tempo. A granularidade da linha do tempo pode ser amarrada a granularidade do sistema de relógio ou pode ser uma referência de tempo logicamente definida pelo usuário(Relógios lógicos resolvem problemas de portabilidade de sistemas, mas eles não garantem que uma apresentação particular poderá ser apresentada em todos os sistemas) (BULTERMAN, 1995).

A vantagem principal da linha secular é que ela provê uma ordenação intuitiva de eventos de objetos. As desvantagens dos sistemas de autoria baseados em linha secular incluem os efeitos colaterais de editar objetos, controle de execução e navegação da apresentação.

O sistema MAEstro (DRAPEAU, 1991), QuickTime (APPLE, 1994), Macromedia Director (Macintosh e Windows), Animation Works Interactive (Windows), MediaBlitz! (Windows), Producer (Macintosh e Windows), e a norma HyTime (NEWCOMB, 1991) adotam este paradigma.

3.8.3 Autoria baseada em programação

Ambos paradigmas de autoria baseados em gráficos e em linha secular fazem uso de ilustrações para descrever a interação entre itens de mídia em uma apresentação. Porém estas interações de alto-nível são menos apropriadas para descrições detalhadas de comportamento de sistemas complexos. Um sistema baseado em programação dá para um autor recursos de baixo nível para especificar os componentes, seus tempos, layout e interações dentro de uma apresentação. O esforço de programação pode variar de linguagens rápidas de protótipos (tipicamente chamados de scripts), modelos de biblioteca baseada em objetos, para extensões detalhadas de baixo nível ou substituições de linguagens de programação existentes. Estes modelos têm um poder de expressão muito grande, mas a especificação da composição de um documento multimídia na forma textual é difícil de produzir e modificar (ACKERMANN, 1994; HUDSON, 1993). Além disso, a composição temporal dos componentes é difícil de identificar.

Prover grande flexibilidade e recursos pode tornar o software de difícil manipulação. Uma solução tem sido combinar ferramentas de construção simples, similar aos programas de desenho dirigidos a menus com linguagens scripting. Scripting é a maneira de associar um script, um conjunto de comandos escritos numa forma semelhante a programa de computador, com um elemento interativo numa tela, tal como um botão. Alguns exemplos de linguagens de scripting são Apple HyperTalk para HyperCard, Lingo Macromedia para Director e Asymetrix OpenScript para Toolbook. Essa solução permite aos autores novatos começarem a trabalhar rapidamente em uma apresentação e permite aos autores mais avançados criarem comportamentos personalizados e sofisticados.

3.8.4 Autoria baseada em estrutura

Os três tipos de paradigmas de autoria discutidos anteriormente têm uma característica comum: eles definem documentos em termos da colocação e ativação de grupos de objetos de mídia. Uma outra abordagem é separar a definição da estrutura lógica de uma apresentação e os objetos de mídia associados ao documento. Neste paradigma baseado em estrutura, as vantagens da visão de estrutura e composição usadas na engenharia de software podem ser transferidas ao domínio da multimídia. Esta abordagem antecede a ligação de dados com os documentos, permitindo ao autor desenvolver (e também editar) um esboço da apresentação (BULTERMAN, 1995).

Documentos baseados em texto são desenvolvidos freqüentemente utilizando um modelo de começo-meio-fim, no qual freqüentemente são construídas seções individuais como uma introdução, um corpo de texto principal e um resumo. Documentos inteiros podem ser criados então de cima para baixo ou de baixo para cima (ou uma mistura dos dois modos). A navegação dentro de um documento é relacionada tipicamente com sua estrutura lógica. Em um documento multimídia normalmente existe um local de interação entre objetos de mídia: são definidos assuntos de sincronização entre itens que estão estruturalmente mais relacionados e itens que estão mais separados, na estrutura do documento (BULTERMAN, 1995).

Um exemplo deste paradigma baseado em estrutura é o paradigma **centralizado na informação**, o conteúdo é obtido de informações existentes, se disponíveis. Estas informações são então estruturadas e armazenadas em uma base de dados. A estruturação envolve a divisão da informação em nós e identificadores chaves e em seguida a ligação

destes conceitos (GIBBS, 1991). O WWW (World Wide Web) é um exemplo de um sistema centrado em informação.

O **paradigma baseado em cartões ou páginas** também pertence a essa metáfora baseada em estrutura. Neste paradigma os elementos são organizados em páginas de um livro ou uma pilha de cartões. Ferramentas de autoria baseadas nesse paradigma permitem que o autor ligue as páginas ou cartões formando uma estrutura de páginas ou cartões.

Sistemas de autoria baseados em cartões ou páginas fornecem um paradigma simples para organizar elementos multimídia. Esses tipos de ferramentas de autoria permitem que o autor organize os elementos em seqüências ou agrupamentos lógicos tal como capítulos e páginas de um livro ou cartões em um catálogo.

Sistemas de autoria baseados em páginas são orientados a objeto (APPLE, 1994): objetos são botões, campos de texto, objetos gráficos, fundos, páginas e cartões, e mesmo o projeto em si. Cada objeto pode conter um script, ativado quando ocorre um evento (tal como um clique no mouse) relacionado ao objeto. Exemplos de ferramentas de autoria adotando esse paradigma incluem: HyperCard (Macintosh), SuperCard (Macintosh), ToolBook (Windows) e VisualBasic (Windows).

3.9 Algumas aplicações de ferramentas de autoria

Existe atualmente uma série de ferramentas de autoria, que utilizam o paradigma baseado em páginas ou cartões. As próximas seções apresentam alguns aspectos dessas ferramentas:

3.9.1 KeeBook Creator

KeeBook Creator é uma ferramenta de autoria multimídia que reúne a capacidade de armazenamento e de organização de documentos eletrônicos. Possibilita a criação de *workbook,* que são livros de exercícios que possuem espaços para que estudantes escrevam suas respostas, para auxiliá-los a praticar o que aprenderam. Informações sobre essa aplicação encontram-se em KEEBOOK (2003).

A autoria com a ferramenta KeeBook se dá através da inserção de capítulos no livro, que podem ser visualizados através de abas do lado esquerdo e à direita. A Figura 3-4 ilustra a tela de autoria da ferramenta KeeBook. A construção de cada capítulo se dá através da importação de documentos elaborados por outros softwares e principalmente páginas da internet. Assim sendo, esta ferramenta tem uma grande integração com a internet, fazendo

com que a utilização da mesma seja focada na autoria e visualização on-line na internet. Isso faz com que o usuário da ferramenta tenha suas ligações para páginas da internet juntamente com suas anotações pessoais.

A ferramenta permite também a publicação dos livros produzidos. Ela não permite manipular diretamente as mídias.

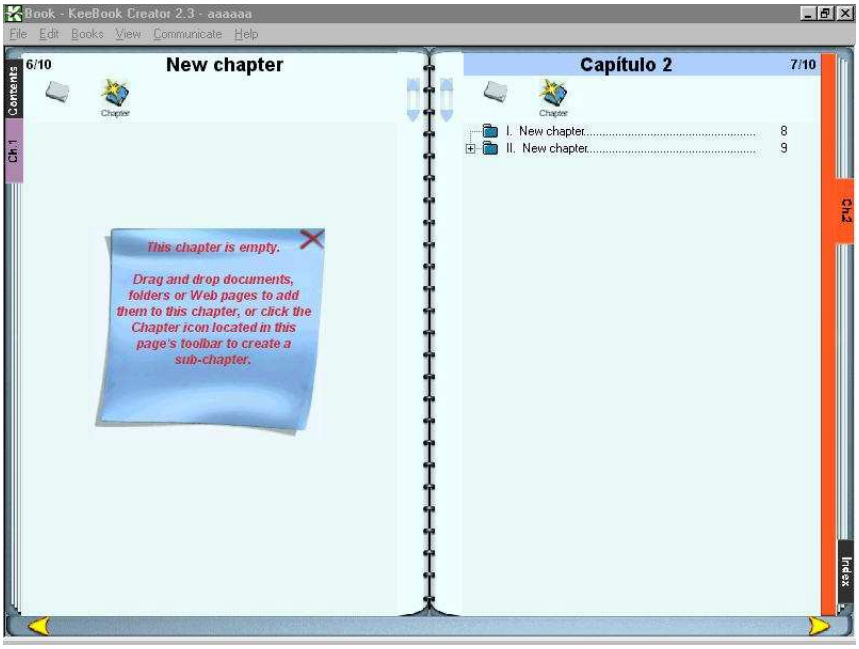


Figura 3-4 Tela de autoria da ferramenta KeeBook Creator

3.9.2 ToolBook

O Toolbook é um progrma que permite a criação de hiperdocumentos multimídia, sem exigir que o utilizador tenha conhecimentos de programação. Uma aplicação feita em Toolbook é "um livro" sobre um determinado tema. Porém, é um livro especial: pode solicitar a intervenção do leitor permitindo-lhe optar por tópicos consultáveis, ou requerer a sua participação mais ativa, como acontece quando se solicita uma resposta a uma pergunta concreta.

A leitura interativa que o Toolbook propõe não é, de forma nenhuma, sequencial. Antes se adapta aos desejos, necessidades do leitor. Permite a exportação de livros para o formato HTML[7]. As páginas do livro são convertidas em páginas HTML podendo ser publicadas na WEB. Há, no entanto, alguns condicionalismos: nem todas as potencialidades do Toolbook podem ser utilizadas.

Os livros são constituídos por páginas nas quais se colocam outros objetos do Toolbook tais como: grafismos (imagens, construções geométricas, etc.), campos (objetos que contêm texto) e botões. Outros objetos do Toolbook são os backgrounds (conjunto de objetos partilhados por várias páginas e que lhes servem de "fundo") e os *viewers* (janelas onde são mostradas as páginas). Na Figura 3-5 pode se observar um esquema representando os componentes de um livro.

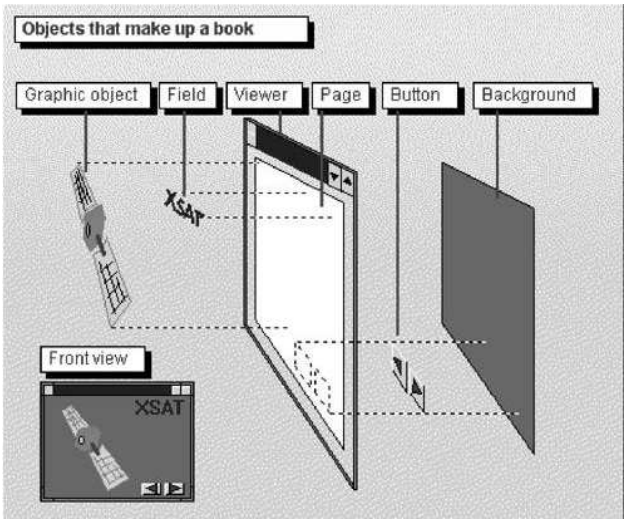


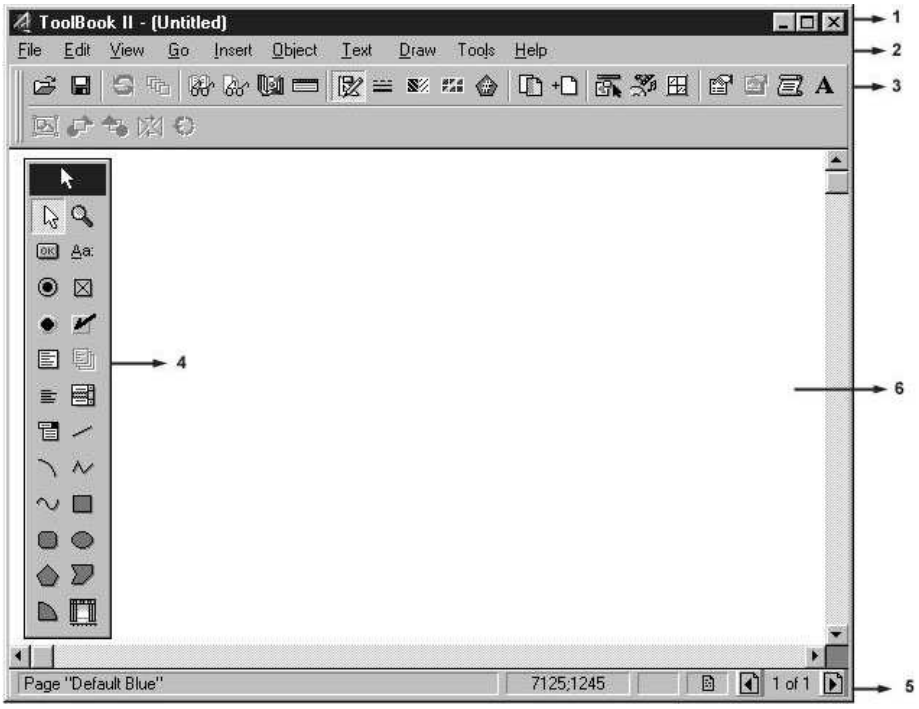
Figura 3-5 - Componentes de um livro

A criação de um livro se dá através da definição de suas páginas, colocando em cada uma delas os respectivos objetos (grafismos, campos e botões) e associando a cada um as ações que pretendemos que eles desencadeiem. Estas ações, caso sejam simples, podem ser definidas pelo autor do livro recorrendo aos menus do Toolbook . Se forem mais complexas têm de ser codificadas em *OpenScript* , que é a sua linguagem de programação. A Figura 3-6 apresenta a interface de autoria do ToolBook.

O Toolbook disponbiliza um conjunto de objetos - incorporados no seu catálogo - a que já estão associados scripts[8]. O utilizador tem apenas que os configurar de acordo com o que pretende realizar. Possui ainda um conjunto de scripts básicos - os auto-scripts - que podem ser facilmente associados a objetos e completados pelo utilizador, em função do resultado que se pretender obter. Mais informações sobre o ToolBook encontra-se em (TOOLBOOK, 2003).

---

[7] HTML é o formato dos documentos do serviço WWW da internet
[8] SCRIPTS = Módulos de programas que definem o comportamento dos objectos.



Legenda:
1 - Barra do Título
2 - Barra de Menus
3 - Barra de Ferramentas
4 - Paleta de Ferramentas
5 - Barra de Estado
6 - Página do Livro

Figura 3-6A Interface do ToolBook

3.9.3 Visual Class

O Visual Class é um programa de autoria que permite a criação de apresentações multimídia. Possibilita a criação de documentos multimídia sem a necessidade de conhecimentos de programação. A autoria acontece através da inserção de diferentes mídias para a construção de telas (VISUAL CLASS, 2003).

Os arquivos de telas são salvos separadamente e é necessário a atribuição de um nome e um código aos mesmos. Após salvas, as telas são incluídas em um projeto. A ferramenta possui um controle dos projetos e também um controle da identificação dos usuários que desenvolveram os mesmos.

O Visual Class possui recursos de inserção de vários efeitos de apresentação dos objetos inseridos nas telas. Oferece opção de alterar em tempo de execução entre as línguas portuguesa e inglesa. Permite também a criação de exercícios pré-montados como: múltipla escolha, liga e associa, verdadeiro e falso e quebra-cabeça, sendo que esse último está ilustrado na

Figura 3-7.

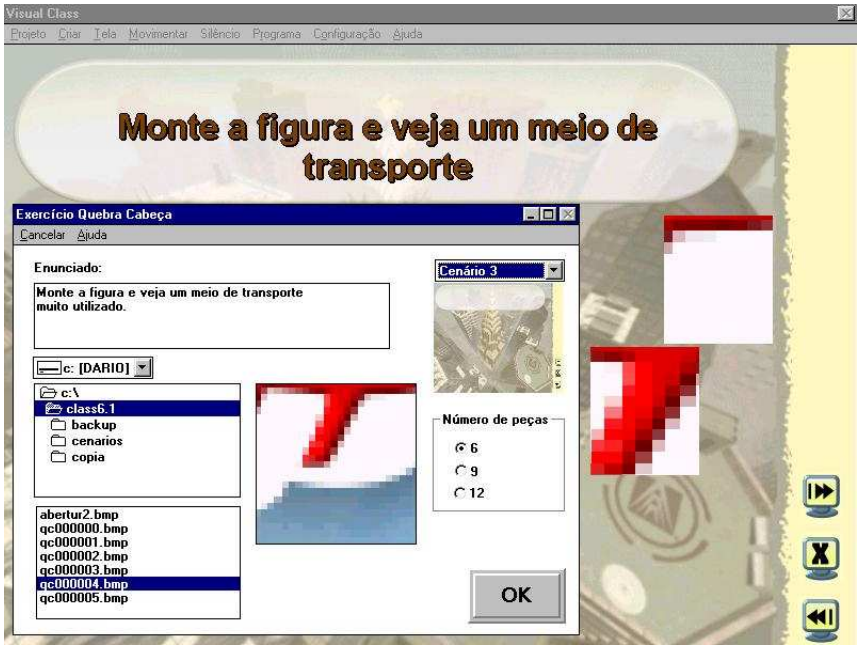


Figura 3-7Autoria de um quebra-cabeças no Visual Class

3.9.4 Everest

O Everest é um programa de autoria, uma espécie de "oficina de criação", equipado com diversas ferramentas que permitem o desenvolvimento de projetos multimídia. Com ele é possível criar aplicações com facilidade, sem necessitar de conhecimentos de programação, agregando elementos como: sons, imagens, vídeo, textos, animações e bancos de dados. A Figura 3-8 ilustra a tela principal de autoria do Everest com uma janela de configuração das propriedades da mídia texto.

Algumas características do programa de autoria Everest (EVEREST, 2003):

- Possibilidade de inserção de imagens (BMP, DIB, JPEG, GIF (animado), PCX, TGA, TIF, WMF, WPG) com mais de 100 tipos de efeitos de transição, textos com fundo transparente e barras de rolagem, rótulos, botões de uma ou duas fases (tipo liga-desliga), animações, vídeos (AVI, MPEG, MOV, SWF-Flash), placares (variáveis cujo conteúdo pode ser visualizado em tela ou servir para inserção de textos pelo usuário), hipertexto, sons (MID, MP3 e WAV) e botões invisíveis;

- Inserção de cursores customizados;
- Inserção de ações de macros que podem ser associadas em grupo ou individualmente a cada recurso disponível como: movimento de imagens, execução de som, mostra/esconde: imagem, vídeo, texto; botão, placar, mapas, etc, desvio de tela, consulta a banco de dados, temporizador, execução de aplicativo externo, impressão de texto, leitura de teclado, teste condicional.
- Possui recursos de acesso a informações de banco de dados (XLS, MDB e DBF) , e também capacidade de relacioná-las com imagens captadas através de scanners ou outros aplicativos.

Inclui ferramentas de criação de questões fanzendo com que tanto alunos como professores podem criar bancos de questões, que geram relatórios completos facilitando a avaliação dos resultados obtidos. Os projetos multimídia criados no Everest podem ser colocados na internet, pois a aplicação gera arquivos em formato HTML. É também possível executar esses projetos em computadores sem a ferramenta de autoria instalada, pois esta gera executáveis que podem ser rodados a partir de disquetes, CDs, e até mesmo em rede. O Everest possui ainda recursos para interfaces de robótica.

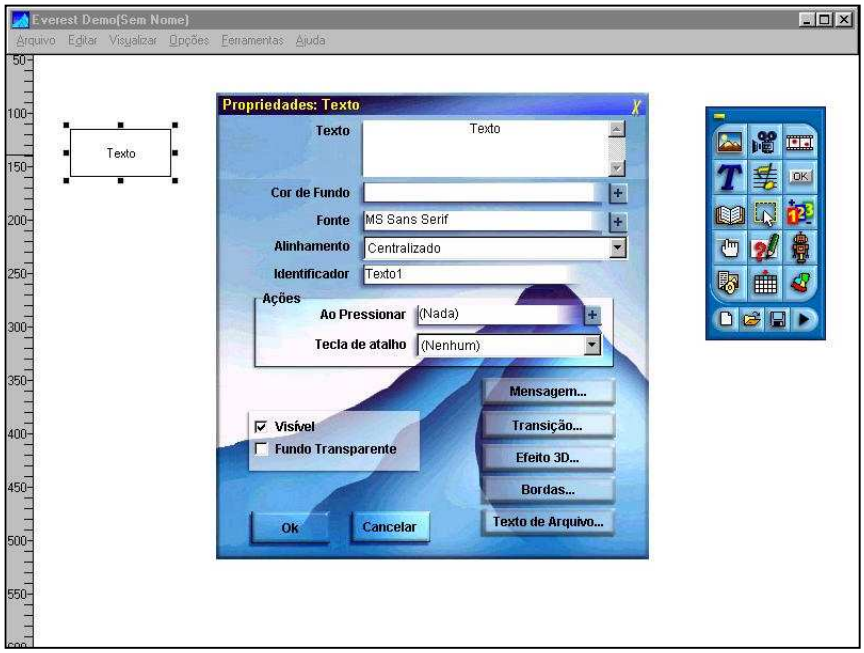


Figura 3-8Configuração de propriedades de uma mídia de texto no Everest

3.9.5 Microsoft PowerPoint

O Microsoft PowerPoint (MICROSOFT, 2003) é um dos mais famosos e usados softwares para a criação de apresentações multimídia.

Uma apresentação multimídia é composta de vários slides. A edição de uma apresentação pode ser feita através de diferentes modos de visualização como normal, estrutura de tópicos, slides, classificação de slides e apresentação de slides. A Figura 3-9 ilustra o modo classificação de slides do Microsoft PowerPoint.

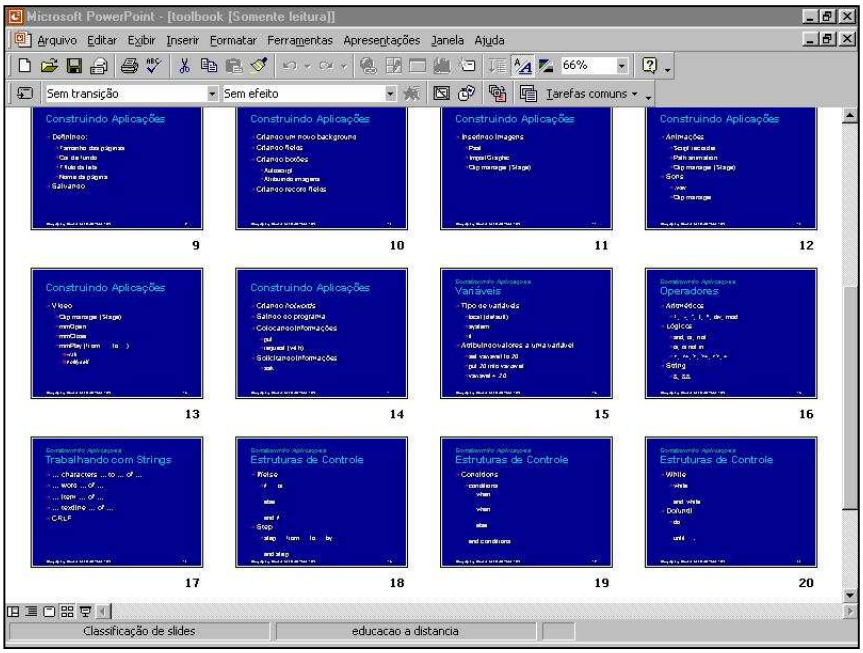


Figura 3-9Modo de classificação dos slides no Microsoft PowerPoint

O Microsoft PowerPoint permite a inserção de caixas de texto, figuras, autoformas, organograma, letras com efeito (WordArt), filmes, sons, gráficos, tabelas e objetos de vários outros programas. Disponibiliza também uma galeria com um substancial número de arquivos de figuras, sons e filmes. Permitindo ainda inserção de cabeçalhos e rodapés com numeração dos slides.

Este software apresenta também recursos para criação de animação dos objetos dos slides e efeitos de transição dos slides. O mesmo permite a autoria colaborativa através da internet; possui um software visualizador, para exibir apresentações sem a necessidade de instalação do mesmo e ainda, permite a criação de ações sobre os objetos possibilitando a criação de hyperlinks para a navegação não-sequencial dos slides da apresentação.

3.9.6 Multimedia Builder

O Multimedia Builder é um programa de autoria voltado para criar aplicações multimídia profissionais (MULTIMEDIA BUILDER, 2003). O mesmo é destinado à criação de pequenos projetos, menus executáveis, navegadores de CD, mp3 Player, CD Player, etc. Sua característica principal é a facilidade de uso. A maior parte de sua tela é a área de projeto, usada para desenhar e empilhar em camadas os objetos multimídia. Permite também o agrupamento de objetos. Com um duplo clique nos objetos poderão ser alteradas suas propriedades, entre elas as interações lógicas e ações do objeto.

Na parte inferior da tela há uma área para as páginas, através da qual se visualiza, adiciona-se e se exclui páginas, como pode ser visualizado na Figura 3-10. Dando um duplo clique na página pode se definir suas propriedades. Na área direita da tela tem-se o navegador de objetos, em que estão listados todos os objetos da página corrente – todos os grupos e relações podem ser vistos nela. Possui também uma linguagem de Script.

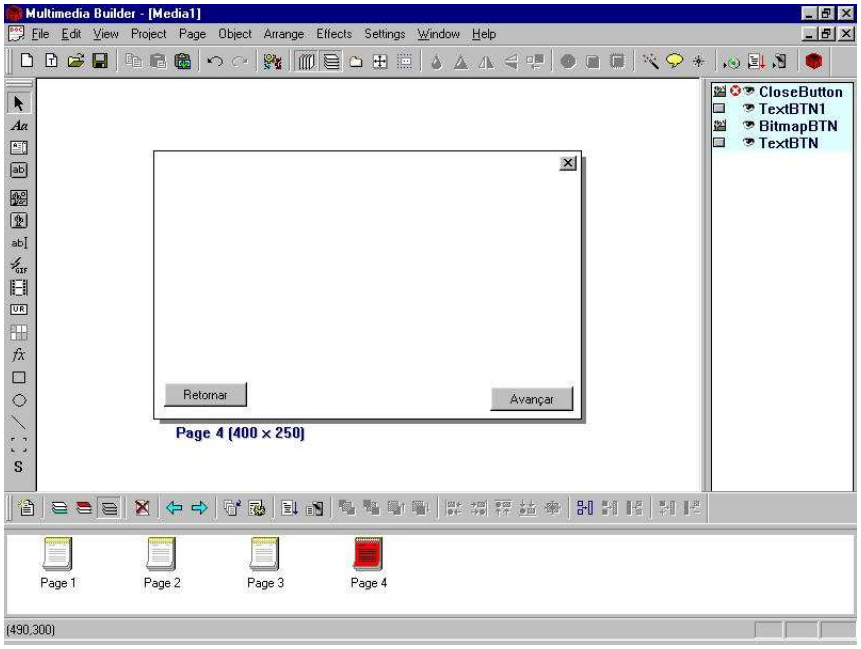


Figura 3-10Tela principal do Multimedia Builder

3.9.7 Perception Quest for Windows

O programa Questionmark Perception é um pacote integrado que permite a criação de testes e questionários para a distribuição on-line. Esses tipos de questões on-line e

exercícios de responder são todos conhecidos como avaliação. Pessoas que passam por avaliações são chamadas de participantes (PERCEPTION, 2003). .

Quando os participantes são avaliados, seus resultados são armazenados em um banco de dados que podem ser usados por administradores para análises e relatórios. Existem várias formas de usar o programa, mas todas elas envolvem os estágios básicos exibidos na Figura 3-11 a seguir.

Figura 3-11Estágios de uso do programa Perception

Estes estágios são:

Autoria (Authoring) – É o processo de compor um banco de questões estruturadas e então selecionar algumas apropriadas para serem dadas para a avaliação de determinados participantes.

Publicação (Publishing) – quando a avaliação foi construída a partir das questões, elas devem ser copiadas para um banco de dados (juntas com alguns arquivos associados) dos quais elas podem ser distribuídas através de um servidor web, ou usadas por participantes que não tem acesso à internet.

Distribuição (Delivery) - os participantes podem agora obter a avaliação, todos através de um acesso a um URL (endereço internet de um programa servidor) utilizando os seus navegadores de internet (se eles tiverem acesso à internet), ou usando um programa Windows especial(se eles não tiverem).

Relatórios (Reporting) – quando os participantes obtiverem suas avaliações, o Perception oferece duas ferramentas poderosas para o relatório e análise dos resultados: Enterprise Reporter e Windows Reporter.

A Figura 3-12 a seguir ilustra todas as opções disponíveis para a criação, distribuição e relatório de avaliações.

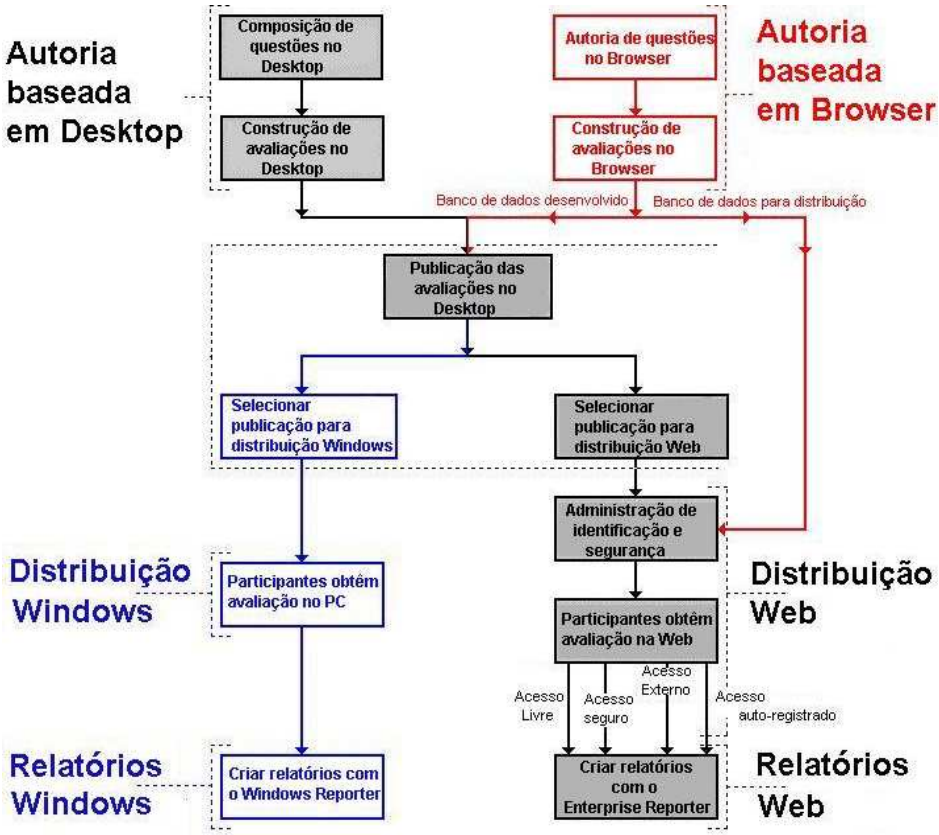


Figura 3-12Diagrama de todo o processo de autoria da ferramenta Perception

A autoria das questões (processo representado pelo primeiro retângulo cinza na Figura 3-12) consiste nas atividades de compor as questões, e organizá-las em uma estrutura de uma árvore de tópicos contendo questões, subtópicos ou ambos. Quando as questões estão prontas, o gerenciador de avaliações possibilita que as mesmas possam ser selecionadas e incluídas em uma avaliação de várias formas. O usuário poderá definir quais questões o

participante verá e em que ordem. A Figura 3-13 ilustra a exibição de uma questão e sua correção ao participante.

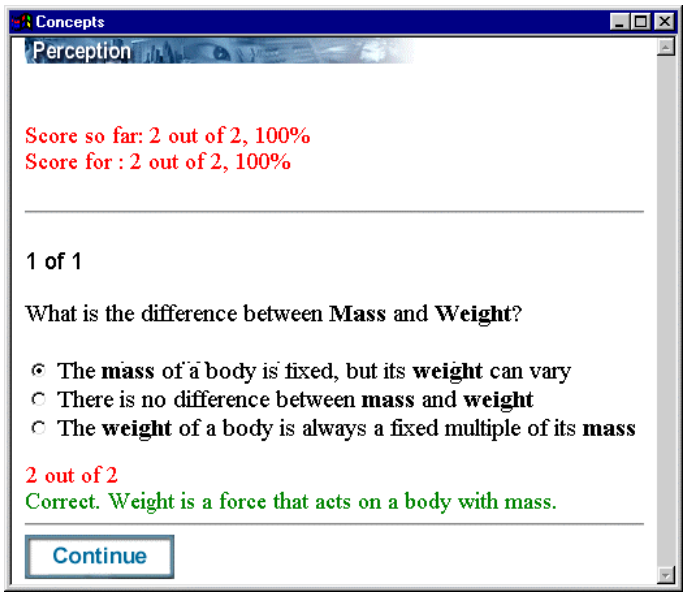


Figura 3-13Exibição de uma questão (com a correção) vista pelo participante

3.10 Considerações Finais

A partir das informações apresentadas neste capítulo, pode-se perceber que à medida em que se populariza o uso dos documentos multimídia para a comunicação através de computadores, novos sistemas (ferramentas) que manipulem esses documentos precisam ser desenvolvidos para tornar cada vez mais eficiente essa forma de comunicação (BULTERMAN, 1995).

Documento multimídia é uma estrutura descrevendo a coordenação e estilo de apresentação de uma coleção de mídias estáticas e dinâmicas (WILLRICH, 1996). Uma ferramenta ideal que manipula documentos multimídia deve seguir alguns requisitos como (ISO 8613, 1998): ser capaz de descrever as relações lógicas (estrutura lógica); representar, como, onde e quando os diferentes componentes serão apresentados (estrutura de apresentação); descrever e manipular as informações que constituem os componentes (estrutura de conteúdo). Além disso, existem vários paradigmas diferentes que podem ser usados pelas ferramentas de autoria para o desenvolvimento de documentos multimídia. Paradigmas como autoria baseada em gráfico, em linha do tempo, em programação e em estrutura podem ser utilizados.

Baseado nos diferentes requisitos e paradigmas anteriormente apresentados pode-se concluir que a atividade, que deve ser desempenhada por uma ferramenta de autoria ideal, não é trivial (BULTERMAN, 1995). Obviamente que desenvolver essas ferramentas constitui-se em uma tarefa mais árdua ainda.

É a partir dessa justificativa, de que existe uma ascenção do uso de documentos multimídia e ao mesmo tempo um aumento da complexidade das ferramentas que manipulem tais documentos, bem como do seu desenvolvimento, é que o autor optou por tratar o problema do desenvolvimento desse tipo de software através da abordagem do reuso. A possibilidade de reutilizar um artefato de software pré-elaborado, sob a forma de um framework orientado a objetos, para facilitar o desenvolvimento de ferramentas de autoria, não somente proporcionará o reuso de projeto e de código, mas também incentivará a pesquisa e criação de novas ferramentas com maior facilidade de uso, mais recursos e mais eficientes. Este capítulo apresentou na seção 3.9 a análise de várias ferramentas de autoria que serviu de subsídio para o desenvolvimento do framework. Esse framework foi denominado de HyperToolBuilder e será apresentado em detalhes no próximo capítulo.

4. FRAMEWORK HYPERTOOLBUILDER

O HyperToolBuilder é um framework orientado a objetos, projetado para facilitar o desenvolvimento de ferramentas de autoria para a criação e edição de documentos multimídia. Ele provê funcionalidades para a construção do ambiente de autoria, a estruturação dos documentos, a persistência e a execução desses documentos.

O desenvolvimeto do framework HyperToolBuilder resultou nos dois produtos seguintes:

Um projeto orientado a objetos genérico que pode ser reutilizado para o projeto e implementação de aplicações de ferramentas de autoria, em qualquer linguagem de programação orientada a objetos. Desenvolvedores de qualquer linguagem de programação orientada a objetos podem reutilizar este artefato ao invés de iniciar as aplicações do zero.

Um protótipo do framework implementado na linguagem JAVA utilizando o ambiente JBuilder 7 (JBUILDER, 2002) - ambiente de desenvolvimento visual orientado a objetos da Borland - com o intuito de validar o framework através do desenvolvimento de duas ferramentas de autoria – HyperBook e HyperTest.

4.1 Considerações iniciais e modelagem de análise do framework HyperToolBuilder

4.1.1 Ferramentas utilizadas

O HyperToolBuilder foi desenvolvido utilizando as seguintes ferramentas computacionais:

- **Ferramenta de Modelagem UML**: foi utilizado o Rational Rose (RATIONAL, 2002) para criar e documentar a modelagem estática e a modelagem dinâmica do framework.
- **Ambiente Visual de Desenvolvimento Orientado a Objetos**: o ambiente de desenvolvimento JBuilder 7 (JBUILDER, 2002) da Borland foi usado para a implementação do framework. Sua escolha se deve ao fato do autor já conhecer outras ferramentas similares do mesmo fabricante, não sendo necessário absorver a cultura de outras ferramentas.

- **Linguagem de programação**: foi utilizado o SDK Java 1.4 da Sun (JAVA 2 SDK, 2003) para a implementação do HyperToolBuilder. O objetivo primordial para a escolha da linguagem Java é o fato dela ser uma linguagem multiplataforma, fazendo com que os desenvolvedores possam criar ferramentas de autoria para diferentes equipamentos e sistemas operacionais. Outra justificativa para o seu uso é o fato dela ser uma linguagem orientada a objetos e por possuir diversos componentes prontos (pacotes AWT, SWING) para a reutilização (JAVA TUTORIAL, 2003). Mais informações sobre alguns componentes Java encontra-se em ZUKOWSKI (1999) e ZUKOWSKI (2001).

- **Componentes gráficos** – além de utilizar exaustivamente os recursos gráficos, principalmente dos pacotes java.awt e javax.swing, disponíveis pela linguagem Java, foi utilizado também o Java Media Framework 2.1.1 – um framework desenvolvido pela Sun para facilitar a manipulação de componentes multimídia como, por exemplo, áudio, vídeo, animação e outros. Maiores informações sobre o JMF 2.1.1 encontram-se em (JAVA MEDIA FRAMEWORK API, 2003).

- **Componente para a definição de fontes de letra** - a possibilidade de formatação de diferentes fontes, estilos e tamanhos de letra é um recurso fundamental para as ferramentas de autoria. Para realizar esta função foi utilizada uma classe de domínio público chamada JFontChooser (JFONTCHOOSER, 2003). Seu objetivo é prover funcionalidades gráficas para a definição de fontes de letras na autoria de documentos multimídia.

4.1.2 Cenário

As ferramentas de autoria são aplicações projetadas para fornecer ferramentas completas e integradas de criação e organização de uma variedade de elementos de mídia como texto, gráficos, imagens, animações, áudio e vídeo a fim de produzir documentos multimídia. O termo ferramenta de autoria refere-se a aplicações gerais para criação de qualquer documento multimídia, em qualquer área do conhecimento. Elas podem ser usadas na área educacional e em outras áreas quaisquer (apresentações multimídia, títulos em CD-

ROM, quiosques, treinamento, visualizações artísticas, páginas Web, etc...). O framework HyperToolBuilder é voltado para auxílio no desenvolvimento dessas aplicações.

4.1.2.1 Indivíduos envolvidos com o framework HyperToolBuilder

Existe uma distinção entre os indivíduos envolvidos no desenvolvimento tradicional de aplicações e desenvolvimento a partir de frameworks (SILVA, 2000). No desenvolvimento tradicional de aplicações existem dois tipos de indivíduo: o desenvolvedor de aplicação e o usuário de aplicação como mostra a Figura 4-1. O desenvolvedor levanta os requisitos de uma aplicação, desenvolve-a e a entrega aos usuários. Os usuários usam a aplicação interagindo com a mesma apenas através de sua interface.

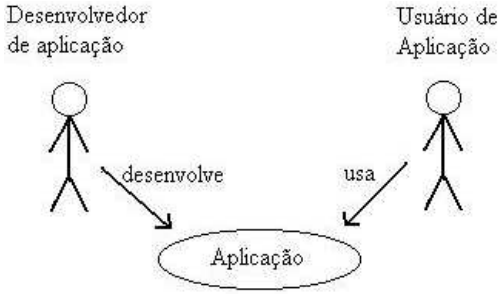


Figura 4-1Elementos do desenvolvimento tradicional de aplicações (SILVA, 2000)

No desenvolvimento de frameworks aparece um outro indivíduo entre o desenvolvedor da aplicação e o usuário: o desenvolvedor de framework. Como pode ser visto na Figura 4-2 o desenvolvedor do framework tem a responsabilidade de produzir e manter frameworks e ensinar como usá-los para produzir aplicações. O desenvolvedor de aplicação passa a ser um usuário de um framework; ele tem as mesmas funções do caso anterior, porém de uma forma diferente: ele produzirá aplicações estendendo e adaptando a estrutura de um framework (SILVA, 2000).

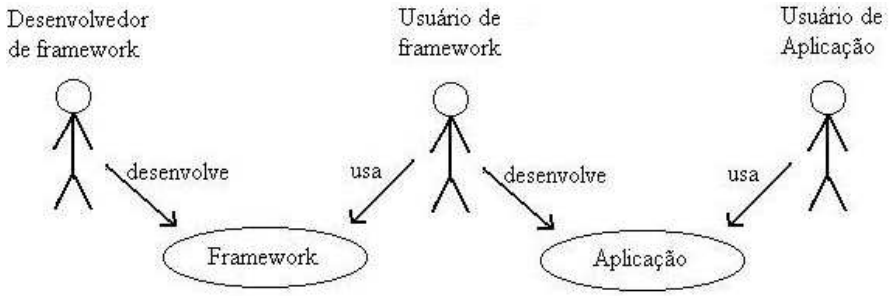


Figura 4-2Elementos do desenvolvimento de aplicações baseado em framework (SILVA, 2000)

Especificamente as aplicações desenvolvidas sobre o framework HyperToolBuilder possuem dois tipos de usuários: o usuário de aplicação que é o autor de documento multimídia e o usuário de aplicação que é o leitor de um documento multimídia. Assim foi convencionado neste trabalho utilizar o termo usuário-autor para representar o usuário que faz uso da ferramenta de autoria para criar um documento, e o termo usuário-leitor[9] representando o usuário que faz uso (executando ou lendo) do documento produzido pelo autor. Dessa forma pode-se representar os elementos envolvidos no desenvolvimento de aplicações (ferramentas de autoria) baseado no framework HyperToolBuilder através da Figura 4-3.

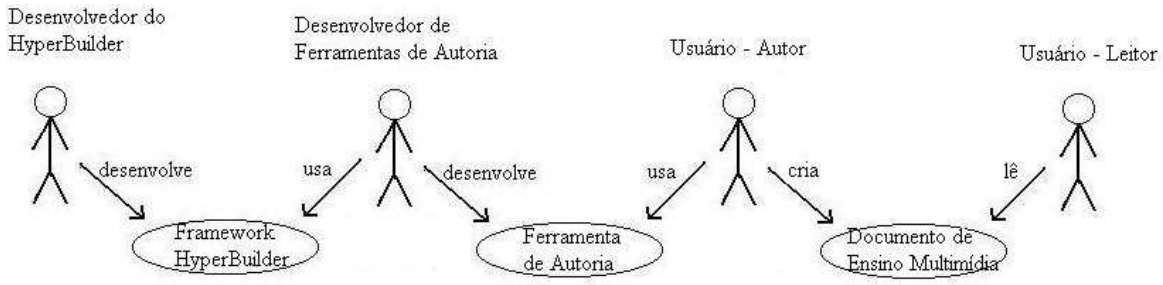


Figura 4-3Elementos do desenvolvimento de aplicações baseado no HyperToolBuilder

A Figura 4-4 ilustra com exemplos como acontece a interação entre os elementos envolvidos no desenvolvimento de aplicações baseado no framework HyperToolBuilder. O desenvolvedor de ferramenta de autoria utiliza o HyperToolBuilder para produzir ferramentas específicas. Em um segundo momento o usuário-autor faz uso da ferramenta produzida para criar diferentes documentos multimídia. Por último, o usuário-leitor lê o documento produzido.

[9] O termo leitor não se refere à leitura de livros convencionais em que o mesmo exerce uma função passiva, mas sim um leitor de documentos multimídia em que o mesmo exerce uma função ativa interagindo com o documento, respondedo exercícios, por exemplo.

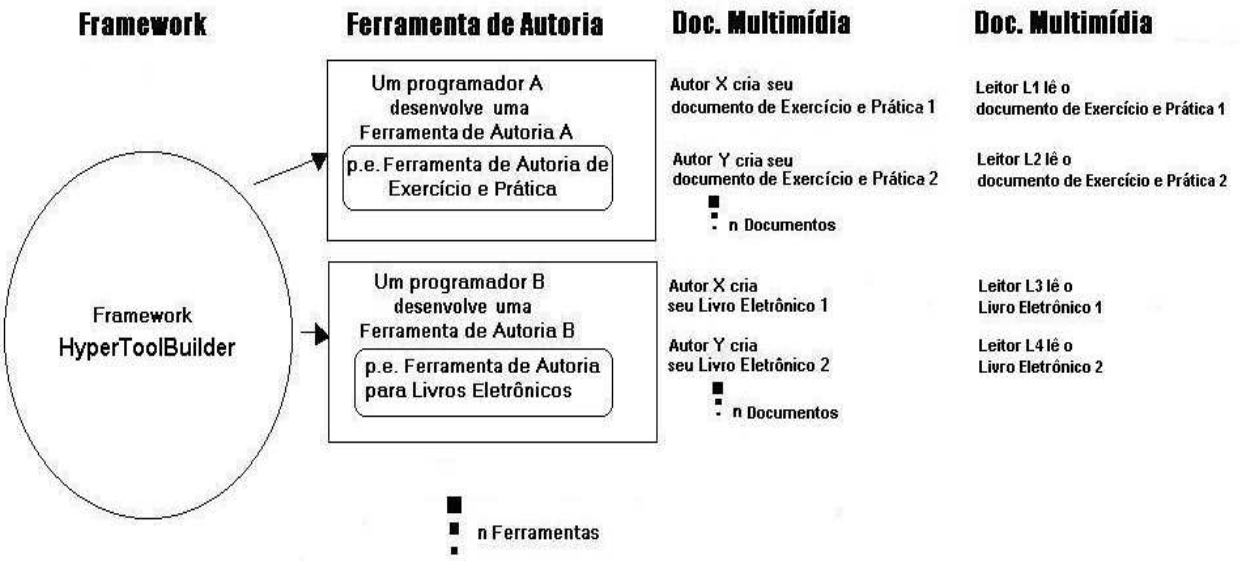


Figura 4-4Elementos envolvidos no desenvolvimento de aplicação baseado no framework HyperToolBuilder

4.1.3 Metodologia de Desenvolvimento

A partir do cenário mencionado na seção anterior, buscaram-se exemplos de ferramentas de autoria acadêmicas e comerciais já existentes para obter conhecimento a respeito do domínio. As principais aplicações analisadas foram Everest 5.0, Visual Class, Microsoft PowerPoint, MultimediaBuilder, KeeBook e Perception Quest for Windows. Estas aplicações são usadas para a criação de vários tipos de documentos multimídia, destancando-se os seguintes: apresentações multimídia, livros eletrônicos, programas de exercício e prática, jogos educativos e tutoriais. Aspectos relacionados às principais ferramentas estudadas foram discutidos na seção 3.9 do capítulo anterior.

Optou-se na etapa inicial do desenvolvimento do HyperToolBuilder pela utilização principalmente da metodologia de Projeto Dirigido por Exemplo, abordado na seção 2.5.1.

A partir da análise e da comparação das aplicações anteriormente citadas foi feita a abstração das funcionalidades comuns para o desenvolvimento do HyperToolBuilder. Para a evolução do framework, após sua primeira versão estar completa, foi utilizada também a metodologia de desenvolvimento de frameworks, chamada Projeto Dirigido por Hot Spot, onde foram identificadas as partes flexíveis na estrutura de classes do framework.

O processo de desenvolvimento do framework aconteceu de forma iterativa e incremental, em que cada ciclo de desenvolvimento (iteração) um conjunto novo de

problemas era analisado de forma que, no final da iteração, o framework incluísse novas funcionalidades (incremento).

Durante o desenvolvimento do framework foi utilizada a técnica de *refactoring*. A referida técnica consiste em modificar o software sem introduzir novas funcionalidades, isto é, visa apenas aprimorar o projeto da solução para facilitar tanto a manutenção do software como a introdução de novas funcionalidades ao mesmo (SILVA,2000).

O HyperToolBuilder é um framework orientado a objetos caixa-cinza (conceito visto na seção 2.3.3 do capítulo 2) e, por isso, sua adaptabilidade está na especialização das classes através de herança e na composição de objetos. Através dos atributos e métodos disponíveis em cada classe da arquitetura disponível nos diagramas, será possível obter uma idéia acerca das funcionalidades da mesma e da interação existente entre os objetos que compõem os diagramas. Toda essa interação ocorre, na abordagem de orientação a objetos, através da troca de mensagens proporcionadas pelos métodos das classes.

Para essa primeira versão desenvolvida do framework, adotou-se o paradigma de autoria baseado em páginas ou cartões (visto na seção 3.8.4) para as ferramentas criadas sob o mesmo.

4.1.4 Análise do Domínio

A análise e definição do modelo de domínio representam a primeira etapa para o desenvolvimento de um framework, como discutido no capítulo 2 (item 2.4). Essa etapa representa o processo de identificação e organização de conhecimento a respeito de uma classe de problemas – um domínio de aplicações – para suportar a descrição e solução destes problemas (ARANGO, 1991).

Uma das decisões iniciais de projeto para a primeira versão do HyperToolBuilder foi optar por ignorar aspectos de suporte a aplicações distribuídas. Isso implicou na impossibilidade de criar ferramentas de autoria colaborativas a partir do framework, permitindo apenas criar ferramentas de autoria *stand-alone*. Esse aspecto está observado na seção 6.3 (trabalhos futuros), como uma possibilidade de evolução futura do HyperToolBuilder para permitir o desenvolvimento de ferramentas colaborativas.

A partir da análise das aplicações de ferramentas de autoria se fez um levantamento dos requisitos comuns entre as mesmas que serviram como requisitos base para a construção

do framework, além disso foram identificados outros requisitos que caracterizaram o nível de abstração a ser fornecido. A seguir são apresentados os principais requisitos que deveriam ser atendidos:

R.1 – Deve ser possível criar diferentes estruturas de documentos multimídia;

R.2 – Deve ser permitida a criação de diferentes ferramentas de autoria de documentos multimídia;

R.3 – Deve ser possível inserir vários tipos de unidades de informação[10] em um documento:

R.3.1 – As diferentes mídias como texto, som, imagem, filme (mídias discretas e mídias contínuas vistas na seção 3.3) deverão ser os principais tipos de unidades de informação, já que os documentos são multimídia;

R.3.2 – Deve ser prevista a possibilidade de associação de ações a qualquer unidade de informação. Quando um usuário estiver executando um documento e clicar sobre uma unidade de informação qualquer, a ou as respectivas ações deverão ser acionadas. Um exemplo seria quando o usuário clicar sobre um botão a ação executar filme será acionada;

R3.3 – Deve ser possível ao usuário do framework criar novos tipos de unidade de informação. O usuário do framework deve ter a possibilidade de criar suas próprias unidades de informação de acordo com as necessidades da ferramenta de autoria por ele desenvolvida;

R.4 – Deve ser possível criar um visualizador de documento;

R.5 – Em sua estrutura deve existir uma separação entre os conceitos do domínio, a visualização e a interação do usuário com as ferramentas de autoria.

4.1.4.1 Casos de uso do framework

A partir do estudo de aplicações e da análise dos requisitos do framework, observaram-se todas as interações exercidas pelo autor com as ferramentas de autoria e

[10] Unidade de informação é um conceito do domínio do framework que representa algo que pode ser inserido em um documento multimídia para compor o mesmo, as unidades de informação podem ser simples como um texto ou uma imagem por exemplo, ou complexas como um filme ou o enunciado de uma questão por exemplo.

chegou-se aos casos de uso do framework. Os casos de uso do HyperToolBuilder estão listados pelo diagrama de casos na Figura 4-5.

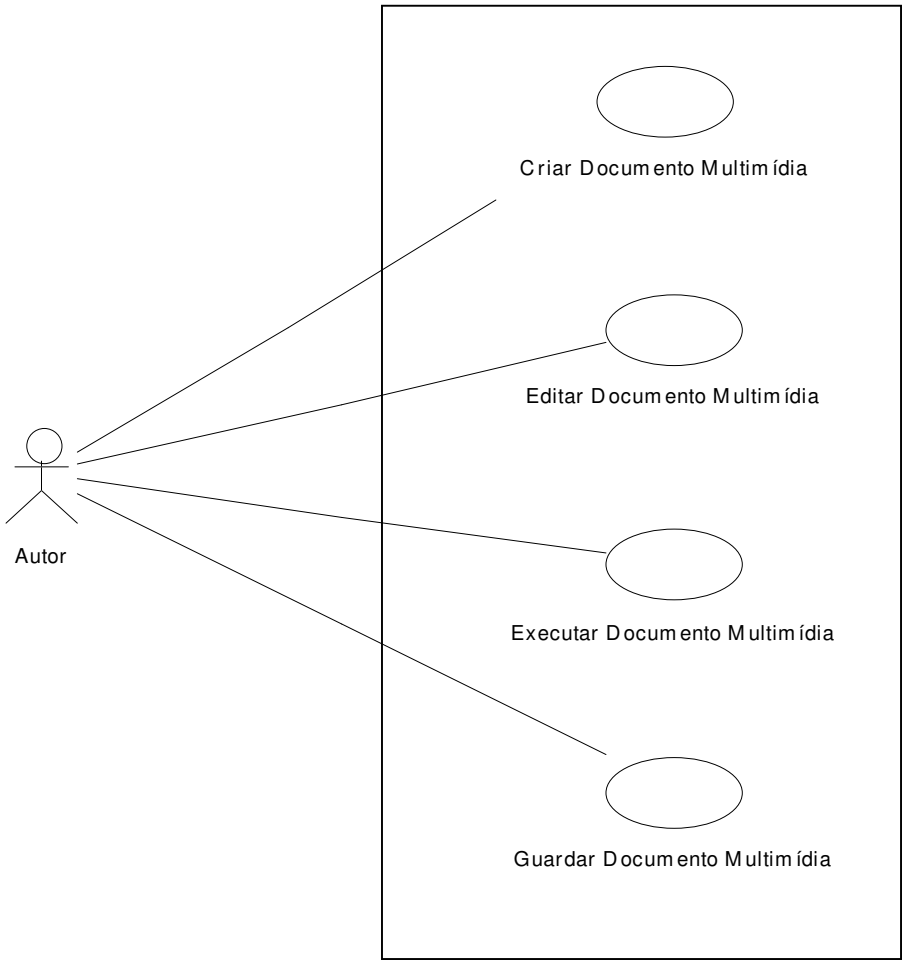


Figura 4-5Diagrama de casos do framework HyperToolBuilder

Aprofundando mais o estudo sobre os requisitos do framework, observa-se que o diagrama da Figura 4-5 está incompleto, pois a edição de documentos envolve outros casos de uso. Estes podem ser classificados de várias formas, uma delas é a classificação em casos de uso de alto nível em casos de uso expandido (LARMAN, 2000). Os casos de uso de alto nível são uma descrição sucinta do processo e são úteis para obter rapidamente alguma compreensão dos principais processos globais. Os casos de uso expandidos mostram mais detalhes que o de alto nível e são úteis para uma compreensão mais profunda dos processos.

Se o caso de uso Editar Documento Multimídia da Figura 4-5 for descrito de uma forma expandida, de acordo com (LARMAN, 2000) percebe-se que o mesmo é uma

generalização de vários outros casos de uso como: Inserir Elemento de Seção, Inserir Unidade de Informação, Configurar Unidade de Informação, Remover Elemento de Seção, etc. Podem existir os seguintes três tipos de relacionamentos entre casos de uso: extensão, generalização e inclusão (OMG, 2001).

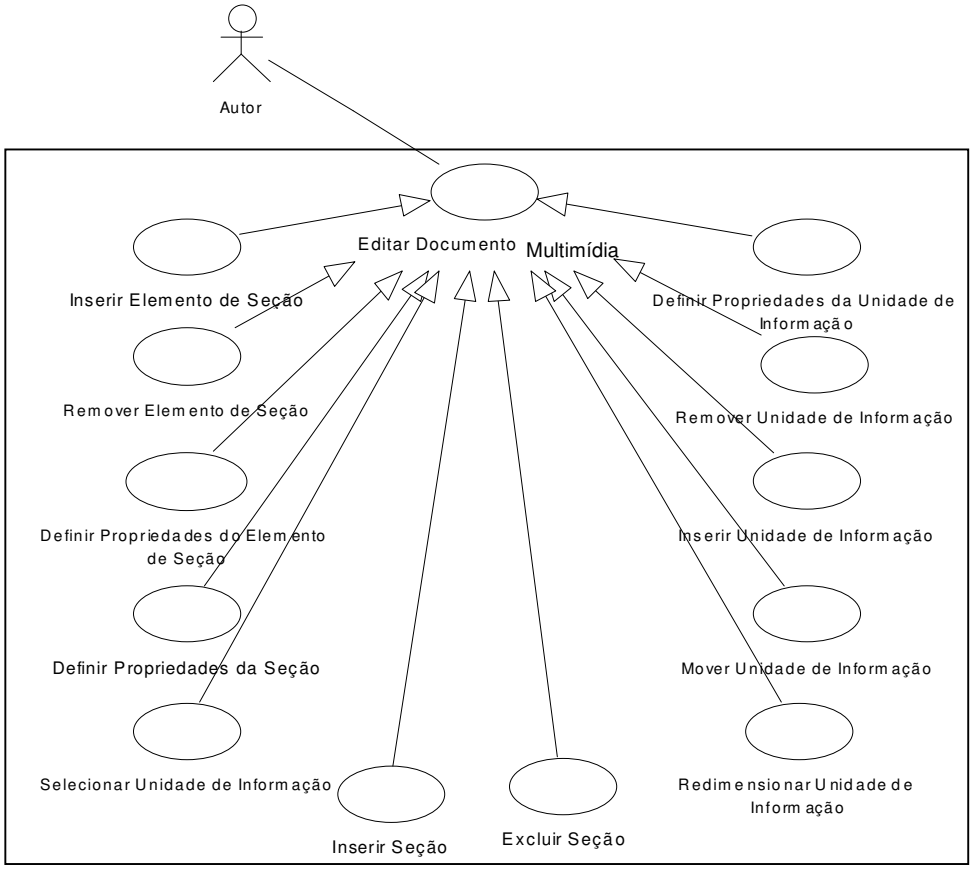


Figura 4-6 Especialização do caso de uso Editar Documento Multimídia

Uma relação de generalização de um caso de uso A para um caso de uso B indica que A é uma especialização de B (OMG, 2001). Baseado nesse conceito criou-se o diagrama de especialização do caso de uso Editar Documento Multimídia, representado pela Figura 4-6. Observando esse diagrama pode-se afirmar que todos os casos da figura que estão à direita, à esquerda e abaixo são casos de uso especializados do caso de uso Editar Documento Multimídia.

Através das abstrações identificadas a partir do processo de análise do domínio, mais especificamente dos casos de uso descritos acima, foram modeladas as classes que compõem a estrutura do HyperToolBuilder, tratada na próxima seção.

4.2 Projeto e implementação do HyperToolBuilder

Essa seção visa apresentar a modelagem do framework HyperToolBuilder. Ressalta as partes de sua estrutura que lhe conferem generabilidade e, também, as partes que são flexíveis para suportar o desenvolvimento de ferramentas de autoria para documentos diferentes.

A modelagem de qualquer sistema – framework ou aplicação - deve ter a capacidade de descrever o mesmo sob uma ótica estática e sob uma ótica dinâmica. A modelagem estática de um sistema orientado a objetos representa a descrição dos elementos desse sistema, descrevendo as classes, seus atributos e relacionametos. A modelagem dinâmica de um sistema orientado a objetos deve levar à identificação e descrição da colaboração entre objetos e dos métodos das classes (SILVA, 2000).

A seção 4.2.1 desse capítulo tem como objetivo apresentar a modelagem estática do framework HyperToolBuilder, observando que, por motivos de brevidade e complexidade do mesmo, optou-se por apresentar apenas as classes e seus relacionamentos, desconsiderando os atributos e métodos. A seção 4.2.2 traz a modelagem dinâmica do HyperToolBuilder, mostrando as colaborações entre os objetos e os métodos das principais classes do framework.

4.2.1 Modelagem Estática

O processo de modelagem de um framework insere um conjunto de requisitos de modelagem não atendidos por metodologias OOAD em geral. Por isso serão usadas algumas convenções criadas por SILVA (2000) para explicitar na documentação de projeto as partes flexíveis do framework. As convenções adotadas são as seguintes:

- Redefinibilidade de classe – estabelece se as classes podem ou não originar subclasses nas aplicações nos artefatos desenvolvidos sob o framework. O usuário do framework poderá criar subclasses apenas de classes explicitadas como redefiníveis. As classes redefiníveis serão representadas pelo estereótipo R no diagrama de classes.
- Essencialidade de classe – apenas as classes redefiníveis podem ser classificadas como essenciais. Classe essencial quer dizer que todo o artefato de software produzido a partir do framework deve utilizar esta classe (ou uma subclasse dessa). As classes essenciais são obrigatoriamente redefiníveis e elas serão representadas pelo estereótipo RE nos diagramas.

- Conceito externo – é um conceito criado fora do contexto do framework. Uma classe externa, por exemplo, denota uma classe referenciada pelo framework, que pode inclusive gerar subclasses. Classes externas serão definidas pelo estereótipo EXTERNAL.

A partir dos requisitos identificados no capítulo anterior e das considerações a respeito das convenções adotadas para a modelagem de frameworks, iniciou-se o projeto da estrutura de classes que compõe o HyperToolBuilder. A Figura 4-7 apresenta um diagrama de classes da estrutura arquitetural do framework.

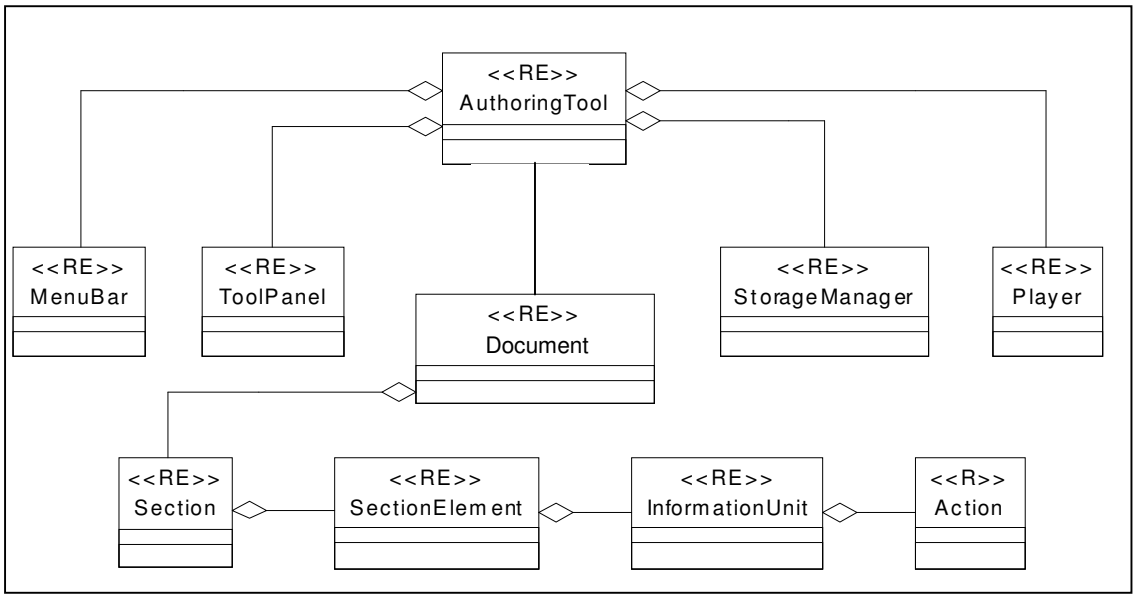


Figura 4-7Diagrama da estrutura arquitetural do framework HyperToolBuilder

Observando o diagrama de classes anterior, pode-se ter um primeiro entendimento e uma visão geral do HyperToolBuilder. Toda a aplicação desenvolvida sob o framework deverá especializar a classe *AuthoringTool,* que é responsável por centralizar todo o controle da ferramenta de autoria. Ela agrega as classes *MenuBar, ToolPanel, Document, StorageManager* e *Player*. A classe *Menubar* é responsável por receber e executar os comandos disponíveis pela aplicação. *ToolPanel* corresponde ao painel que contém as ferramentas respectivas de cada aplicação. Através dele o autor seleciona uma destas ferramentas para a edição de um documento. A classe *StorageManager* possui os recursos necessários para a persistência dos documentos produzidos. *Player* corresponde a classe responsável pela execução dos documentos. Finalmente a classe *Document* representa o

documento multimídia que a ferramenta de autoria é capaz de manipular (criar, editar, excluir, imprimir, executar).

A estrutura de um documento pode ser subdividida em unidades chamadas seções, percebe-se isso na Figura 4-7 através da agregação de *Section* em *Document*. As seções por sua vez serão subdivididas em unidades menores chamadas de Elementos de Seção. A classe *Section* possui uma agregação de *SectionElement*.

O autor tem como tarefa principal no processo de edição de um documento a inserção de unidades de informação (mídias, por exemplo) nos elementos de seção (páginas, por exemplo). Por isso a classe *SectionElement* agrega *InformationUnit*.

Outra tarefa importante no processo de autoria é a possível definição de ações que podem ser executadas a partir de um evento ocorrido (clique, por exemplo) sob uma unidade de informação. Observa-se isso na Figura 4-7 na agregação de *Action* em *InformationUnit*. Estas classes representadas na Figura 4-7 representam a arquitetura básica do HyperToolBuilder. A idéia geral do framework está baseada nessa estrutura, porém para aprofundar melhor os entendimentos do mesmo optou-se por organizar e dividir essa estrutura central em vários diagramas de classes menores e mais detalhados.

Serão apresentados e discutidos a seguir os seguintes diagramas de classes que formam o framework:

- Diagrama da estrutura de documentos multimídia;
- Diagrama da estrutura da ferramenta de autoria;
- Diagrama dos tipos de unidade de informação;
- Diagrama das ações sobre as unidades de informação;
- Diagrama da separação entre os conceitos do domínio, sua visualização e controle da interação do usuário;
- Diagrama da estrutura da ferramenta de visualização.

4.2.1.1 Estrutura de documentos multimídia

As ferramentas de autoria constituem um tipo de aplicação de manipulação de documentos multimídia. O framework HyperToolBuilder suporta a definição de diferentes estruturas de documentos. Na aplicação HyperBook, por exemplo, desenvolvida sob o

framework e apresentada no capítulo 5, o documento manipulado é um livro que possui uma determinada estrutura, já na aplicação HyperTest, apresentada também no capítulo 5, o documento manipulado é um teste que possui uma outra estrutura. Para suprir a flexibilidade necessária para suportar diferentes tipos de documentos multimídia, parte da estrutura do framework HyperToolBuilder corresponde à definição de uma estrutura de documento genérica, a ser estendida para a produção de estruturas específicas.

A Figura 4-8 mostra a estrutura hierárquica das classes do framework HyperToolBuilder que dá suporte à definição de documentos multimídia.

A abordagem adotada para o desenvolvimento do framework prevê três tipos de documento: documento de apresentação de informações, documento de prática de informações e documento composto pelos dois anteriores. Baseado nisso, todo documento deve ser uma instância de subclasse de uma das seguintes classes abstratas: *Presentation*, *Practice* ou *Tutorial*. Essa abordagem citada anteriormente foi uma decisão de projeto do framework e foi baseada em ALESSI (2001).

Como se pode observar na Figura 4-8 foi utilizado o padrão de projeto *Composite* (GAMMA, 2000) para a definição da classe *Tutorial*. Através dela se possibilita ao usuário do framework a criação de um documento híbrido, ou seja, um documento composto por uma mescla de vários outros documentos de qualquer tipo – apresentação, prática ou até por outros tutoriais.

Todo documento agrega uma ou mais seções, instâncias de subclasses de *Section*, que por sua vez agrega um ou mais elementos de seção, instâncias de subclasses de *SectionElement*. *Section* possui uma subclasse chamada *SectionComposite* que implementa o padrão de projeto *Composite* (GAMMA, 2000), para possibilitar a criação de seções compostas e complexas. Isso permite a utilização de diferentes tipos de seção em um mesmo documento. Permite também que um documento possa ter ilimitadas dimensões, ou seja, podem ser criadas seções de seções e assim sucessivamente.

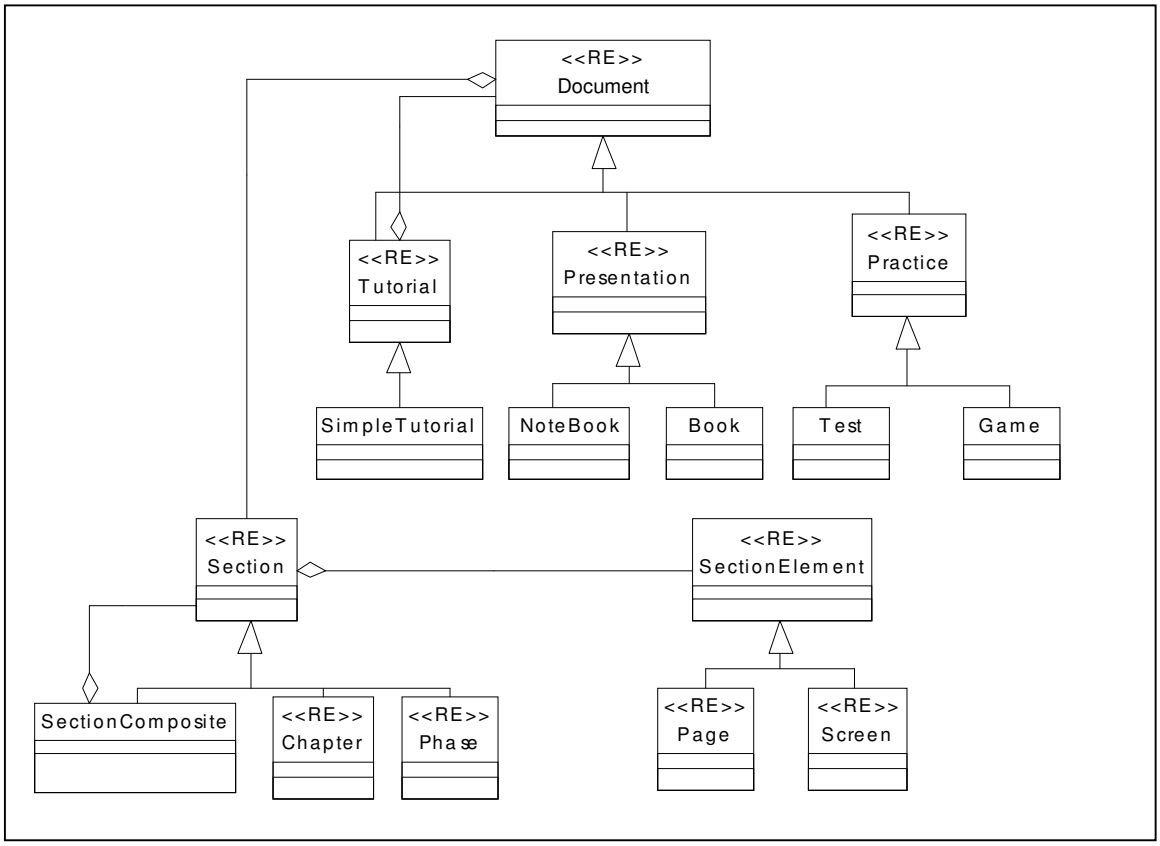


Figura 4-8Diagrama de classes que define a estrutura de um documento

Produzir uma estrutura de documento consiste em definir os elementos de seção (subclasses concretas de *SectionElement*), as seções (subclasses concretas de *Section*) e o documento (subclasse concreta de *Presentation* ou *Practice* ou *Tutorial*).

4.2.1.2 Estrutura da ferramenta de autoria

A classe responsável pelas funcionalidades de controle de uma ferramenta de autoria criada sob o framework HyperToolBuilder pode ser vista na Figura 4-9.

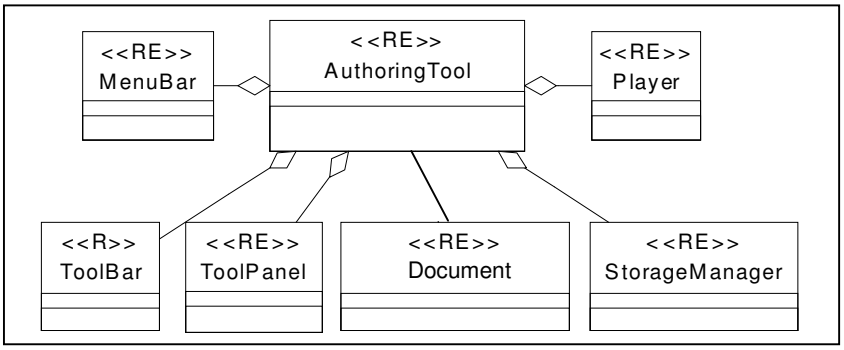


Figura 4-9Diagrama de classes da ferramenta de autoria

Uma ferramenta de autoria específica é produzida a partir da criação de uma subclasse concreta de *AuthoringTool*. Como pode ser visto na Figura 4-9, em uma subclasse de *AuthoringTool* são definidas as seguintes características de uma ferramenta de autoria específica:

- Qual é o documento tratado pela ferramenta de autoria – por isso ela associa um *Document*;
- Quais os comandos disponíveis pela ferramenta de autoria – os comandos podem ser executados por meio de menus (por isso agrega a classe *MenuBar*) ou por meio de botões de atalho(por isso agrega a classe *ToolBar*);
- Quais as ferramentas disponíveis para manipular um documento multimídia – agrega a classe *ToolPanel* que por sua vez agrega as respectivas ferramentas;
- Qual o mecanismo de armazenamento de documentos utilizado pela ferramenta de autoria - *StorageManager*;
- Qual o mecanismo de execução do documento – *Player*.

4.2.1.3 Tipos de unidades de informação

A atividade do processo de autoria utilizando as ferramentas criadas a partir do framework HyperToolBuilder consiste basicamente em inserir, modificar ou remover unidades de informação em um elemento de seção. A Figura 4-11 apresenta o diagrama de classes que mostra as unidades de informação que estão previstas no framework e que podem ser agregadas em um elemento de seção. Uma instância de subclasse concreta de *SectionElement* poderá agregar várias instâncias de subclasse concreta de *InformationUnit*.

Toda unidade de informação tem associada uma moldura para melhor visualização da mesma durante o processo de autoria. Uma subclasse concreta de *InformationUnit* terá sempre associada uma instância de *Edge*. A moldura por sua vez terá sempre três áreas quadriculadas (*handles* que podem ser vistos na Figura 4-10) para edição da unidade de informação, uma no canto esquerdo superior para mover a unidade de informação, a segunda no canto esquerdo inferior para redimensionar para esquerda e para baixo e a terceira no canto direito superior para redimensionar para direita e para cima. Como pode ser visto na Figura 4-11 uma instância da classe *Edge* agrega sempre três instâncias da classe *Handle*.

A Figura 4-10 apresenta um exemplo da exibição de uma unidade de informação[11] no modo de autoria de um documento multimídia. A visualização de todas as unidades de informação previstas pelo framework terá a moldura e os três *handles* apresentados na figura.

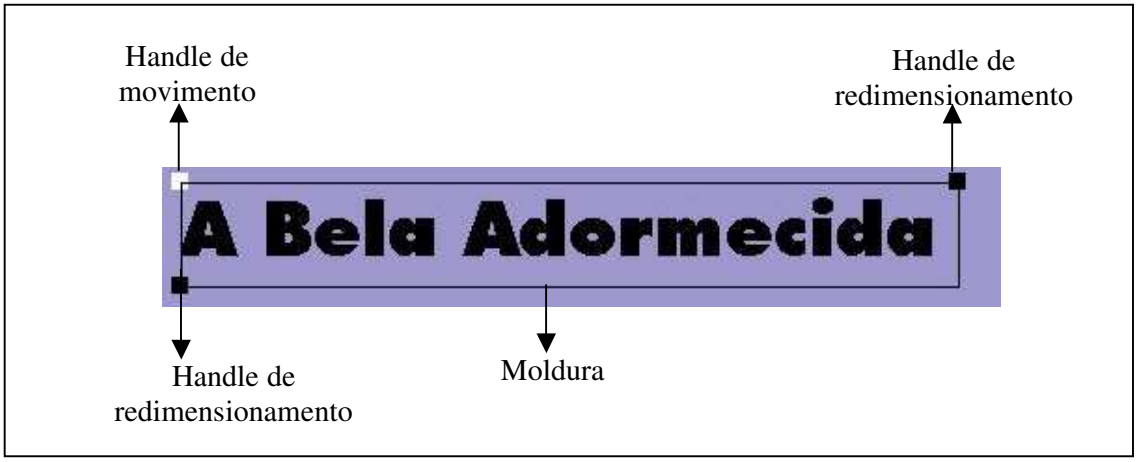


Figura 4-10 Visualização da moldura e dos handles de uma unidade de informação

Unidade de informação é uma abstração que representa o grupo de todos os elementos que podem ser inseridos em um elemento de seção. Possui as subclasses Mídia, Questão, Alternativa, Marcador, Objeto de Jogo que representam subgrupos específicos de unidades de informação previstos a partir do estudo das aplicações exemplo. Como pode se observar todos estes subgrupos são *hotspots* do framework, são classes abstratas que permitem a criação de novas subclasses não previstas.

A subclasse *Media* representa um dos conceitos mais importantes das ferramentas de autoria a serem criadas sob o HyperToolBuilder, já que a mesma possuirá subclasses para representar as principais mídias (texto, imagem, som, filme) de um documento multimídia. E um dos requisitos do framework é a produção de documentos com recursos multimídia (requisito R.3.1 visto na seção 4.1.4), ou seja, fazendo o uso de vários tipos de mídia, conceitos discutidos no capítulo 3 (seções 3.1 e 3.3). A Figura 4-11 mostra a classe *Media* que é uma classe abstrata e possui as subclasses *StaticMedia* e *DynamicMedia,* também abstratas, que representam os grupos das mídias estáticas e dinâmicas que são respectivamente sinônimos dos conceitos de mídias discretas e contínuas, vistos na seção 3.3. O anexo II apresenta a hierarquia de classes das diferentes mídias previstas pelo framework.

[11] O exemplo foi extraído da aplicação HyperBook desenvolvida sob o framework HyperToolBuilder e apresentada no capítulo 5. A unidade de informação apresentada corresponde a uma mídia do tipo texto de uma

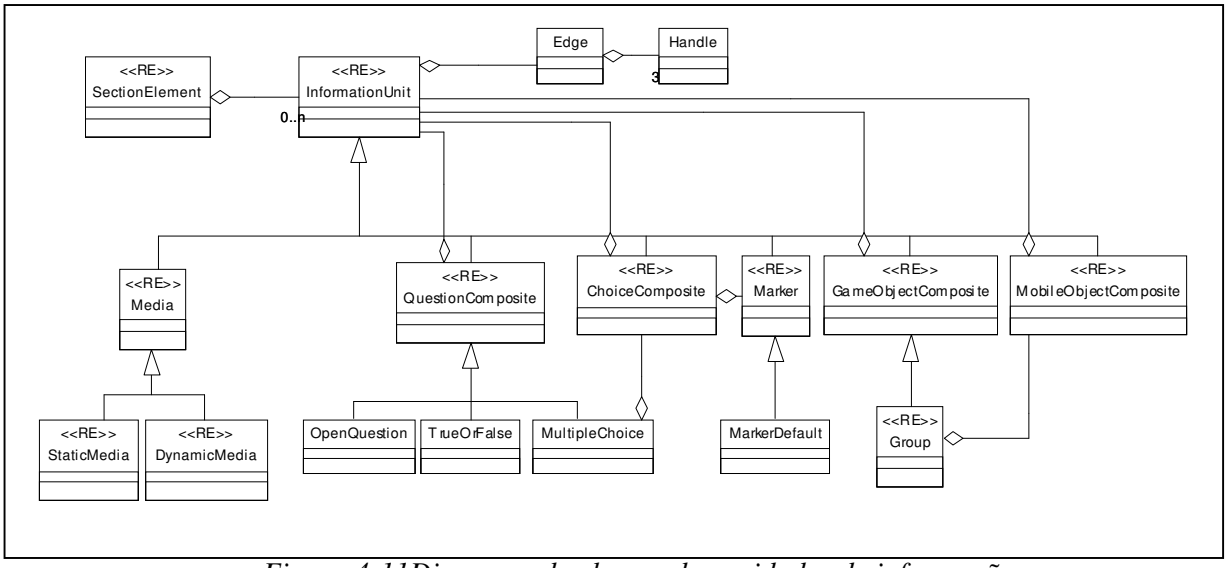


Figura 4-11Diagrama de classes das unidades de informação

As outras subclasses de unidade de informação surgiram a partir da identificação dos aspectos (hotspots) que diferem entre algumas aplicações de ferramentas de autoria estudadas. As classes *QuestionComposite, ChoiceComposite* e Marker surgiram para cobrir requisitos das ferramentas de autoria para exercícios. *Question* representa o enunciado de uma questão, *Choice* representa as possíveis alternativas de uma questão e Marker é um marcador para representar a visualização da seleção de uma alternativa pelo usuário. *Question* e *Choice* implementam o padrão de projeto *Composite* (GAMMA, 2000), pois a construção de enunciados e alternativas de uma questão se dá através da inserção e edição das diferentes mídias, que também são unidades de informação.

As classes *GameObjectComposite* e *MobileObjectComposite* surgiram no estudo de uma ferramenta de autoria para um tipo de jogo. A idéia do mesmo é existir grupos representados por uma instância de subclasse de *Group* (*Group* é subclasse de *GameObjectComposite*) e vários objetos que podem ser movidos, representados por uma instância de subclasse de *MobileObjectComposite*. A atividade do jogo é mover os objetos para os seus respectivos grupos. Existem inúmeros tipos diferentes de aplicações de jogos. Por motivos de brevidade e tempo para concluir a primeira versão do framework foi utilizado apenas esse um tipo de jogo descrito acima. Seria muito importante para o aprimoramento do framework analisar e envolver futuramente um vasto número de jogos no mesmo.

---

só linha (frase).

As classes *GameObjectComposite* e *MobileObjectComposite* também implementam o padrão *Composite,* permitindo que os grupos e o objetos móveis possam ser objetos complexos. Qualquer grupo ou objeto móvel de um jogo pode ser composto por quaisquer subclasse de *InformationUnit*, inclusive por outros grupos ou objetos móveis.

4.2.1.4 Ações sobre as unidades de informação

De acordo com o requisito R.3.2 deve ser possível associar ações às unidades de informação. A Figura 4-12 apresenta as classes do framework que representam a solução de projeto para o cumprimento desse requisito.

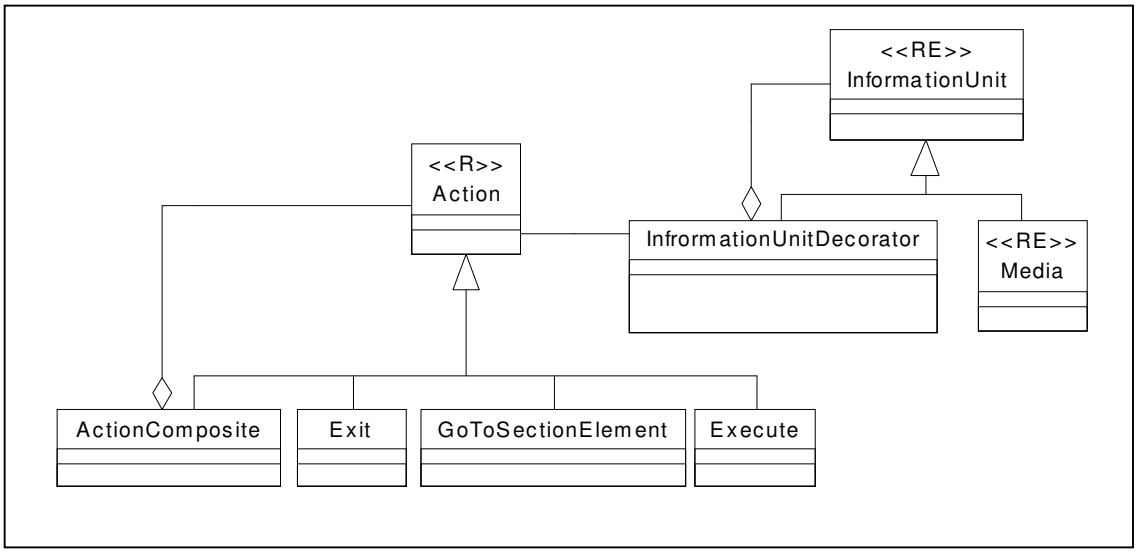


Figura 4-12Diagrama da associação das ações às unidades de informação

A classe *InformationUnit* possui uma subclasse concreta chamada *InformationUnitDecorator* que implementa o padrão de projeto *Decorator* (GAMMA, 2000) e está associada a uma *Action*. Dessa forma se consegue anexar responsabilidades adicionais, nesse caso associar uma ou mais ações dinamicamente a uma instância concreta de subclasse de unidade de informação. Como a ação é uma decoração de uma unidade de informação, sua presença é transparente para seus objetos clientes.

Uma unidade de informação pode ter associada a ela não apenas uma, mas inúmeras ações, isso é possível, pois a classe *Action* possui uma subclasse chamada *ActionComposite* que implementa o padrão de projeto *Composite*. Com o uso desse padrão de projeto se permite que um *InformationUnitDecorator* possa associar um objeto que pode ser composto por várias ações.

Além da classe *ActionComposite* a Figura 4-12 apresenta algumas outras subclasses concretas de *Action* para ilustração. As classes *Exit, GoToSectionElement* e *ExecuteSound* representam respectivamente as ações: sair de um documento, ir para o próximo elemento de seção e executar um determinado som. O anexo III traz o diagrama de classes com as ações previstas no framework a partir da análise das aplicações exemplo.

A associação da classe abstrata *Action* com classe *InformationUnitDecorator* caracteriza o padrão de projeto *State* (GAMMA, 2000). Este padrão de projeto permite a um objeto alterar seu comportamento quando o seu estado interno mudar, o objeto irá aparentar mudar de classe. Um objeto da classe *InformationUnitDecorator* dessa forma poderá assumir vários estados diferentes, dependendo da ou das ações – subclasses concretas de *Action* -que estarão associadas a ele. Com a utilização dos padrões *Decorator* e *State* se possibilita associar várias funcionalidades diferentes a uma unidade de informação já que se consegue decorar essa unidade de informação e fazer com que essa decoração possa mudar de estado.

5.2.1.5 Estrutura da separação entre os conceitos do domínio, sua visualização e controle da interação do usuário

A interface gráfica foi considerada como um dos aspectos relevantes durante a realização da análise e projeto do framework HyperToolBuilder, em função de sua importância para as aplicações do domínio. A interface gráfica é importante para essas aplicações, pois sua facilidade de uso é um fator que afeta o número de potenciais usuários das mesmas. Ferramentas de autoria que manipulem documentos multimídia devem fazer uso extensivo de interfaces gráficas amigáveis ao usuário, complexas e flexíveis. Em função disso se fez necessário projetar e programar essas interfaces e aplicar os mesmos princípios de orientação a objetos que foram aplicados aos conceitos do domínio. Freqüentemente, as interfaces gráficas com o usuário são simplesmente reunidas e jogadas no sistema como uma cogitação posterior (SINTES, 2002).

Uma interface flexível deve apresentar as seguintes características:

Alterabilidade – refere-se à capacidade de alterar a funcionalidade presente, sem conseqüências imprevistas sobre o conjunto da estrutura;

Extensibilidade – refere-se a ampliar a funcionalidade presente, sem conseqüências imprevistas sobre o conjunto da estrutura.

A melhor maneira de obter flexibilidade é projetando o framework de uma forma que exista um desacoplamento entre os elementos do domínio do problema e os elementos de interface (apresentação visual) com o usuário. Interfaces gráficas desacopladas são importantes pois permitem ao mesmo sistema possuir interfaces diferentes, muitas vezes não relacionadas. Além disso, a interface com o usuário pode mudar com o amadurecimento de uma aplicação, no decorrer do tempo. Para facilitar essas atualizações da interface é necessário que o projeto da mesma seja flexível e aceite alterações prontamente.

Dessa forma, o HyperToolBuilder dá flexibilidade para que se crie diferentes interfaces com o usuário no desenvolvimento de uma ferramenta de autoria. Além disso, facilita as futuras alterações das interfaces gráficas das ferramentas de autoria criadas sob o mesmo, sem ter de fazer alterações nas classes que compõem o domínio do problema.

Um projeto desacoplado permite testar os recursos de uma aplicação, mesmo que não se tenha terminado de desenvolver a interface gráfica com o usuário. E também permite que se aponte precisamente os erros que são da interface e os que são da aplicação (SINTES, 2002).

Para conseguir esse desacoplamento optou-se por utilizar o modelo de interação MVC (Model View Controller) que é descrito em LEWIS (1995). Esse modelo fornece uma estratégia para o projeto de interfaces com o usuário que permitem desacoplar os elementos do domínio do problema (Model) dos elementos da interface com o usuário (View). A utilização deste modelo no desenvolvimento do framework cumpre o requisito R.5 visto na seção 4.1.4.

A Figura 4-13 apresenta o diagrama de classes mostrando a abordagem de utilização do framework MVC para desacoplar as classes que compõe os elementos do domínio do problema(model) das classes de interface (view) do HyperToolBuilder. Toda interface com o usuário que possui interação pode ser dividida em dois conceitos: o conceito da visão -view – que representa a apresentação visual da interface e o conceito do controlador -controller - que é responsável por controlar a interação do usuário com a interface. Esses dois conceitos que formam a interface são representados pelas classes View e Controller e podem ser visualizados na Figura 4-13. Dessa forma todo elemento que compõe um documento é uma subclasse de *Model.* Se esse elemento será de alguma forma visualizado pelo usuário, o desenvolvedor da aplicação deverá criar uma subclasse de *View* responsável pela visualização

do elemento. Na Figura 4-13 foram utilizadas algumas classes do framework para ilustrar o uso do modelo MVC, *SectionElement* é um conceito (classe) que compõe a estrutura de um documento, logo, deve ser uma subclasse da classe externa do framework MVC chamada *Mode*l e como *SectionElement* possuirá uma visualização deve ter a respectiva subclasse da classe externa *View,* que no framework se chama *SectionElementView*.

Se o elemento de visualização, subclasse de *View*, poderá sofrer alguma ação ou interação por parte do usuário deverá ter também associada uma subclasse de *Controller*, subclasse *de MouseController* no caso da interação através do mouse e *KeyController* no caso de interação através do teclado.

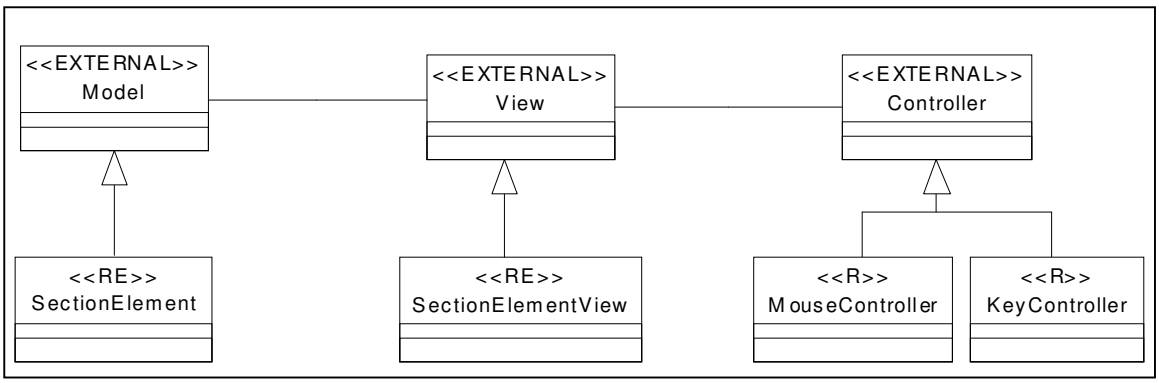


Figura 4-13Separação entre os conceitos do domínio, visualização e controle da interação do usuário

Toda classe descendente de *Model* herdará um atributo que é uma lista de observadores, um método para adicionar um observador na lista – *register()*, outro para remover um observador na lista – *deregister()* e um terceiro para avisar todos os observadores que se atualizam toda vez que o *Model* sofrer alguma modificação – *notifyObservers()*, isso pode ser visualizado na Figura 4-14. Os observadores de um *Model* serão as subclasses de *View*. A classe *View* implementa o padrão de projeto *Observer* (GAMMA, 2000). Todas as subclasses de *View* herdarão as características do padrão de projeto *Observer*, sendo associadas ao *Model* e implementando o método – *update()* - que representa a sua própria atualização quando o *Model* sofre alguma alteração e lhe comunicar.

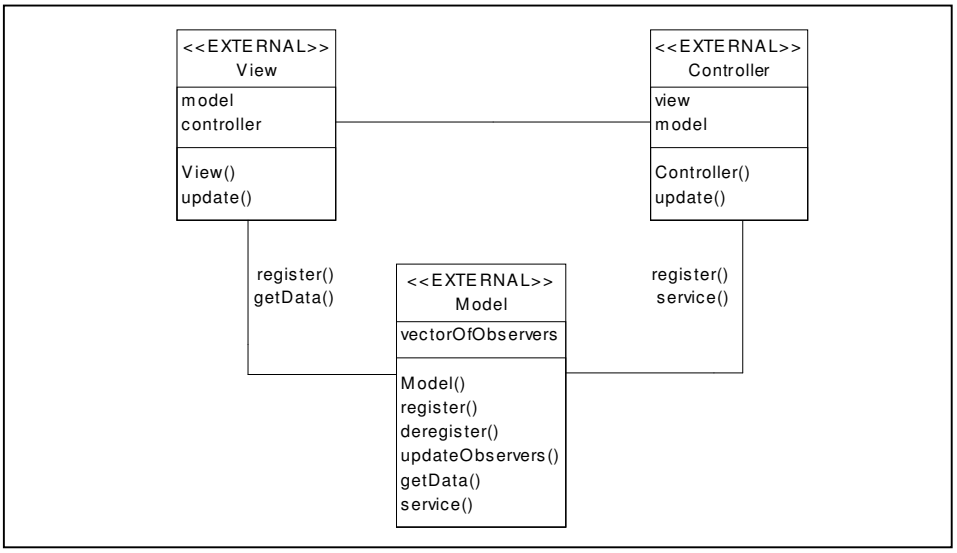


Figura 4-14Ilustração das referências entre Model, View e Controller

As classes descendentes de *MouseController* e *KeyController* herdarão uma referência para o modo de visualização (*View*) e para o modelo (*Model*), essas referências podem ser observadas na Figura 4-14. As classes descendentes de MouseController herdarão a capacidade de interceptar os eventos de mouse do usuário do modo de visualização; as subclasses de *KeyController* herdarão recursos de interceptar os eventos de teclado. O desenvolvedor apenas utilizará as classes descendentes de *Controller* para, de acordo com os eventos do mouse ou teclado sob o modo de visualização, transformar em pedidos do modelo ou do modo de visualização.

4.2.1.6 Estrutura da ferramenta de visualização

Como foi visto na seção 3.7 um dos requisitos para um documento multimídia é a estrutura de apresentação, ou seja, como a estrutura de um documento multimídia será efetivamente apresentada ao usuário.

A partir da necessidade de uma estrutura de apresentação para documentos multimídia, surgiu o requisito R.4 do framework (visto na seção 4.1.4) que é a criação de uma ferramenta que visualize ou apresente os documentos.

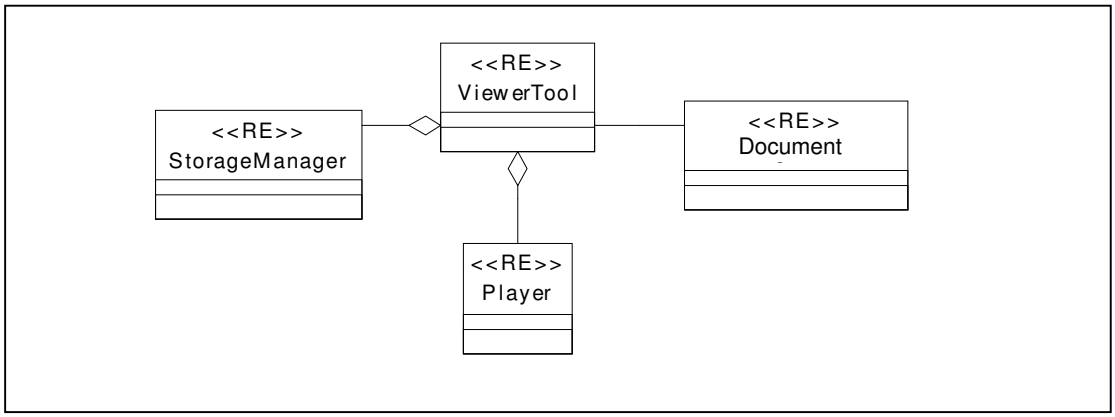


Figura 4-15Classes da ferramenta de visualização

A Figura 4-15 ilustra a classe *ViewerTool* que representa a ferramenta de visualização que agrega um gerenciador de armazenamento (*StorageManager)* e a classe responsável por executar a apresentação do documento (*Player*) e possui uma associação para a estrutura do documento multimídia (*Document*).

A referida estrutura de classes deve ser entendida como uma aplicação separada da ferramenta de autoria. A idéia é produzir a partir do framework aplicações de ferramenta de autoria que permitam a criação, edição e armazenamento de documentos por um usuário-autor. E para cada ferramenta de autoria ser capaz também de produzir a partir do HyperToolBuilder uma ferramenta de visualização correspondente que será usada pelo usuário-leitor. O autor utiliza a aplicação ferramenta de autoria para produzir um documento e o leitor utiliza uma outra aplicação chamada ferramenta de visualização para apenas visualizar o documento. Os termos autor e leitor foram convencionados na seção 4.1.2.1.

4.2.2 Modelagem Dinâmica

O objetivo da modelagem dinâmica sob a abordagem de orientação a objetos é descrever a colaboração entre objetos, bem como identificar e detalhar (para posterior implementação) os métodos das classes (SILVA, 2000). A modelagem dinâmica pode ser ilustrada de duas formas diferentes: através de diagramas de sequência e de diagramas de colaboração (FURLAN, 1998). Por mostrar de uma forma clara a interação entre os objetos em ordem temporal, optou-se por utilizar diagramas de sequência para representar a modelagem dinâmica do framework.

Será feita uma convenção nesse trabalho de que nos diagramas de seqüência utilizados, a presença de classes abstratas identificando objetos corresponde a dizer que esses

objetos serão instâncias de subclasses concretas dessas classes abstratas. Essa convenção foi baseada em SILVA (2000).

Para se chegar aos diagramas de seqüência do HyperToolBuilder, foi analisada a colaboração entre as classes de várias aplicações exemplo e abstraído um comportamento genérico para que o framework possua um protocolo de controle. Fazendo assim com que toda ferramenta de autoria desenvolvida sob o mesmo siga este protocolo.

4.2.2.1 Inicialização da ferramenta de autoria

O procedimento apresentado na Figura 4-16 representa o comportamento genérico de inicialização de qualquer ferramenta de autoria criada sob o framework HyperToolBuilder. Inicializar uma ferramenta de autoria é um procedimento comum às aplicações do domínio, porém cada aplicação tem aspectos particulares. Isso caracteriza a possibilidade de uso do metapadrão unificação (PREE, 1995), em que um método template é usado para generalizar a estrutura do algoritmo de inicialização e as particularidades de cada caso ficam sob responsabilidade de métodos hook.

Como pode ser observado na Figura 4-16, o método *initialize()* da classe abstrata *AuthoringTool* do framework HyperToolBuilder é responsável por promover a instanciação e inicialização dos objetos necessários no inicio da execução de uma ferramenta de autoria[12]. O processo genérico de inicialização de ferramentas de autoria segue o seguinte algoritmo:

- apresentar um painel de abertura - através do método *presentOpeningPanel()* o usuário do framework poderá definir se quer ou não apresentar uma tela de abertura e como será a mesma, para isso este método deverá ser definido;
- obter os parâmetros para a criação de um documento – através do método g*etDocumentParameters()* o usuário deverá implementar como será a definição do número de seções e número de elementos de seção iniciais de um documento;
- produzir uma barra de menu e uma barra de comandos específicos – de acordo com o tipo especifico de ferramenta de autoria o usuário criará menus e barra de atalho para a definição dos possíveis comandos a serem disponibilizados pela aplicação. Para isso os métodos *createMenuBar()* e *createToolBar()* deverão ser redefinidos;

[12] As duas aplicações implementadas com base no HyperToolBuilder apresentam dois exemplos de processo de inicialização de ferramentas de autoria e são abordados nas seções 5.1.1 e 5.3.1.

- produzir uma barra de ferramentas especifica – cada usuário irá definir através do método *createToolPanel()* a barra que possuirá as ferramentas que estarão disponíveis para criação e manipulação das unidades de informação em um elemento de seção.
- produzir um gerente de armazenamento específico – através do método *createStorageManager()* será possível definir gerentes de armazenamento específicos que controlem diferentes formas de armazenar um documento;
- criar um visualizar específico – da mesma forma, através do método *createPlayer()* está se oferecendo a possibilidade de criação de visualizadores de documentos específicos que permitem executar ou visualizar um documento de formas diferentes;
- criar e inicializar um documento específico – o método *createDocument()* permite a criação em tempo de projeto de diferentes estruturas de documento. Este método depende do método *getDocumentParameters()*, pois um documento só poderá ser criado e inicializado tendo a informação do número de seções e elementos de seção do mesmo;
- criar a visualização de um documento – o método *createDocumentView()* deve ser definido para criar a visualização do mesmo na tela. Através deste método o framework permite a criação de diferentes visões para documentos, sendo um aspecto flexível do mesmo.

Cada método que compõe o referido procedimento, que define o método *initialize()*, é implementado na forma de um método hook. Os métodos hook são métodos abstratos na classe AuthoringTool e que deverão ser implementados por suas subclasses.

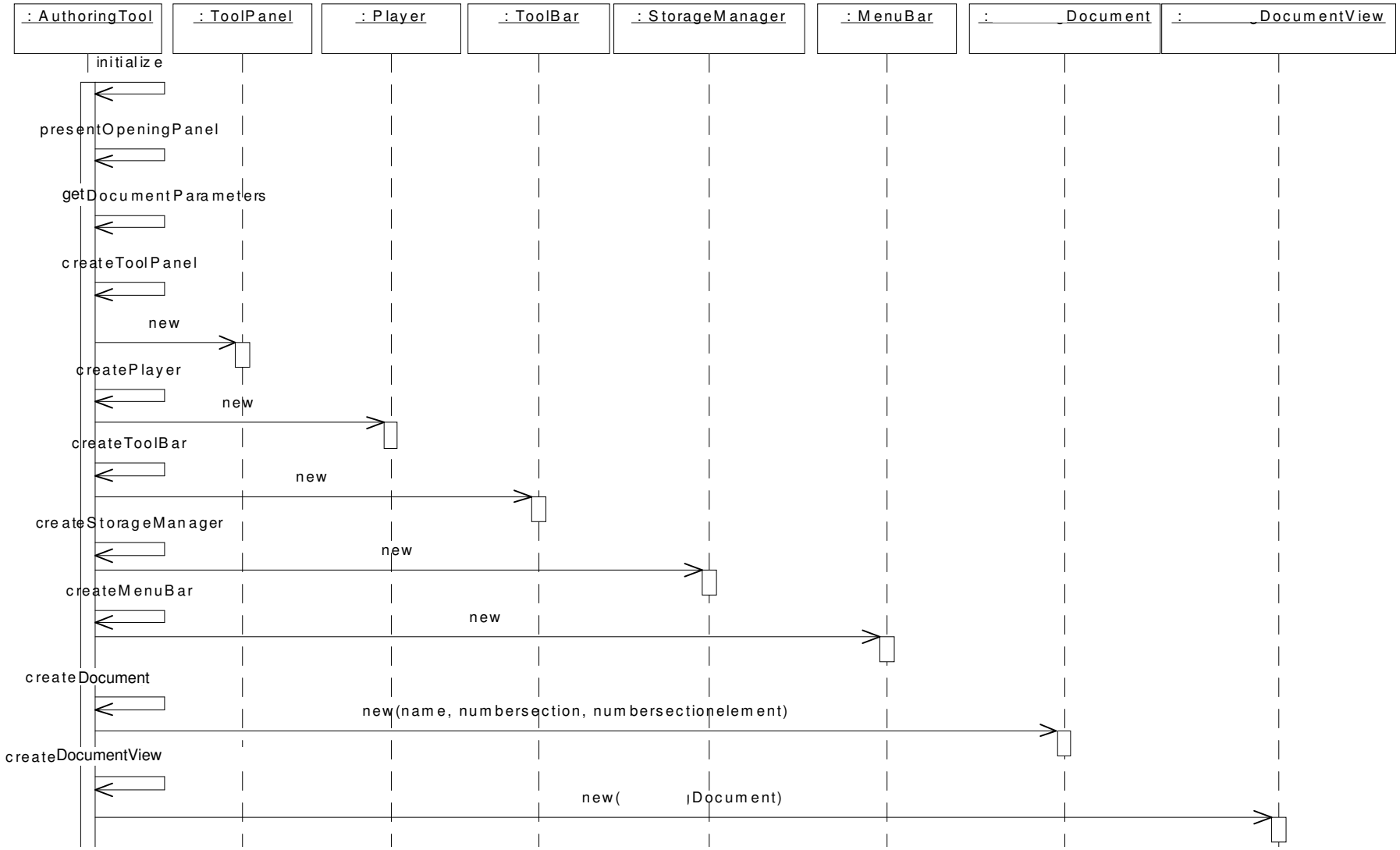


Figura 4-16Procedimento de inicialização de ferramenta de autoria

4.2.2.2 Criação do documento multimídia

O método de criação de um documento será disparado toda vez que a ferramenta de autoria for inicializada, como foi visto na seção anterior, ou quando o usuário-autor escolher o comando de criação de um novo documento através da barra de menu (*MenuBar*) ou da barra de atalho (*ToolBar*).

Como pode ser visualizado na Figura 4-16, toda vez que o usuário do framework quiser criar um documento ele deve definir os métodos *createDocument()* e *createDocumentView(),* que são métodos hook da classe abstrata AuthoringTool.

A implementação do método *createDocument*() do framework HyperToolBuilder adota o padrão de projeto *Abstract Factory* (GAMMA, 2000). Em função disso a única responsabilidade do método, que deve ser definido na subclasse concreta de *AuthoringTool*, é chamar o método construtor da classe correspondente ao documento específico da ferramenta de autoria. A Figura 4-17 ilustra a aplicação do projeto Abstract Factory no framework, produzindo um acoplamento apenas entre as subclasses de *AuthoringTool* e *Document*, e não o contrário. Assim, um documento pode ser reutilizado com diferentes ferramentas de autoria.

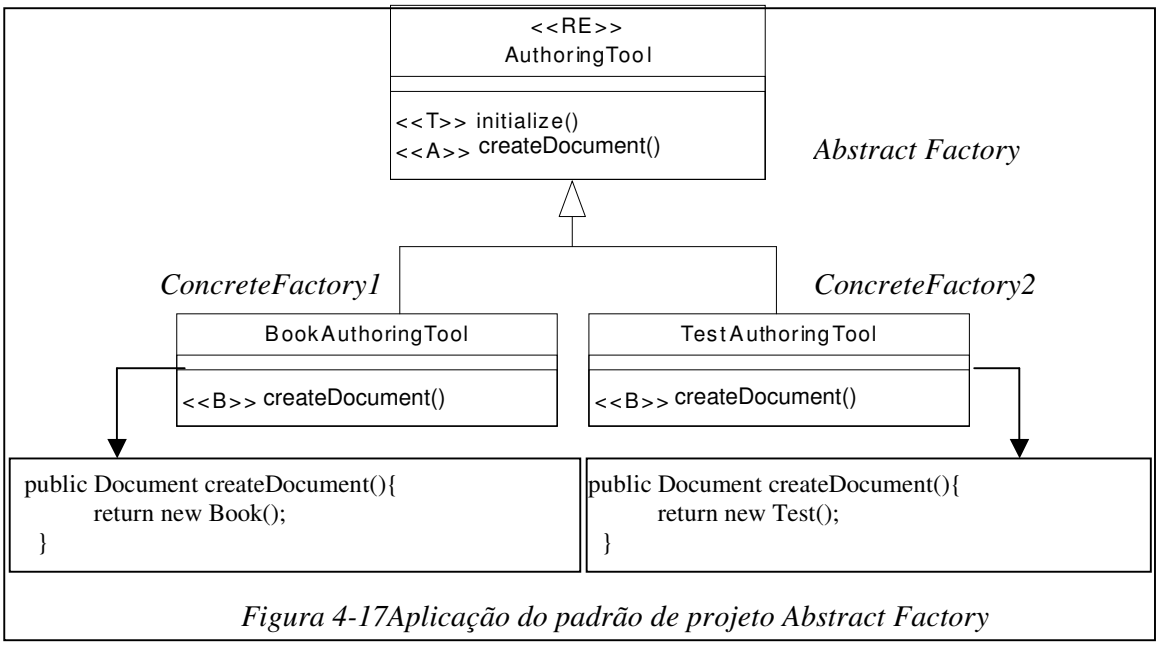


Figura 4-17Aplicação do padrão de projeto Abstract Factory

O procedimento apresentado na Figura 4-18 ilustra a instanciação de um documento pelo framework. Esse procedimento é reusado por diferentes documentos e produz a estrutura

de um documento[13] que é composta por um determinado número de seções e que, por sua vez, são compostas por um determinado número de elementos de seção[14].

A classe abstrata *Document* também utiliza o metapadrão denominado unificação para a sua inicialização, já que a mesma é um procedimento comum para qualquer documento. Na instanciação de um documento será invocado o método *template* chamado *initialize()*. Esse método possui um algoritmo genérico que invoca três métodos *hook* para flexibilizar os aspectos particulares na criação de diferentes documentos, que são: *createSection()*, *addSection()* e *setCurrentSection()*. Os métodos *hook* são invocados repetidamente, de acordo com o número de seções em que um documento será iniciado.

[13] Estruturas de documentos multimídia foram discutidas na modelagem estática do framework na seção 4.2.1.
[14] As duas aplicações implementadas com base no HyperToolBuilder possuem exemplos de processo de inicialização de documentos e são apresentados nas seções 5.1.2 e 5.3.1.

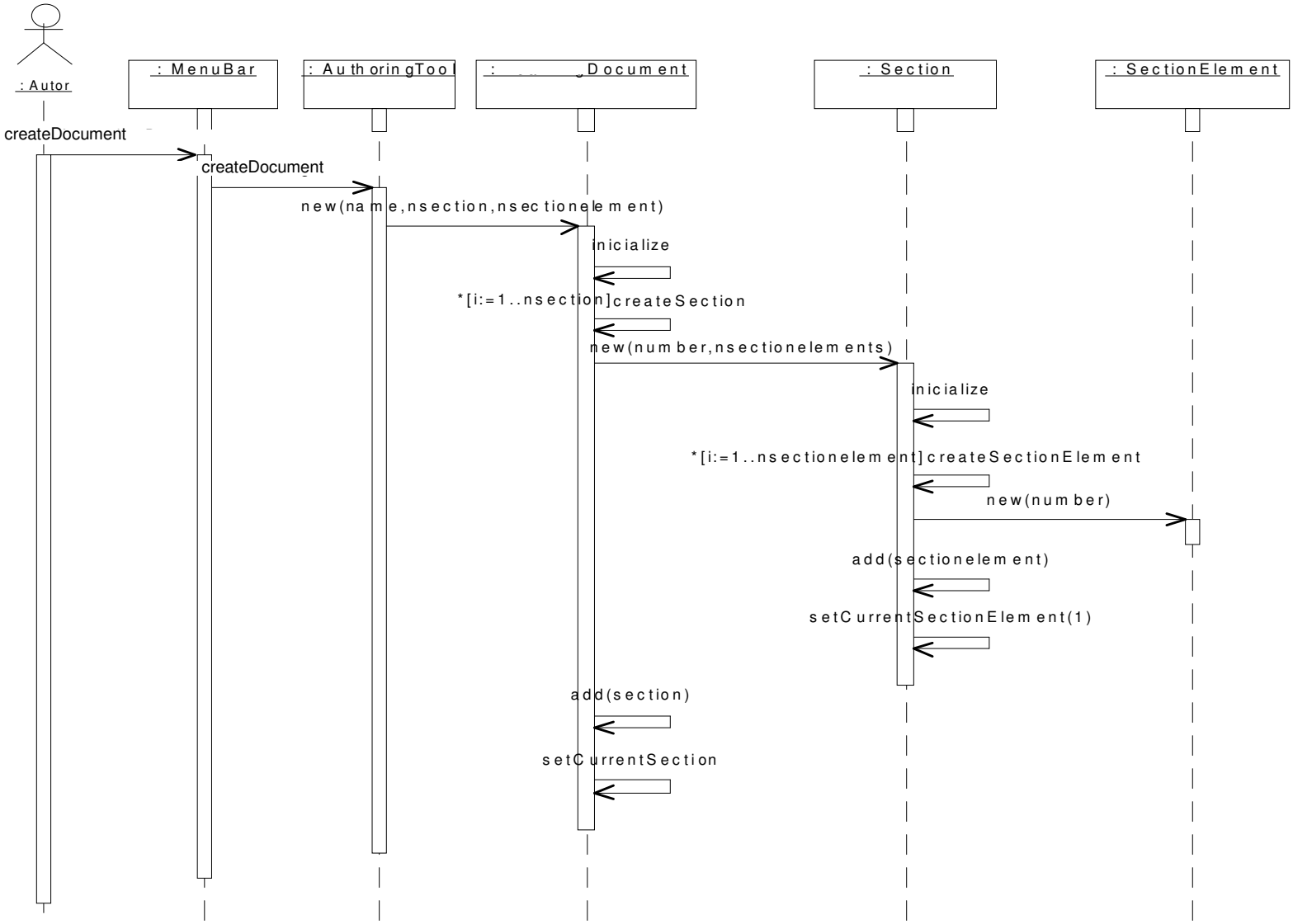


Figura 4-18Procedimento de criação de um documento pelo HyperToolBuilder

A classe *Section* também possui um algoritmo genérico para sua inicialização e por isso implementa da mesma forma que um documento o método *template initialize()*. O mesmo invoca iterativamente os métodos *createSectionElement()*, *addSectionElement()* e *setCurrentSectionElement()* de acordo com o número de elementos de seção em que se deseja iniciar cada seção.

4.2.2.3 Edição do documento multimídia

Como foi visto na seção 4.1.4.1, a atividade de edição de um documento pode consistir em um dos seguintes casos de uso: inserir, remover e definir propriedades de elementos de seção; inserir, remover e definir propriedades de seções; inserir, remover, selecionar, definir propriedades, mover e redimensionar unidades de informação.

Alguns casos de uso serão discutidos nas próximas seções, outros foram desconsiderados por brevidade.

Inserção de uma unidade de informação

Uma das principais atividades do processo de autoria de um documento multimídia é a construção dos elementos de seção através da inserção de unidades de informação nos mesmos. A inserção se dá através da seleção da respectiva ferramenta e dos seus desenhos na visão do elemento de seção através de um procedimento de "arraste e liberação" com o mouse.

As ferramentas são utilizadas nas aplicações de ferramentas de autoria para a execução de operações para a edição de cada elemento de seção de um documento. Por exemplo, uma ferramenta de criação pode ser usada para a inserção de uma unidade de informação; uma ferramenta de seleção pode ser usada para que as unidades de informações sejam selecionadas permitindo com que operações possam ser aplicadas sobre elas.

A execução das atividades de cada uma das ferramentas requer que a ferramenta de autoria seja capaz de modificar seu comportamento em tempo de execução, conforme a ferramenta selecionada. Foi utilizado o padrão de projeto State (GAMMA, 2000) no framework HyperToolBuilder para que seja possível esta alteração de comportamento. A

Figura 4-19 ilustra a hierarquia de classes que compõe o padrão State aplicado ao framework. Ilustra, também, as subclasses concretas de ferramentas previstas no

HyperToolBuilder a partir das aplicações exemplo analisadas e das duas ferramentas de autoria implementadas.

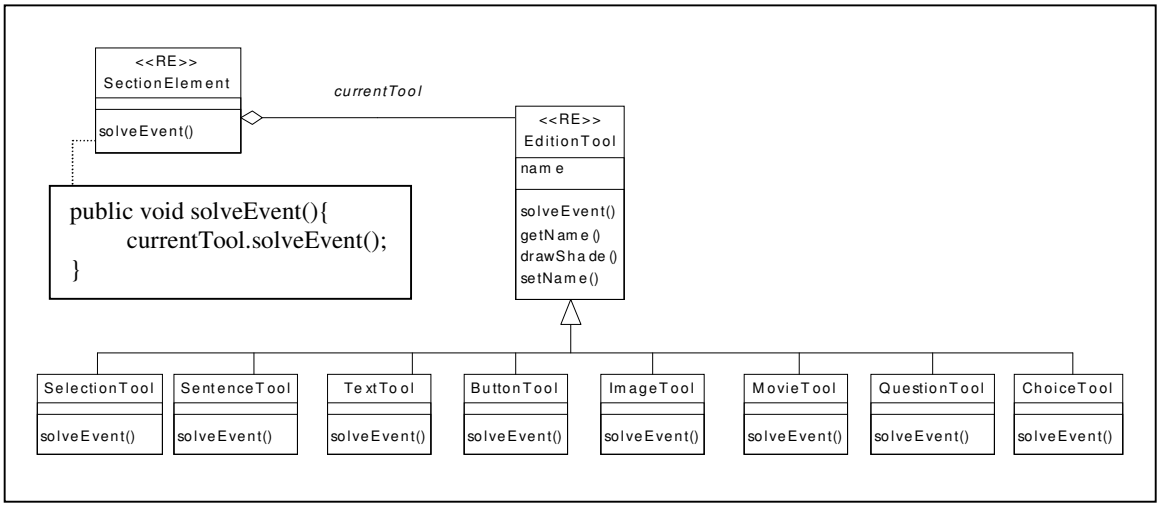


Figura 4-19Padrão de projeto State(GAMMA, 2000) aplicado para a modificação de comportamento das ferramentas

Foi criada a classe abstrata *EditionTool* para modelar o comportamento comum das várias ferramentas, um objeto de uma subclasse de *SectionElement* mantém uma referência para a ferramenta selecionada e delega as operações de tratamento de eventos para ela. O comportamento específico é obtido através das subclasses concretas de *EditionTool* que deverão redefinir os métodos de tratamento de eventos. As subclasses já criadas e previstas no HyperToolBuilder são: *SelectionTool* que trata a operação de seleção de unidades de informação; *SentenceTool, TextTool, ButtonTool, ImageTool* e *MovieTool*[15] que tratam as operações de criação das unidades de informação das mídias mais conhecidas usadas na maioria dos documentos multimídia; e ainda *QuestionTool* e *ChoiceTool*[16] que tratam operações de criação das unidades de informação do tipo questão e alternativa.

A criação de uma unidade de informação de qualquer ferramenta de autoria criada sob o framework HyperToolBuilder é sempre a criação do conceito unidade de informação seguida da produção da representação gráfica da mesma. Para isso o usuário do framework

[15] As mídias para compor qualquer documento multimídia como frase, texto, botão, imagem, filme e animação surgiram como um requisito do framework e por isso ferramentas para manipulação das mesmas foram criadas e previstas no HyperToolBuilder. O uso da implementação dessas ferramentas pode ser visto na seção 5.1.3 em que é apresentada a utilização das mesmas no protótipo HyperBook desenvolvido com base no framework.

deverá redefinir nas subclasses concretas de *EditionTool* os métodos *createInformationUnit()* e *createInformationUnitView()*, que podem ser vistos no diagrama. Esses métodos implementam o padrão *Abstract Factory* e, em função disso, o desenvolvedor de aplicações necessitará apenas chamar os métodos construtores das respectivas subclasses de *InformationUnit* e de *InformationUnitView*.

[16] As ferramentas *QuestionTool* e *ChoiceTool* surgiram a partir da implementação da ferramenta de autoria para a construção de testes – questionários de múltipla escolha – chamado HyperTest. Estas ferramentas servem para a criação de questões e alternativas, seu uso pode ser visualizado na seção 5.3.3

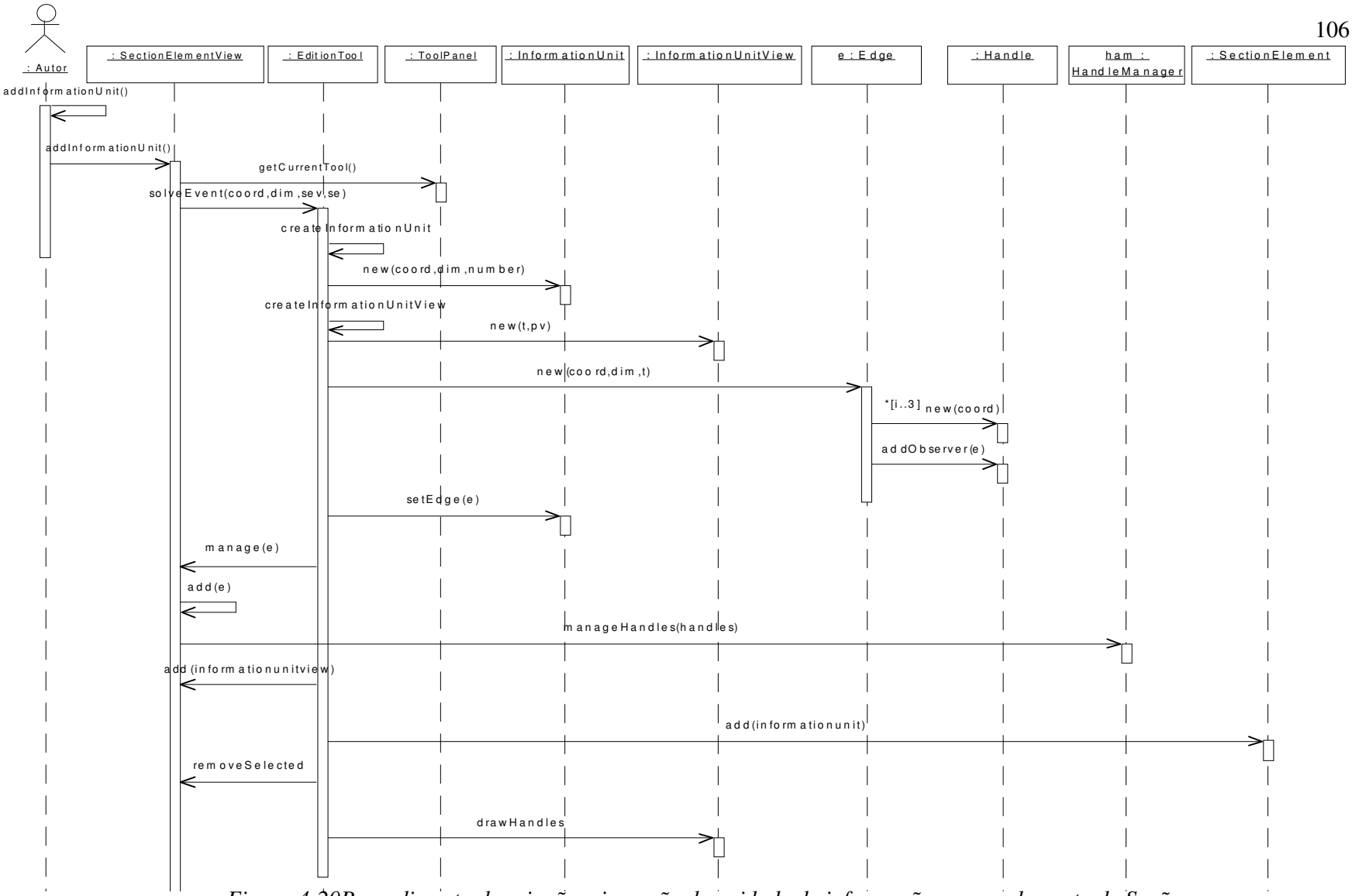


Figura 4-20Procedimento de criação e inserção de unidade de informação em um elemento de Seção

Após a instanciação de uma unidade de informação e a sua visão, outros procedimentos são necessários para a conclusão do processo de inserção da mesma em um documento. Os métodos do framework para essa atividade são:

- instanciação de uma moldura, representada pela invocação do método construtor da classe concreta *Edge* do framework, que faz por sua vez, a criação de três objetos da classe concreta *Handle*. Objetos da classes *handle* servem para mover e redimensionar a moldura e por conseqüência a unidade de informação. A classe *Edge* implementa o padrão de projeto *Observer* (GAMMA, 2000), dessa forma uma instância de *Edge* se adiciona como observador dos três *handles* – método *addObserver()* da Figura 4-20 – e assim que o usuário agir sobre os *handles*, a moldura será modificada automaticamente;
- estabelecimento de um relacionamento entre a moldura e a unidade de informação, para isso a unidade de informação manterá uma referência para a moldura, isso é feito pelo método *setEdge()*;
- invocação do método *manage()* da classe *SectionElementView* que será responsável por fazer com que esta classe tenha uma referência a todas as molduras inseridas na mesma; esse método faz também com que a classe *SectionElementView* tenha uma referência para um gerenciador de todos os *handles* inseridos na mesma, através da classe concreta *HandleManager*; essa referência é feita através da chamada, dentro do método *manage(),* do método *manageHandles()* de *HandleManager*;
- a visão da unidade de informação é adicionada na visão do elemento de seção através da invocação do método *addView()* de *SectionElementView*;
- o conceito unidade de informação é adicionado no conceito elemento de seção através da invocação do método *add()* de *SectionElement*;
- e toda vez que se encerra o processo de inserção de uma unidade de informação, as outras unidades de informação já existentes em um elemento de seção deverão estar desmarcadas e essa última deverá estar selecionada. Para

isso os métodos *removeSelected()* de *SectionElementView* e *drawHandles()* de *InformationUnit* devem ser invocados.

Remoção de unidade de informação

Quando uma unidade de informação é removida deverá deixar de existir no documento em edição. A Figura 4-21 descreve o procedimento de remoção de uma unidade de informação na edição de um documento multimídia.

A remoção de uma unidade de informação acontecerá quando o usuário selecionar uma unidade de informação e pressionar a tecla *Delete*. Uma subclasse concreta de *SectionElementController* deverá detectar o pressionamento da tecla *Delete* e invocar o método *removeInformationUnit()* da subclasse concreta de *SectionElementView*. Esse método irá verificar qual unidade de informação está selecionada e removerá sua moldura e a sua referência contida na classe *SectionElement.* E finalmente o conceito de unidade de informação será excluído. A remoção do conceito será propagada também para a sua respectiva visão, já que *InformationUnitView* é um observador de *InformationUnit*.

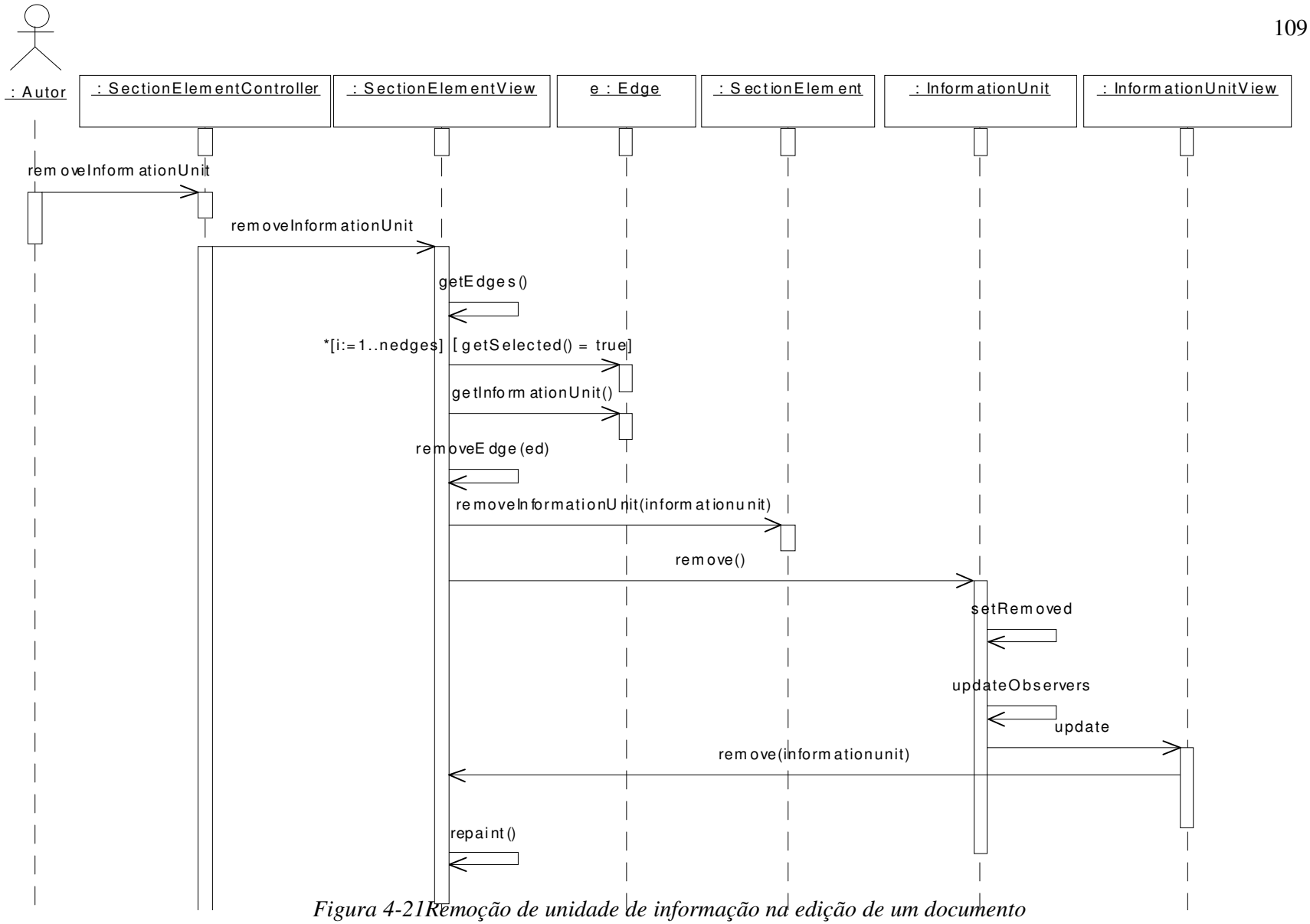


Figura 4-21Remoção de unidade de informação na edição de um documento

Definição de propriedades de unidade de informação

A partir de um duplo clique sobre a visão de uma unidade de informação, o usuário terá a possibilidade de definir as propriedades da mesma. A Figura 4-22 descreve como o framework prevê essa definição. Após o clique duplo, a subclasse concreta de *SectionElementView* irá invocar o método construtor de uma subclasse concreta da classe *PropertiesPanel*. Essa classe é responsável por construir uma janela[17] ou painel onde as propriedades como, por exemplo, conteúdo, cor, fonte, ações, serão definidas. A partir da confirmação dessas propriedades na janela, os atributos responsáveis pelas propriedades das subclasses concretas de *InformationUnit* serão atualizados ou setados. A atualização da unidade de informação será propagada para as suas visões já que as mesmas implementam o padrão de projeto *Observer* (GAMMA, 200).

[17] Exemplos de janelas de definição de propriedades de unidades de informação são encontrados nas aplicações HyperBook e HyperTest, nas seções 5.1.5 e 5.3.5.

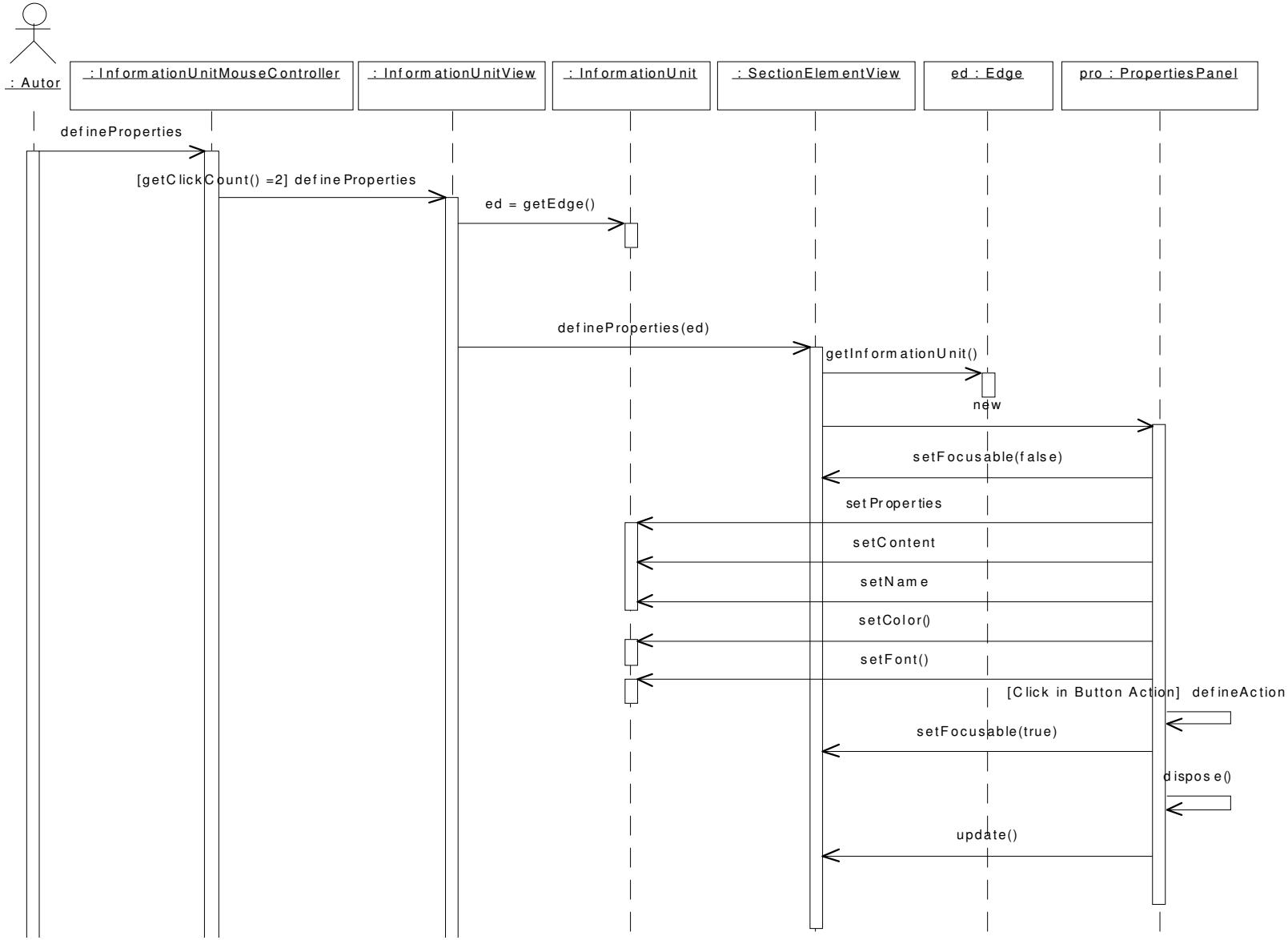


Figura 4-22Procedimento de definição de propriedades de unidade de informação

Definição da propriedade ação de unidade de informação

Se o usuário do framework optar por dar a possibilidade de definição de ações sobre as unidades de informação, o mesmo deverá redefinir o método *createButton()* da classe abstrata *PropertiesPanel* em uma subclasse concreta da mesma. Este método deverá criar um botão que a partir de um clique invocará o procedimento *defineAction()* da subclasse concreta de *PropertiesPanel*. O refinamento de *defineAction()* está descrito no diagrama da Figura 4-23.

Um exemplo de ação possível a partir do framework é executar um som caso o usuário-leitor clique sobre uma imagem. E assim várias outras ações são possíveis de serem definidas pelo usuário-autor. As ações previstas pelo HyperToolBuilder estão no Anexo III.

O procedimento *defineAction()* inicia criando uma janela para a definição das ações sobre a unidade de informação, através da instanciação de um objeto de subclasse de *ActionPanel*. Após devem ser instanciados objetos da classe *Action* para definir todas as ações disponíveis pela ferramenta de autoria que está sendo desenvolvida e associá-los a um objeto da classe *ActionList* que gerencia a lista das ações disponíveis. Uma tabela (*Table*) de duas colunas deverá ser criada e exibida na janela. Na primeira coluna representada pela classe *ActionEditor* o usuário-autor poderá selecionar uma ação a partir da lista de ações. Após a escolha de uma ação a *classe ParameterEditor,* que representa a segunda coluna da tabela, será atualizada exibindo os parâmetros disponíveis para a ação selecionada na primeira coluna. Essa atualização acontece automaticamente já que *ParameterEditor* implementa o padrão de projeto *Observer*. Um exemplo desse processo poderia ser que quando o usuário selecionar a ação "Ir para Elemento de Seção" na primeira coluna, a segunda coluna já disponibiliza como opções de parâmetros todos os números de elemento de seção disponíveis no documento. Após serem definida a ação com o seu respectivo parâmetro é invocado o método *factory()* da classe *Action* que implementa o padrão *Factory Method* (GAMMA, 2000), que instancia um objeto de uma subclasse de *Action*. Após é feita a instanciação de um objeto da classe *InformationUnitDecorator* que possui uma referência para unidade de informação e outra para a ação, fazendo assim com que a mesma "decore" a unidade de informação com a ação definida. Finalmente se faz com que *SectionElement* deixe de referenciar a unidade informação e passe a referenciar o objeto da classe *InformationUnitDecorator,* através do método *replaceInformationUnit().*

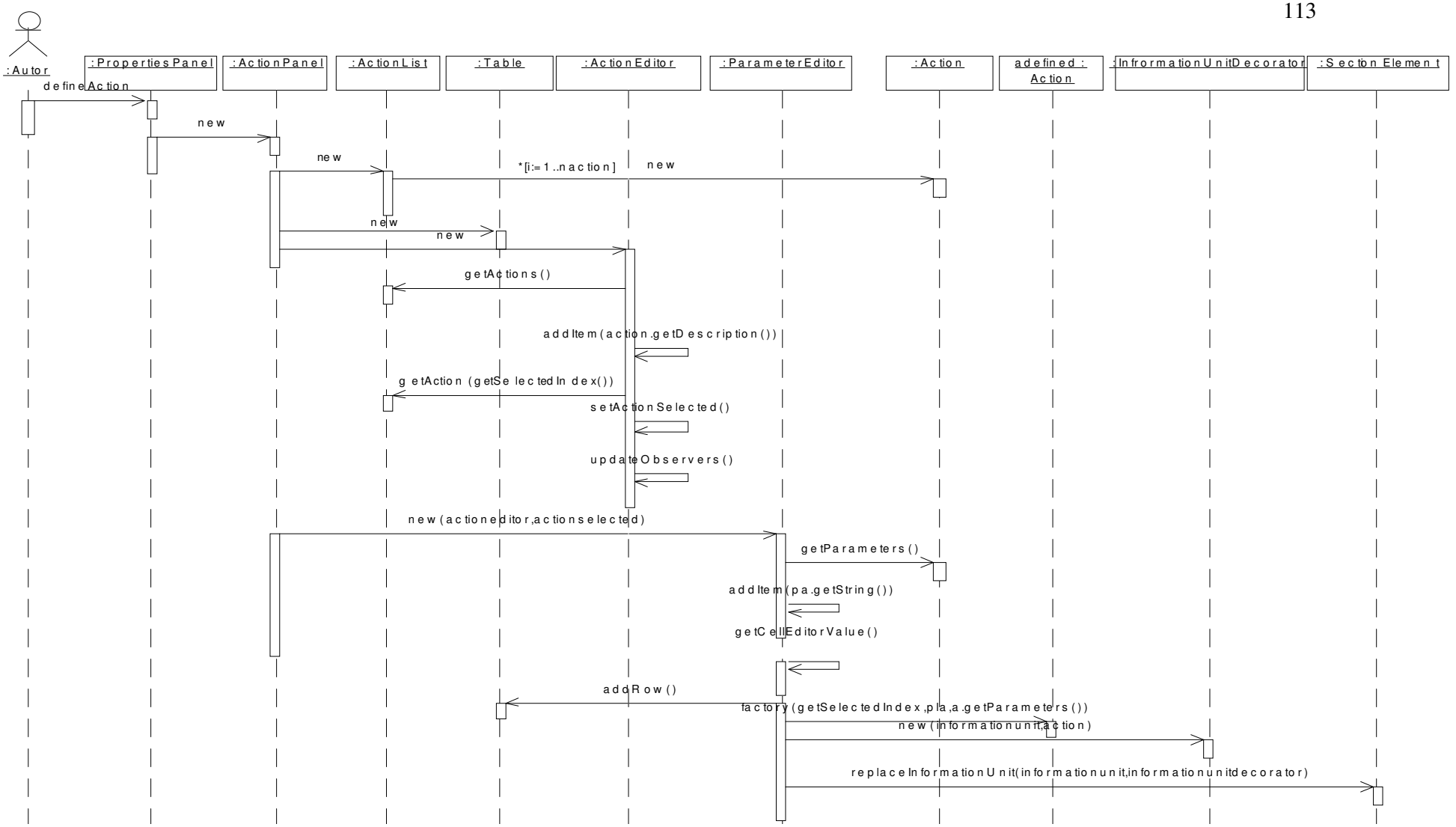


Figura 4-23Refinamento do procedimento de definição da propriedade ação de unidade de informação

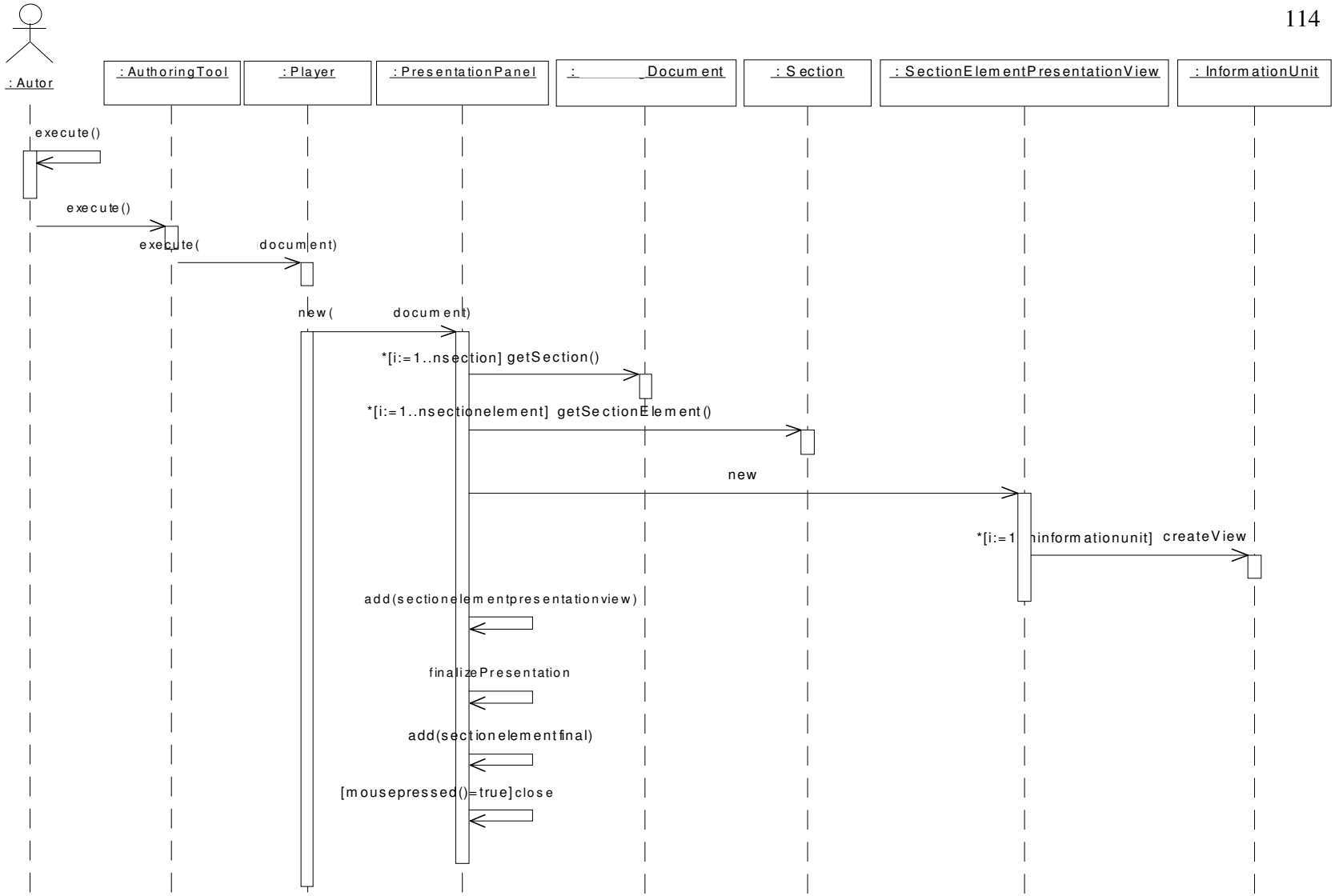


Figura 4-24Procedimento padrão de execução de um documento multimídia

É previsto pelo framework definir várias ações para a mesma unidade de informação já que *Action* possui uma subclasse que implementa o padrão de projeto *Composite*. Essa classe é chamada de *ActionComposite* e pode ser visualizada na Figura 4-12 da seção 4.2.1.4.

4.2.2.4 Execução do documento multimídia

A execução ou visualização é o produto final do processo de autoria. O usuário-autor cria um documento multimídia para posteriormente disponibilizá-lo ao usuário-leitor para que o mesmo o execute.

Uma execução padrão de um documento é disponibilizada no framework HyperToolBuilder através do método *execute()* da subclasse concreta *PlayerDefault* de *Player*, descrito na Figura 4-24. Caso o usuário do framework desejar criar uma outra forma de execução de um documento, o mesmo deverá criar outra subclasse de Player e definir o algoritmo do método abstrato *execute()*.

O procedimento de execução apresentado no diagrama da Figura 4-24 consiste na criação de um painel de apresentação em tela cheia, representado pela classe *PresentationPanel*. Essa classe obtém referências para todas as seções do documento através do método *getSection()* e, posteriormente, as referências de todos os elementos de seção através da invocação do método *getSectionElement()* da classe *Section*. Dessa forma a classe *PresentationPanel* irá apresentar seqüencialmente todos os elementos de seção e criar suas respectivas visões através da instanciação de objetos de subclasse concreta de *SectionElementView*. A seguir, a classe *PresentationPanel* irá adicionar as visões dos elementos de seção em si mesma através do método *add()*. Após a visualização do último elemento de seção será criado e apresentado um elemento de seção padrão de finalização de execução de documento.

4.3 Considerações finais

O framework HyperToolBuilder desenvolvido é do tipo caixa-cinza e é necessário, portanto, que se conheça a sua estrutura interna, conhecendo as partes flexíveis do framework para produzir novas classes específicas da aplicação em desenvolvimento e conhecendo as interfaces, relacionamentos e responsabilidades para usar as classes concretas disponíveis. Este capítulo apresentou o projeto do framework, com a finalidade de mostrar a sua estrutura interna para propiciar efetivamente o seu uso na produção de aplicações.

No desenvolvimento do HyperToolBuilder foram seguidos iterativamente os passos propostos por SILVA (2000) de generalização, especialização, aplicação de metapadrões, aplicações de padrões, projeto e aplicação de técnicas OOAD até se chegar no projeto descrito nas seções anteriores. A partir da modelagem estática e dinâmica foi implementado um protótipo do framework em linguagem de programação Java e a partir do mesmo realizou-se a implementação de duas aplicações. O desenvolvimento de aplicações sob o framework é importante, pois fornece subsídios para a identificação de possíveis anomalias no desenvolvimento, ou ainda, diagnosticar deficiências que o framework possa apresentar, subsidiando também o desenvolvedor do framework (FREIBERGER, 2002). A apresentação dessas aplicações e do seu desenvolvimento a partir do HyperToolBuilder é realizada no próximo capítulo.

5. EXPERIÊNCIA DE USO DO HYPERTOOLBUILDER

Duas aplicações de ferramenta de autoria foram desenvolvidas sob o framework HyperToolBuilder para a experiência de uso do mesmo. A seção 5.1 discorre sobre a descrição de aspectos funcionais da aplicação de ferramenta de autoria para livros eletrônicos chamada HyperBook; a seção 5.2 apresenta considerações sobre como o HyperBook foi gerado a partir do framework, quantificando as classes reutilizadas; na seção 5.3 são apresentados aspectos funcionais da ferramenta de autoria para construção de exercícios denominada HyperTest; a seção 5.4 mostra a forma como a aplicação HyperTest foi instanciada a partir do framework, quantificando as classes reutilizadas; a seção 5.5 discute aspectos relacionados ao desenvolvimento de outras possíveis ferramentas de autoria a partir do HyperToolBuilder; a seção 5.6 apresenta resultados obtidos de um teste de utilização prática da aplicação HyperBook; e finalmente a seção 5.7 apresenta as considerações finais do capítulo.

5.1 Ferramenta de autoria HYPERBOOK

O objetivo do HyperBook é dar possibilidade de criação e visualização de documentos multimídia do tipo livros eletrônicos.

5.1.1 Inicialização da ferramenta de autoria HyperBook

Ao se entrar no HyperBook é apresentada uma tela de opções para que o usuário-autor possa escolher uma forma de inicializar a ferramenta de autoria, conforme é ilustrado na Figura 5-1. O usuário deverá clicar em um dos botões e a ferramenta será iniciada, se clicar no botão "Você quer criar um livro ?", por exemplo, a mesma será iniciada para a edição de um novo livro.

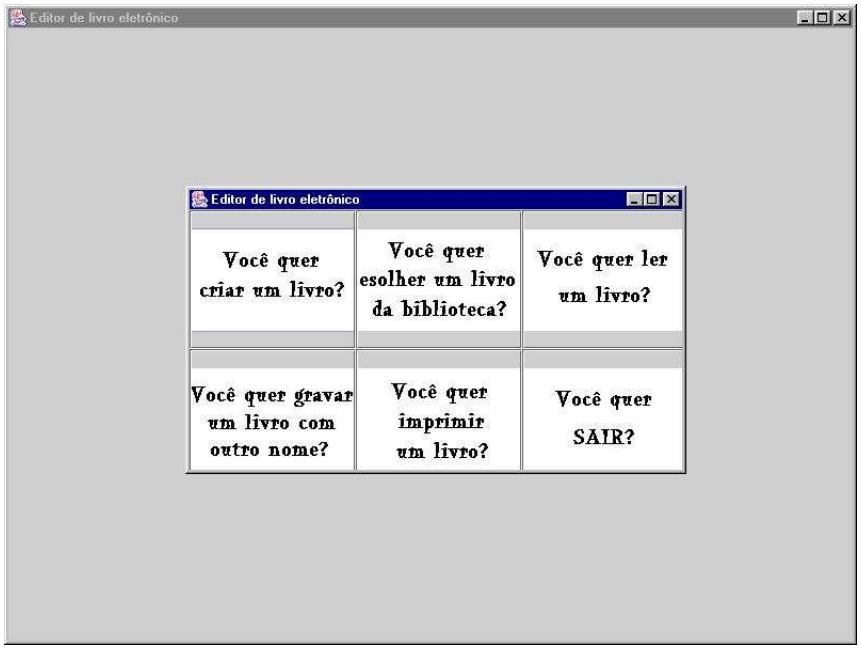


Figura 5-1Tela de inicialização do HyperBook

5.1.2 Inicialização do livro eletrônico

No caso deste protótipo, um livro eletrônico é um documento multimídia que possui uma divisão em seções, chamadas de capítulos, e os mesmos divididos em elementos de seção, chamados de páginas. Assim sendo, todo livro criado com a ferramenta de autoria HyperBook deverá ter um ou mais capítulos e cada capítulo deverá ter uma ou mais páginas. Por isso, quando o usuário optar por criar um novo livro será apresentada uma tela para a inicialização do mesmo, ou seja, uma tela em que o usuário poderá informar com quantos capítulos ele deseja que o livro seja iniciado e também com quantas páginas ele deseja iniciar cada capítulo, essa tela está apresentada na Figura 5-2. A referida tela serve apenas para criar e definir um número de capítulos e páginas iniciais do livro, como será visto posteriormente o usuário terá a possibilidade de inserir e excluir páginas e capítulos como achar conveniente, no processo de autoria do mesmo.

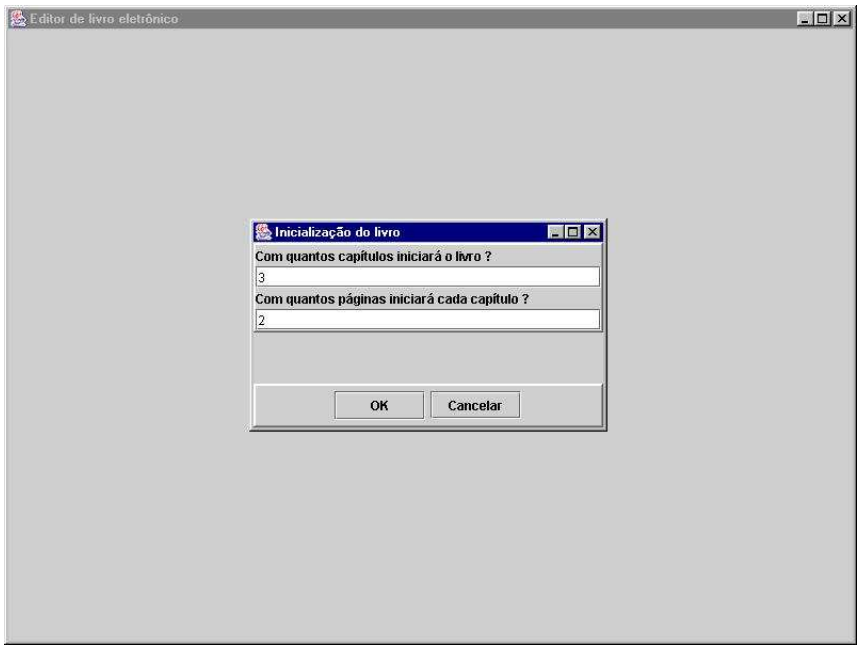


Figura 5-2Tela de inicialização de livro no HyperBook

5.1.3 Tela principal do HyperBook

Após o usuário definir o número de capítulos e páginas iniciais do livro, será apresentada a tela para a realização do processo de autoria, como exibe a Figura 5-3.

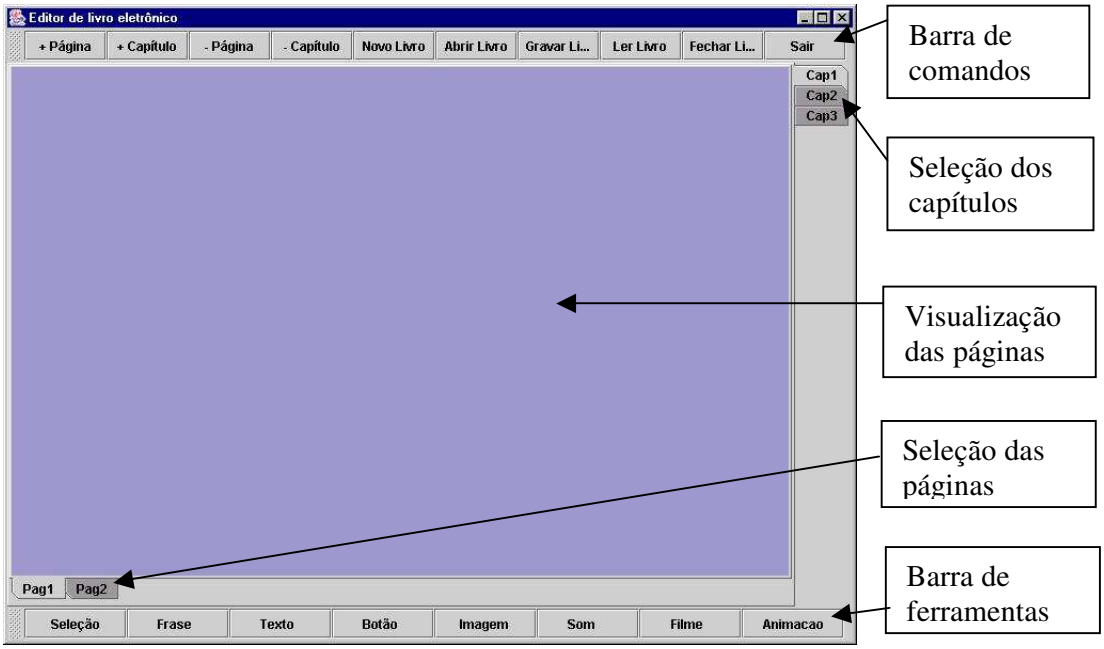


Figura 5-3Tela principal da ferramenta de autoria HyperBook

A tela principal da ferramenta de autoria é composta, na área superior, por um menu[18] e uma barra de comandos pelos quais o usuário executa os comandos como: inserir página, inserir capítulo, excluir página, excluir capítulo, criar um novo livro, abrir um livro que já está armazenado, gravar em disco um livro, ler (executar) um livro, fechar um livro e sair da ferramenta de autoria. Na área inferior, uma barra de ferramentas em que o usuário seleciona ferramentas específicas para editar o livro, como: ferramenta de seleção - usada para selecionar mídias para formatação ou exclusão, ferramenta de frase – para inserir frases com apenas uma linha no livro, ferramenta de texto – para inserir parágrafos de texto e ferramentas de botão, imagem, som, filme e animação para inserção dessas respectivas mídias.

Na área central da tela principal tem-se a visão das páginas do livro, podendo se fazer a seleção das mesmas através da escolha do respectivo capítulo nas abas à direita e a escolha da respectiva página nas abas inferiores.

5.1.4 Inserção das mídias no livro

A autoria de um livro eletrônio no HyperBook consiste basicamente em selecionar através das abas uma determinada página e após realizar basicamente duas etapas. A primeira etapa é selecionar as respectivas ferramentas e inserir as mídias na página. A Figura 5-4 exibe a primeira página do primeiro capítulo de um livro, com todas as mídias previstas[19] no HyperBook imediatamente após a sua inserção.

[18] No protótipo HyperBook se optou por não implementar os menus de comandos, implementou-se apenas uma barra de comandos por brevidade

[19] Apenas a mídia Animação não foi implementada

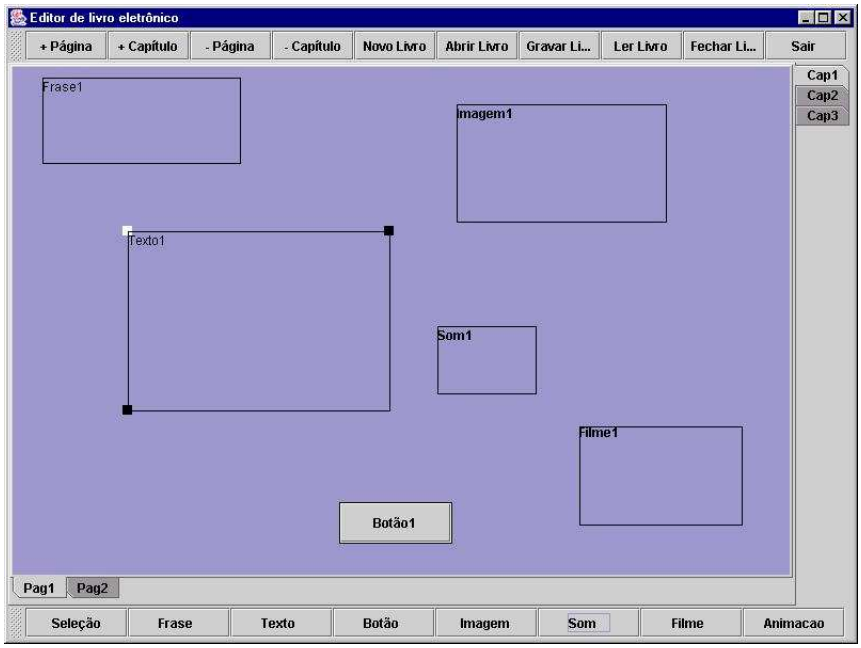


Figura 5-4Tela com as mídias após sua inserção

5.1.5 Definição das propriedades das mídias

A segunda etapa consiste em definir as propriedades das mídias através de um clique duplo sobre as mesmas. A Figura 5-5 exibe a janela de definição das propriedades da mídia texto. As principais propriedades das mídias são o conteúdo, no caso da mídia texto consiste no texto propriamente dito, a cor, tipo de letra e as possíveis ações sobre a mídia.

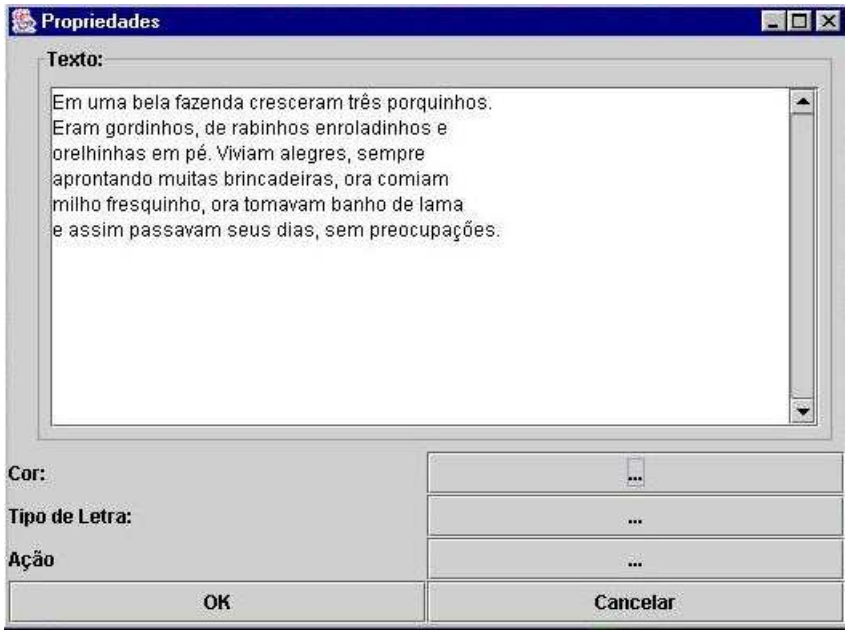


Figura 5-5Janela para definição das propriedades da mídia texto

A definição da propriedade cor é realizada através do clique no botão da cor da janela de definição das propriedades, a janela de configuração de cor de uma mídia está ilustrada na Figura 5-6.

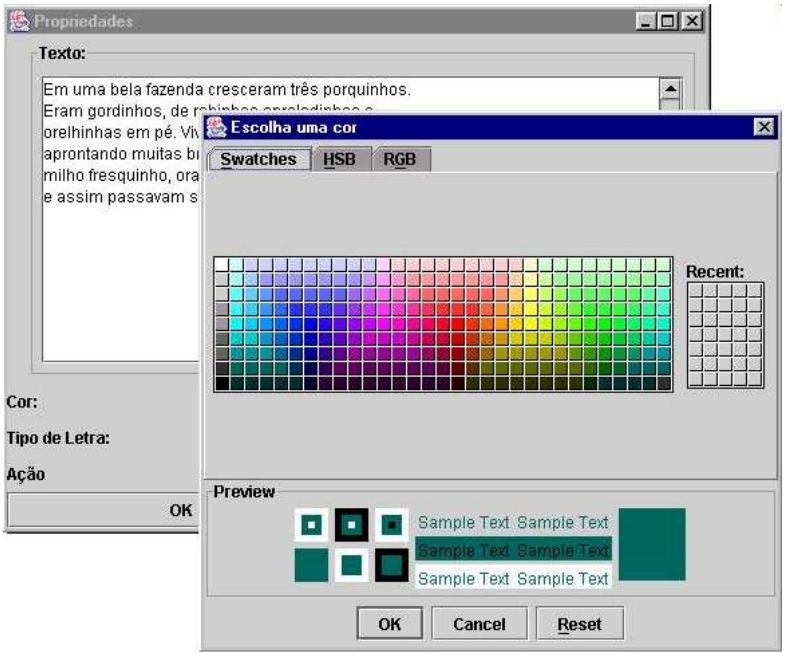


Figura 5-6Janela de configuração de cor de uma mídia

A Figura 5-7 ilustra a janela de definição de fonte de letra das mídias frase e texto. Como se pode observar na figura existe a possibilidade da definição da fonte de letra, tamanho da fonte e efeitos itálico e negrito.

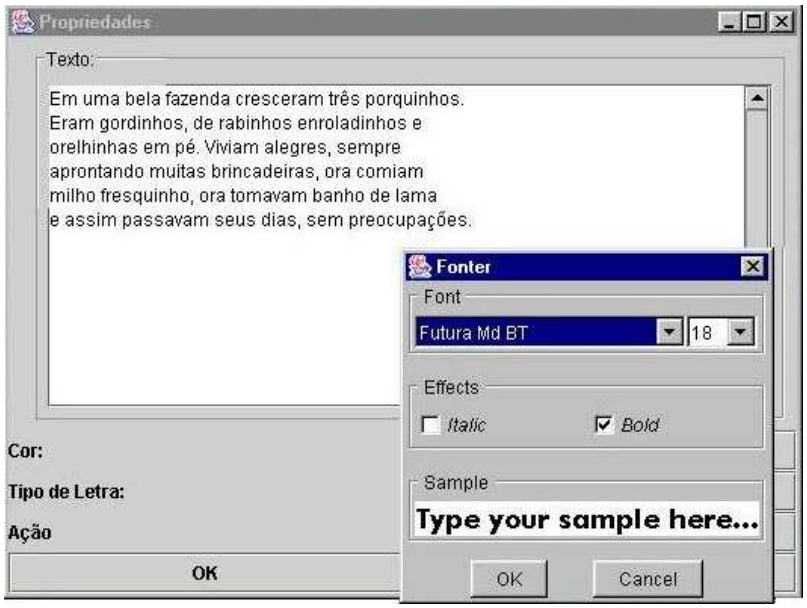


Figura 5-7Janela de configuração de fonte de letra das mídias frase e texto

Outras propriedades que podem ser definidas para qualquer mídia são as possíveis ações que devem ser executadas em caso de um clique sobre a mesma. Pode ser definida uma ou mais ações para uma mídia. No protótipo do HyperBook foram implementadas apenas as ações: Ir para página, Executar som, Executar filme e Sair do Livro, como pode ser visualizado na Figura 5-8.

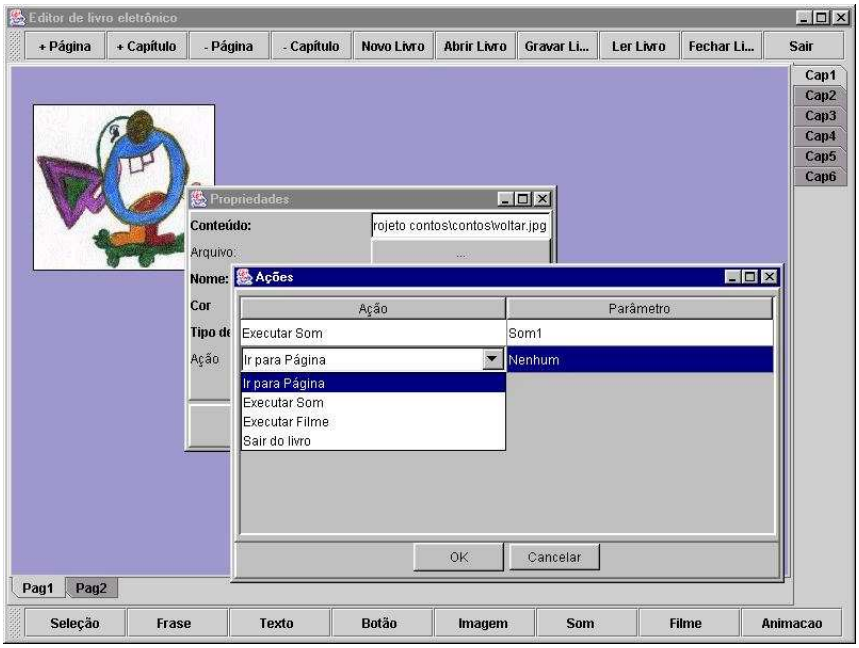


Figura 5-8Janela da definição de ações sobre uma mídia

As mídias do tipo Imagem, Som e Filme têm uma de suas propriedades, chamada conteúdo, que tem o seu valor definido através da escolha de algum arquivo de imagem, som ou filme digitalizado, previamente existente no disco do computador. Dessa forma, essas mídias não são incluídas no livro que será armazenado, existirá apenas uma ligação entre as mídias do livro e os arquivos dessas mídias. Obtém se com isso a vantagem da diminuição do tamanho do arquivo de cada livro armazenado e, por outro lado, a desvantagem da necessidade do controle manual dos arquivos externos de mídia, por parte do usuário. A ligação entre as mídias e seus respectivos arquivos externos de mídia se dá através da definição do caminho e nome dos arquivos externos de cada mídia do livro. Essa definição do caminho e nome do arquivo pode ser feita através de uma janela de escolha de arquivos, como exibe a Figura 5-9.

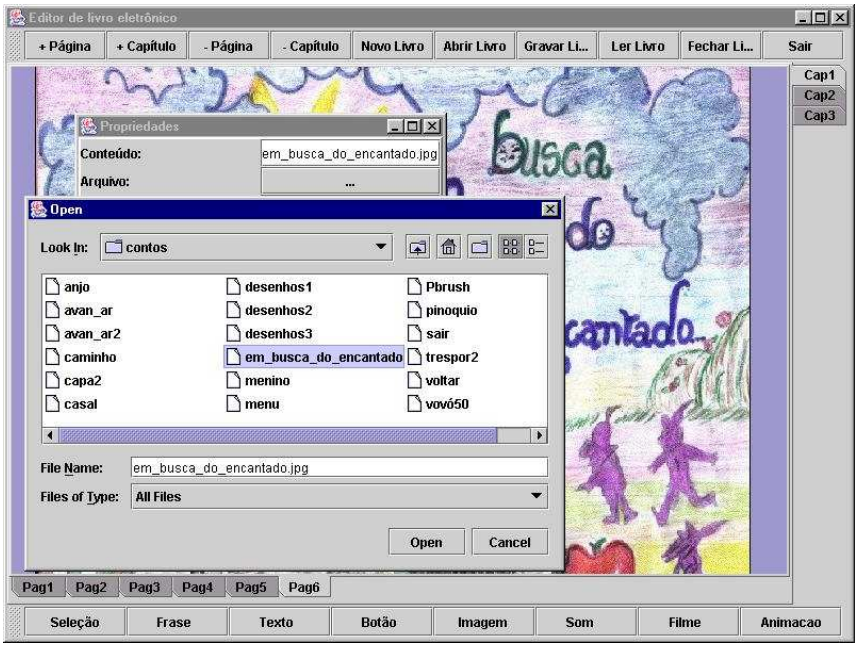


Figura 5-9Tela com a janela de escolha de arquivos de imagem

Após a definição de todas as propriedades das mídias de uma página está concluída a etapa principal do processo de autoria de uma página. A Figura 5-10 ilustra uma página após a definição das propriedades de todas as mídias previstas no HyperBook.

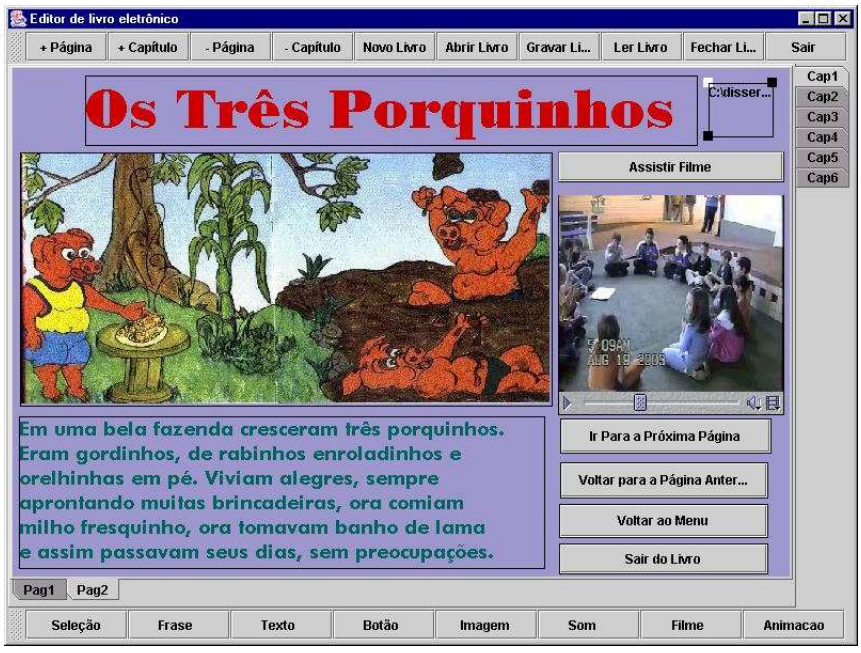


Figura 5-10Página com mídias após a definição de suas propriedades

5.1.6 Definição das propriedades das páginas

Uma outra etapa do processo de autoria de uma página consiste na definição das propriedades da própria página. No HyperBook foi implementada apenas a definição da propriedade cor, a janela da definição de cor de página pode ser visualizada na Figura 5-11.

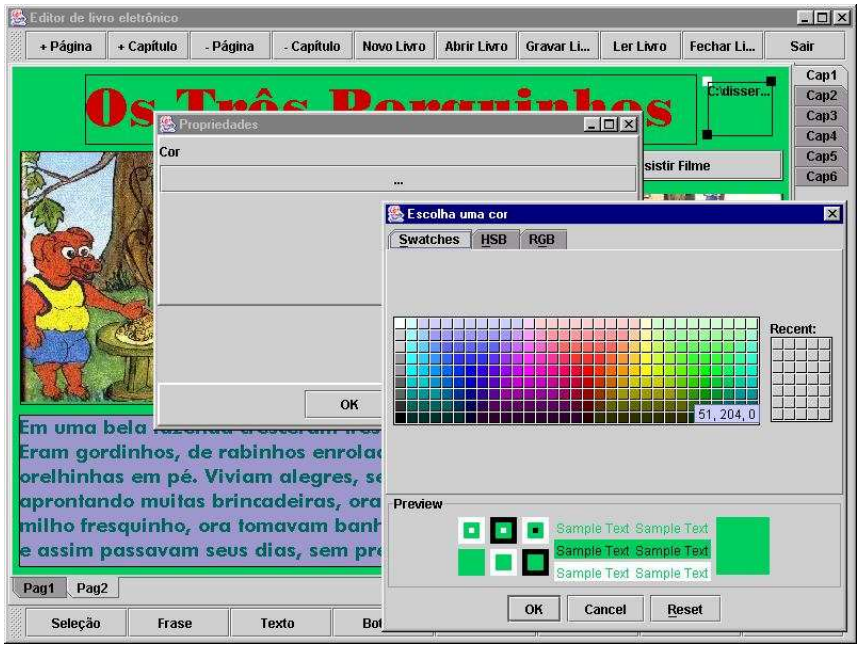


Figura 5-11Tela com a janela de definição de cor de uma página

5.1.7 Armazenar um livro eletrônico

Após a definição de cor das páginas, da inserção das mídias e definição das propriedades das mesmas, para cada página do livro conclui-se o processo de autoria. Após o livro ter sido concluído existe a possibilidade de armazenar o mesmo em disco. Para salvar o livro o usuário clica na opção "Gravar Livro" da barra de comandos e o HyperBook irá apresentar uma janela para a definição do local de armazenamento, bem como o nome do mesmo. A tela com a janela para salvar um livro pode ser visualizada na Figura 5-12.

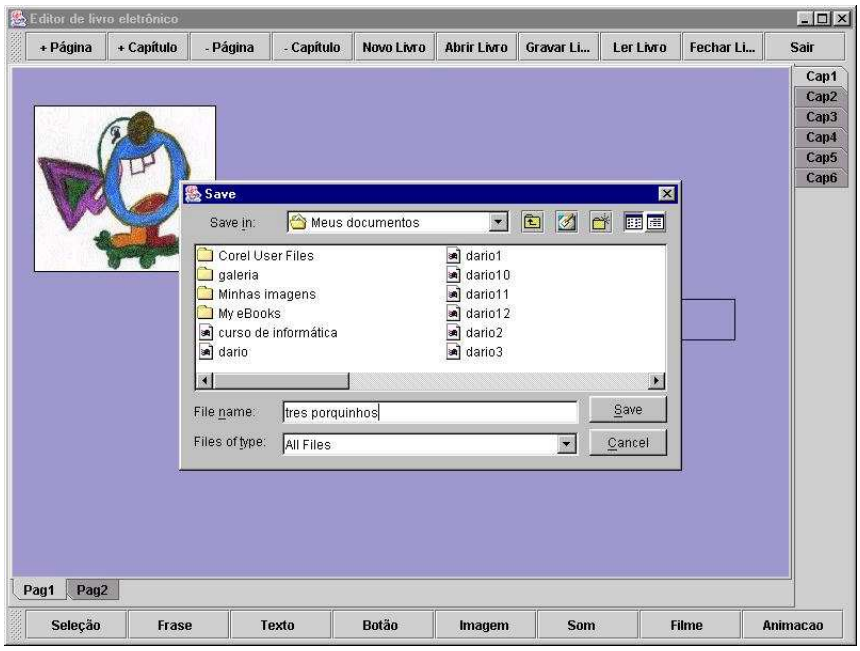


Figura 5-12Tela com a janela para salvar um livro eletrônico

5.1.8 Abrir um livro eletrônico

Após salvo, o usuário pode a qualquer momento abrir novamente o livro para visualizá-lo ou para continuar a editá-lo. Sempre que o usuário optar por abrir um livro será necessário primeiramente fechar o livro que está sendo editado. Isso acontece de forma automática caso o usuário clique no botão "Abrir Livro", da barra de comandos, ou o usuário pode optar por fechá-lo, através do botão "Fechar Livro". O processo de fechar um livro consiste em apresentar a janela para salvar o mesmo, a mesma janela da Figura 5-12, e após desaparecer a visualização do mesmo.

Tendo fechado o livro atual, será apresentada a janela para a escolha do livro eletrônico a ser aberto que está ilustrada na Figura 5-13.

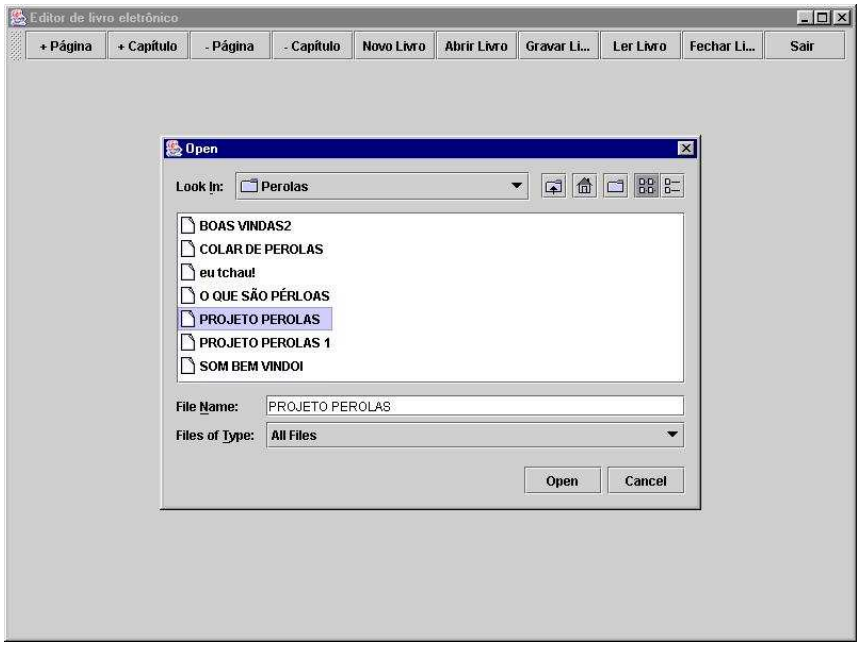


Figura 5-13Tela com a janela para abrir um livro eletrônico

5.1.9 Visualizar um livro eletrônico

Finalmente, após concluída a etapa de autoria de um livro o usuário-autor poderá visualizá-lo através da opção "Ler Livro", da barra de comandos. O modo de leitura ou visualização do livro implementado no protótipo HyperBook é realizado através da apresentação em tela cheia de todas as páginas do mesmo de forma seqüencial. A Figura 5-14 apresenta uma página no modo de visualização. Esta seqüência pode ser alterada através das ações que o autor poderá inserir nas mídias. Dessa forma, a leitura feita pelo usuário-autor pode não consistir apenas em ler as páginas seqüencialmente de uma forma passiva, mas o mesmo poderá interagir com o livro clicando sobre as mídias e, nesse momento, várias ações diferentes podem ser executadas, entre elas a ação de ir para qualquer outra página do livro.

Figura 5-14Exemplo de uma página no modo de visualização ou leitura do livro

O usuário poderá a qualquer momento optar por sair do modo de visualização através da escolha no *menu popup* da opção "Sair da execução do livro". Es te menu será exibido toda a vez que o usuário clicar com o botão direito do mouse, o qual pode ser visualizado na Figura 5-14.

5.2 Considerações sobre o desenvolvimento do HYPERBOOK

Após descritas as funcionalidades da ferramenta de autoria HyperBook cabe ressaltar a forma que a mesma foi desenvolvida baseada no framework HyperToolBuilder.

A Figura 5-15 ilustra as especializações das classes do framework HyperToolBuilder para o desenvolvimento da ferramenta de autoria HyperBook. Para a criação da ferramenta de autoria HyperBook foi criada uma subclasse concreta *BookAuthoringTool,* da classe abstrata *AuthoringTool* do framework. Dessa forma a aplicação herda automaticamente os relacionamentos de suas superclasses do framework. Herda o relacionamento de *AuthoringTool* com *StorageManager*, reutilizando dessa forma o procolo de controle de armazenamento de documentos. Além disso, como o framework já possui uma subclasse concreta de *StorageManager* chamada *FileStorageManager,* ele já disponibiliza através dessa classe toda a funcionalidade para armazenamento de documentos em arquivo. Ao desenvolver a aplicação, o usuário poderá decidir entre apenas instanciar a classe do framework *FileStorageManager* caso deseja armazenar documentos em arquivos, ou criar outras

subclasses de *StorageManager* com as funcionalidades para outras formas de armazenamento, em banco de dados ou em formato XML, por exemplo.

Através da especialização da classe *AuthoringTool* a aplicação herda também o relacionamento com a classe *Player*, responsável por controlar a execução ou visualização de um documento. O framework já possui uma subclasse concreta de *Player* chamada *PlayerDefault,* que apresenta seqüencialmente todas os elementos de seção de um documento em tela cheia. Porém, *Player* também consiste em um ponto flexível do framework já que permite a criação de outras subclasses com outras implementações de execução de documentos.

Para a criação da aplicação HyperBook foram reutilizadas as duas classes concretas: *FileStorageManager* e *PlayerDefault,* herdando do framework toda a funcionalidade de armazenamento em arquivo e execução de livro.

Foi especializada a classe *ToolBar,* criando a subclasse concreta *BookToolBar,* para implementar uma barra de comandos ou barra de atalho (optou-se por utilizar barra de atalho ao invés de barra de menu por brevidade) que disponibiliza os possíveis comandos a serem executados pelo usuário-autor.

E, finalmente, foi especializada a classe *ToolPanel* do framework criando a classe *BookToolPanel* responsável por agregar as subclasses concretas de *EditionTool* já disponíveis no framework HyperToolBuilder. Instâncias de *SelectionTool, SentenceTool, TextTool, ButtonTool, ImageTool* e *MovieTool* foram agregadas à uma instância de *BookToolPanel* para compor a barra com as ferramentas da aplicação.

A Figura 5-16 ilustra as especializações das classes do framework HyperToolBuilder para a composição da estrutura de um documento (livro) manipulado pela ferramenta de autoria HyperBook.

Através da Figura 5-16 pode-se observar o grande número de classes e relacionamentos que são reutilizados do framework para a criação de um documento do tipo livro. Livro é o documento que será manipulado pela ferramenta de autoria HyperBook.

Para criar um livro[20] foi especializada a classe abstrata do framework *Presentation* criando a classe *Book* que é um tipo de documento multimídia. A classe *Section* criando a subclasse concreta *Chapter* representando os capítulos do livro. E a classe *SectionElement* criando a subclasse concreta *Page* que representa as páginas de um capítulo. Como a classe *Book* é subclasse de *Document* ela herda o relacionamento com as superclasses *Section* e *SectionElement*.

Todas as unidades de informação (*Sentence, Text, Button, Image, Sound, Movie*) previstas para serem inseridas em um livro estão previstas no framework, bem como sua utilização. A instanciação das subclasses concretas de *StaticMedia* e *DynamicMedia* acontece em tempo de execução da aplicação, toda a vez que o usuário-autor seleciona a ferramenta correspondente a unidade de informação, a mesma fabrica (instancia) um objeto dessas classes.

O protocolo de controle do relacionamento entre as unidades de informação e as possíveis ações sobre as mesmas também é herdado do framework. Apenas se fez necessário criar subclasses concretas da classe *Action* para disponibilizar na aplicação as ações específicas da mesma. Foram criadas apenas três ações por brevidade através das seguintes subclasses: *ExecuteSound, ExecuteMovie* e *GoToPage*.

Através da Figura 5-15 e Figura 5-16 pode-se perceber a substancial quantidade de reuso obtido para o desenvolvimento do HyperBook baseado no framework HyperToolBuilder. A partir da análise dos diagramas das figuras observa-se que seria necessário a criação de aproximadamente vinte e nove classes para o desenvolvimento da aplicação,[21] sem a utilização do framework. Com a utilização do framework foram reusadas 9 classes do mesmo através de herança de classes abstratas e 20 classes foram reusadas por composição, através da instanciação de classes concretas do framework.

---

[20] O conceito de livro foi convencionado para a criação da aplicação HyperBook como sendo um documento multimídia dividido em capítulos e estes divididos em páginas.

[21] Nem todas as classes do framework que aparecem no diagrama são reutilizadas para a ferramenta de autoria HyperBook. Além disso, a contagem realizada inclui apenas as classes do domínio do problema, não incluindo classes de Visão e Controle do modelo MVC

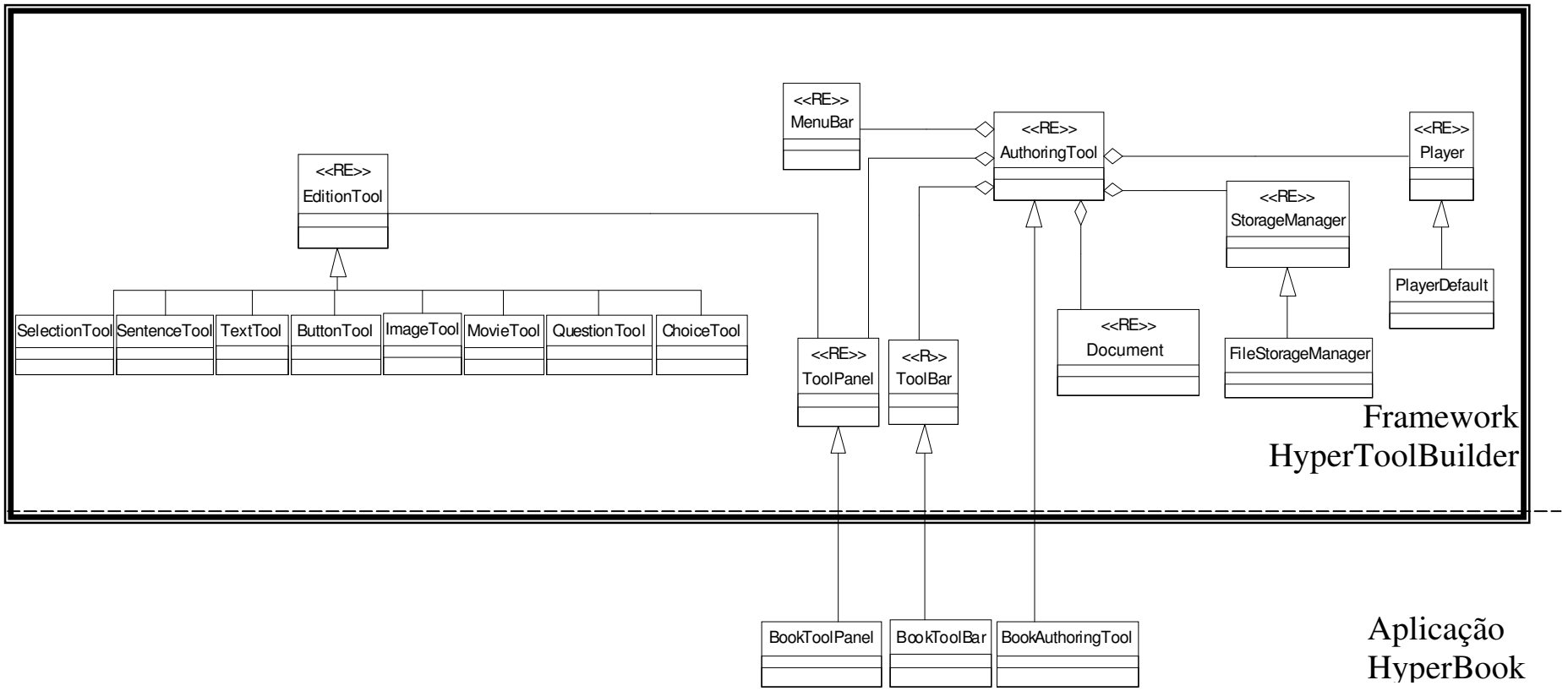


Figura 5-15 Modelo de Objetos da ferramenta de autoria HyperBook, desenvolvida sob o framework HyperToolBuilder

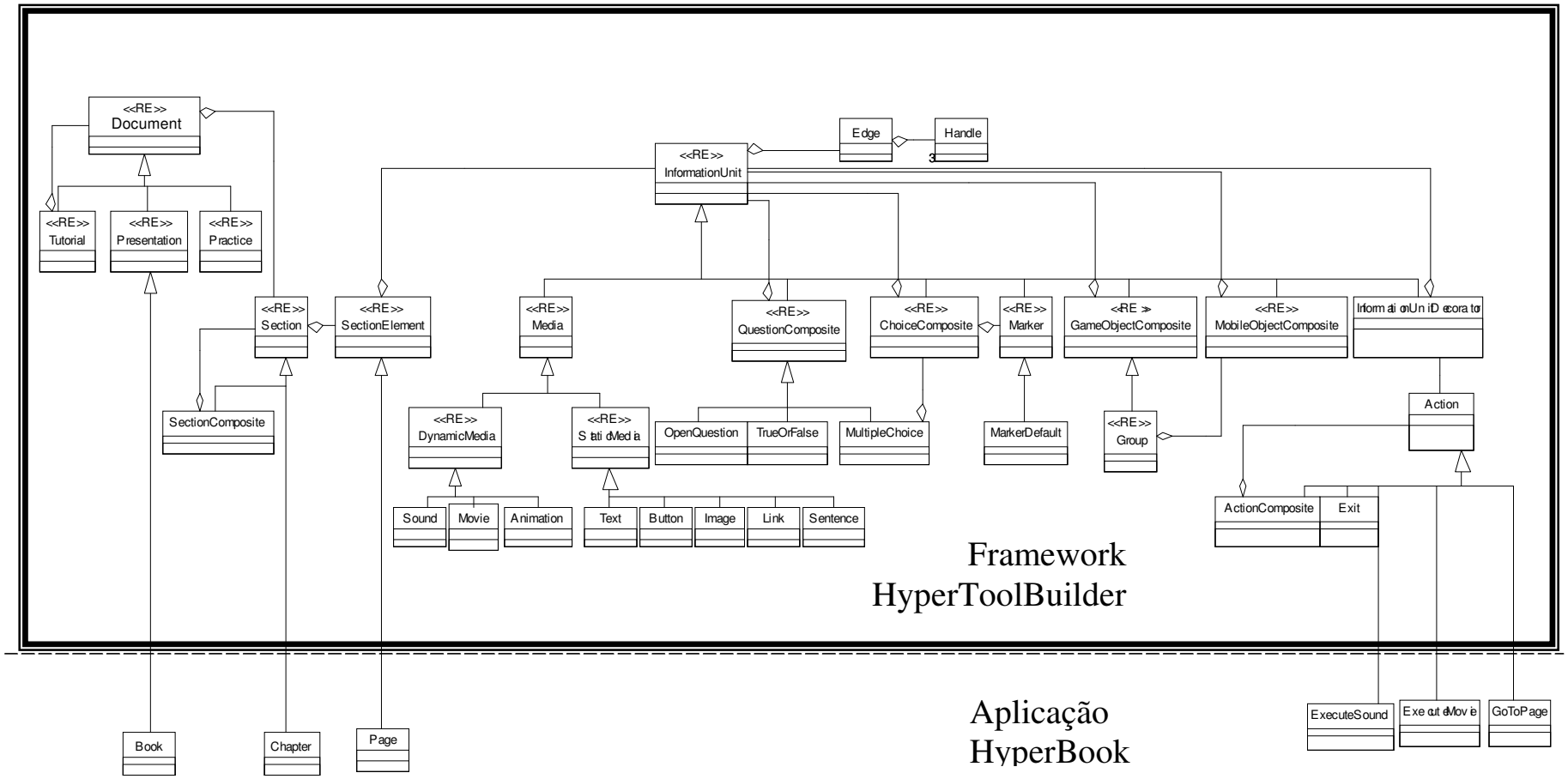


Figura 5-16Modelo de Objetos do documento(livro) da ferramenta de autoria HyperBook, desenvolvida sob o framework HyperToolBuilder

5.3 Descrição da ferramenta de autoria HYPERTEST

Essa seção apresenta aspectos relacionados à análise, projeto e implementação de uma ferramenta de autoria para atividades de exercício e prática chamada HyperTest. O protótipo apenas possibilita a criação de exercícios do tipo questões de múltipla escolha, mas o framework HyperToolBuilder prevê a criação de outros exercícios. Muitas características dessa aplicação coincidem com as da aplicação vista na seção anterior, como: possibilidade de inserção das mídias de frase, texto, imagem, botão, som e filme, a definição das propriedades das mídias e os recursos de salvar e abrir arquivos de testes. Esta seção visa elucidar aspectos da aplicação HyperTest que diferem da aplicação HyperBook vista na seção 5.1.

5.3.1 Inicialização da ferramenta de autoria HyperTest

Toda a vez que a ferramenta de autoria HyperTest é inicializada, a mesma cria automaticamente um novo teste e o apresenta na tela. Processo diferente do HyperBook que possibilitava ao usuário escolher entre as várias opções de inicialização através de um painel e, após, possibilitava também ao usuário a definição do número de capítulos e de páginas iniciais do livro, como foi visto nas seções 5.1.1 e 5.1.2, respectivamente. Através dessa comparação pode se perceber que o framework HyperToolBuilder prevê a possibilidade de diferentes inicializações das ferramentas de autoria desenvolvidas sob o mesmo.

5.3.2 Tela principal do HyperTest

A *Figura 5-17* apresenta a tela principal da ferramenta HyperTest imediatamente após sua inicialização.

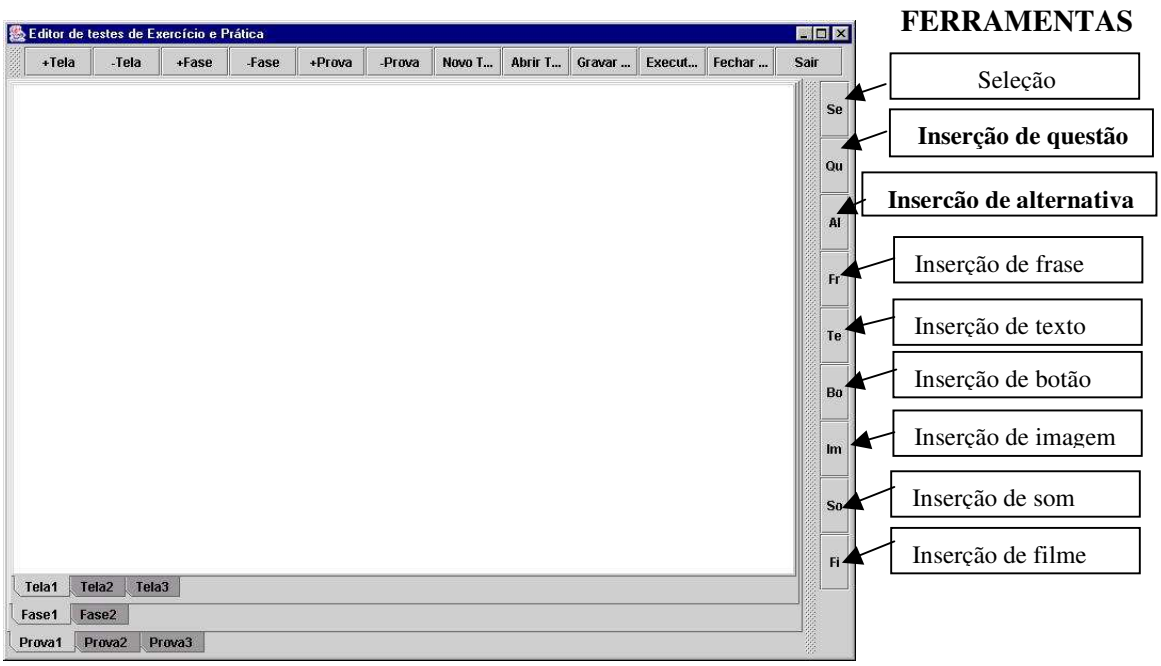


Figura 5-17Tela principal da ferramenta de autoria HyperTest

O documento multimídia produzido sob a ferramenta de autoria HyperTest difere de um livro, pois ao contrário do mesmo que possui duas dimensões – capítulo e página – um teste possui três dimensões – prova, fase e tela. Um teste produzido a partir do HyperTest é composto por uma ou mais provas, que por sua vez possuem uma ou mais fases e estas uma ou mais telas. Toda vez que a ferramenta HyperTest é inicializada é criado um teste incialmente com três provas, cada prova com duas fases e cada fase com três telas.

A tela principal do HyperTest possui na região superior a barra de comandos que difere da do HyperBook, pois possui seis botões para editar as dimensões de um teste: inserir e remover telas, inserir e remover fases e inserir e remover provas.

Na região direita da tela principal encontra-se a barra de ferramentas que possui duas ferramentas diferentes do HyperBook: a ferramenta questão e a ferramenta alternativa. Na região central tem-se a visualização do documento teste, na parte superior visualizam-se as páginas individualmente e na parte inferior é possível navegar pelas provas, fases e telas através das respectivas abas.

5.3.3 Inserção de questões e alternativas no teste

O primeiro passo da atividade de autoria no HyperTest consiste basicamente em inserir questões e suas alternativas. Para isso existem as ferramentas para inserção de questão

e para inserção de alternativas na barra de ferramentas. A *Figura 5-18* apresenta a primeira tela da primeira fase da "prova1" de um teste, imediatamente após a inserção de três questões e suas alternativas.

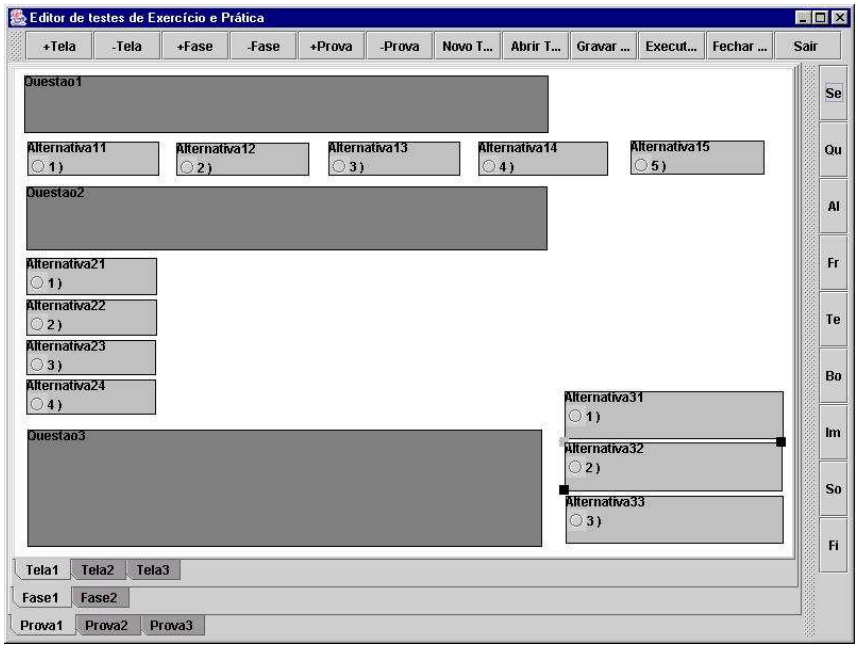


Figura 5-18Tela exemplo após a inserção de três questões e suas alternativas

As questões e suas alternativas podem ser dispostas livremente de acordo com a necessidade do usuário-autor, como pode ser visualizado na *Figura 5-18*. Além disso, o autor tem liberdade de definir o número de questões por tela e o número de alternativas por questão em um teste.

Optou-se na implementação do HyperTest de fazer o relacionamento entre uma questão e suas respectivas alternativas através da ordem de inserção das mesmas. Subentende-se que o usuário irá inserir uma questão e logo após as suas respectivas alternativas, e assim que ele inserir outra questão, subentende-se que as próximas alternativas inseridas serão dessa questão e assim sucessivamente. Essa foi uma abordagem adotada no HyperTest para a solução desse problema, mas várias outras decisões de projeto diferentes poderiam ser tomadas para a resolução do mesmo, como por exemplo, a possibilidade do autor, a qualquer momento, definir uma propriedade que diga a qual questão uma detrminada alternativa pertence.

5.3.4 Inserção das mídias nas questões e alternativas

Após as questões e alternativas terem sido criadas, as mesmas deverão ser construídas através da inserção das mídias básicas como frase, texto, botão, imagem, som e filme e a definição de suas propriedades. Para isso, a barra de ferramentas possui também as ferramentas de mídias respectivas. As questões e alternativas são definidas através da inserção e definição das propriedades de qualquer uma das mídias citadas anteriormente e, além disso, o autor terá a liberdade de definir quantas mídias deseja inserir em cada questão ou alternativa. A *Figura 5-19* ilustra um exemplo de uma tela de um teste com as questões e alternativas após a inserção e definição das propriedades de suas mídias.

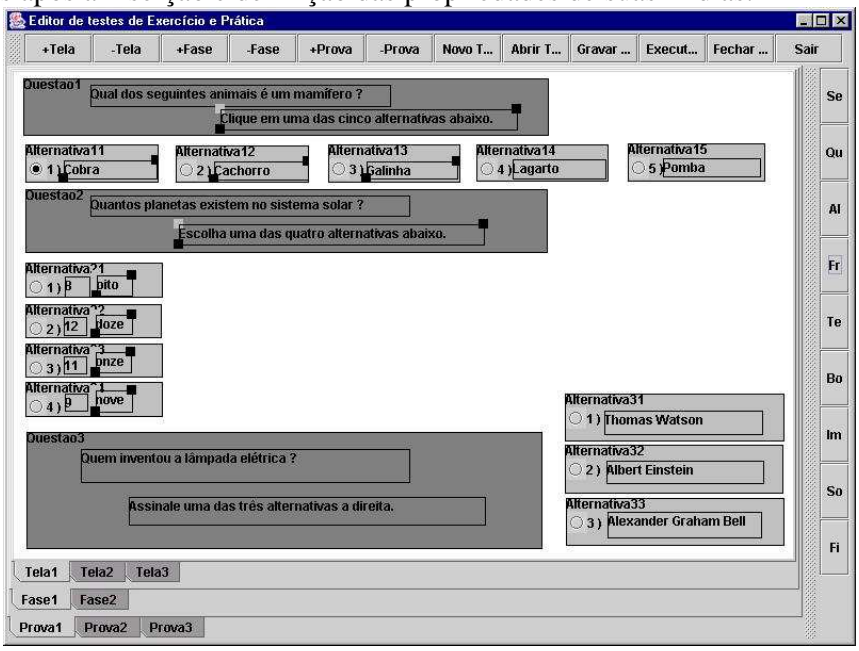


Figura 5-19Tela após a inserção e definição de propriedades das mídias

5.3.5 Definição das propriedades das questões e das alternativas

Após terem sido inseridas e definidas as propriedades das mídias das questões e das alternativas, a ferramenta HyperTest possibilita ao usuário-autor definir as propriedades das próprias questões e alternativas. Como pode ser observado na Figura 5-20, as duas propriedades a serem definidas para as questões são um número para determinar o peso de cada questão e se a questão poderá ou não ter várias alternativas corretas. A propriedade peso possibilita o autor valorizar mais determinadas questões em relação a outras. A propriedade chamada "várias alternat ivas corretas" configura a ferramenta, em caso de estar selecionada, a possibilitar o usuário-leitor de selecionar mais de uma alternativa, e em caso contrário, possibilitar a seleção de apenas uma única alternativa.

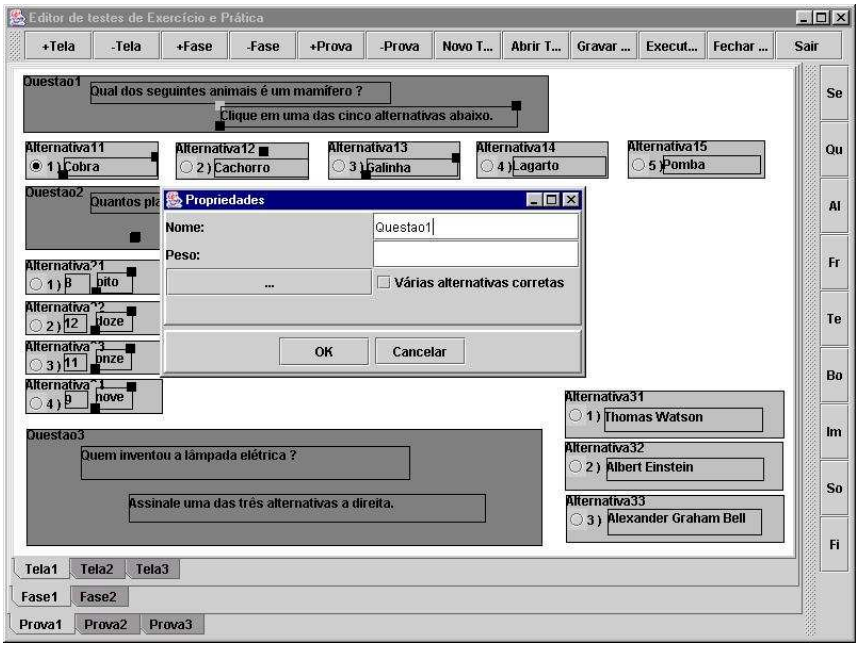


Figura 5-20Tela com a janela de definição de propriedades de questão

A única propriedade das alternativas implementada no protótipo HyperTest é a propriedade alternativa correta. É através dessa propriedade que o autor definirá qual ou quais são as alternativas corretas de uma questão. Somente se o autor definir que a questão possui várias alternativas corretas que o mesmo poderá selecionar mais do que uma alternativa como correta, caso contrário, o mesmo poderá selecionar apenas uma. A Figura 5-21 apresenta uma tela com a janela de definição da propriedade de uma alternativa.

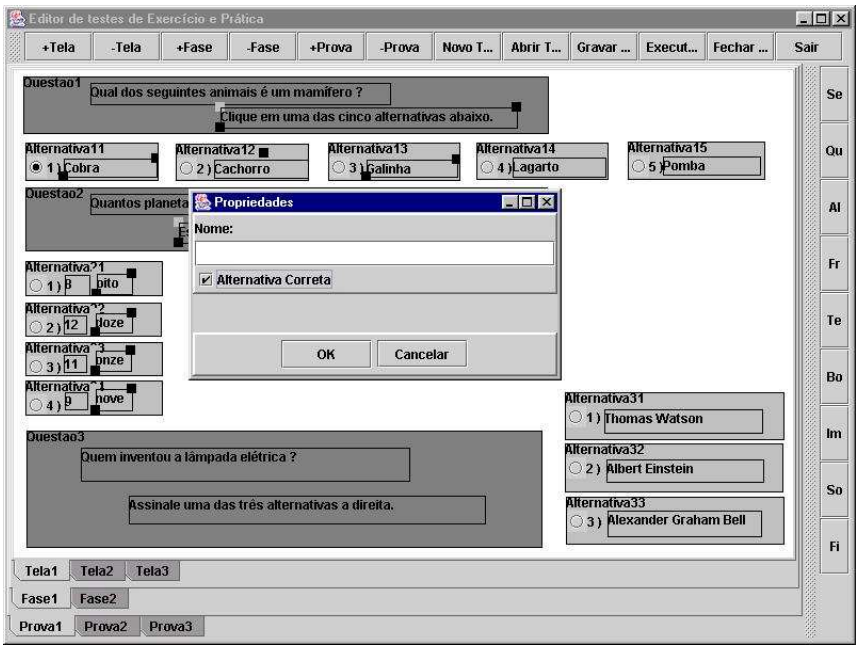


Figura 5-21Tela com definição da propriedade de alternativa

5.3.6 Visualizar ou executar um teste

A execução ou visualização do teste acontece quando o usuário clica no botão "Executar Teste", da barra de comandos. A Figura 5-22 apresenta uma tela de teste no modo de execução.

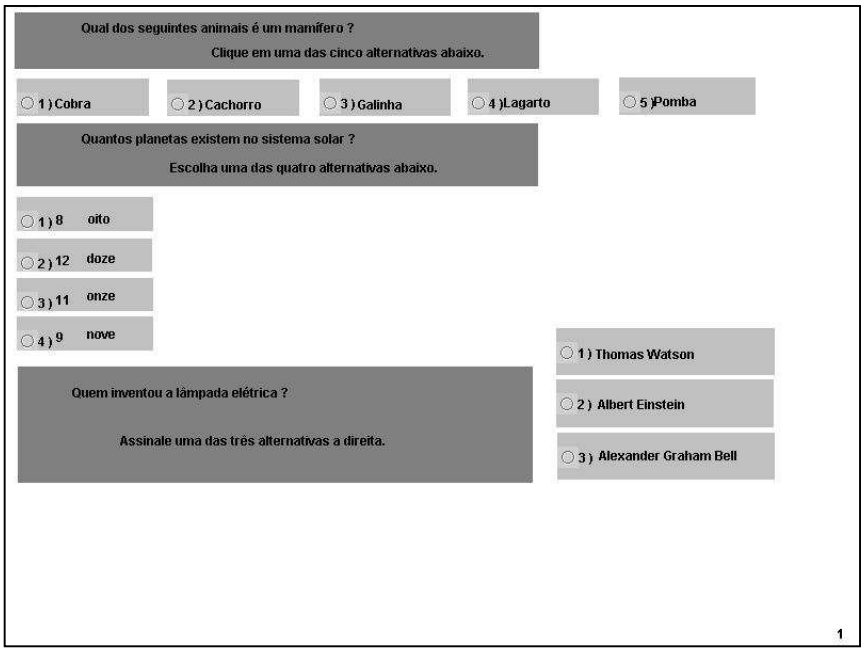


Figura 5-22Tela de um teste no modo de execução

A execução de um teste difere da visualização de um livro, pois no caso do teste deve ser feita uma checagem inicial de consistência do mesmo. A ferramenta de autoria deverá checar se tem alguma questão sem alternativa correta ou ainda se tem alguma questão sem alternativa associada, por exemplo.

5.3.7 Correção das questões

Após feita a checagem de consistência de um documento de teste, o usuário-leitor poderá visualizá-lo ou executá-lo. No processo de execução o usuário de um documento de teste irá fazer a leitura das questões e alternativas e após irá interagir respondendo as questões através da escolha de uma ou mais alternativas.

Uma outra decisão de projeto tomada para a implementação do HyperTest é que na execução de um teste a correção das questões se dá a cada final de fase. Isso poderia ser projetado de forma diferente, correção a cada final de página ou a cada final de prova. A

Figura 5-23 ilustra a correção de um teste através de uma janela, apresentando mensagem de que a resposta do usuário para a "questão 1" está correta e a Figura 5-24 apresenta um janela com mensagem de que a resposta do usuário para a questão 2 está incorreta.

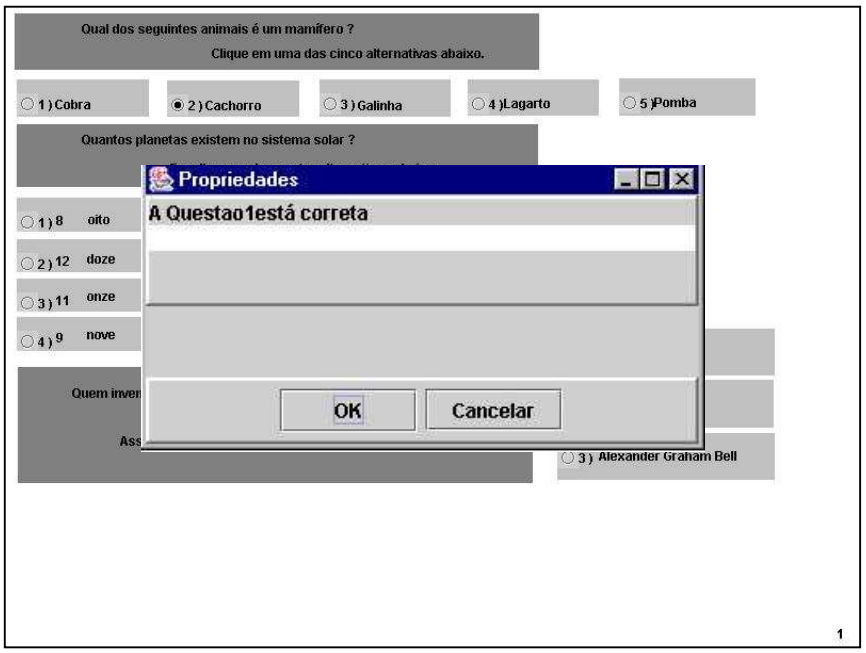


Figura 5-23Tela com janela apresentando acerto de questão

Figura 5-24Tela com janela apresentando erro de questão

5.4 Considerações sobre o desenvolvimento do HYPERTEST

Como um dos principais objetivos desse trabalho é promover o desenvolvimento de software com o enfoque de reutilização de artefatos, essa seção visa demonstrar e discutir como a aplicação HyperTest foi desenvolvida reutilizando projeto e código do framework HyperToolBuilder.

A Figura 5-25 apresenta as especializações das classes do framework HyperToolBuilder para o desenvolvimento da ferramenta de autoria HyperTest.

A estrutura e relacionamento de classes reutilizadas do framework para compor a ferramenta de autoria HyperTest é exatamente a mesma. Isso é um aspecto positivo do framework já que uma estrutura genérica é reutilizada. Nessa aplicação também se criou uma subclasse concreta de *AuthoringTool* denominada *TestAuthoringTool* que é responsável pelo controle de toda a ferramenta. Herdando também as funcionalidades de armazenamento em arquivo e de uma implementação padrão de execução de documento, através das classes *FileStorageManager* e *PlayerDefault* do framework, também reutilizadas na aplicação HyperBook.

Da mesma forma foram especializadas as classes *ToolBar* e *ToolPanel* para implementar a barra de comandos ou barra de atalho (optou-se por utilizar barra de atalho ao invés de barra de menu por brevidade) e a barra de ferramentas da aplicação.

A Figura 5-26 ilustra as especializações das classes do framework HyperToolBuilder para a composição da estrutura do documento de teste, manipulado pela ferramenta de autoria HyperTest.

Teste é o documento que será manipulado pela ferramenta de autoria HyperTest. Para criar um teste[22] foi especializada a classe abstrata do framework *Practice* criando a classe *Test* que representa esse documento multimídia. E a classe *Section* criando a subclasse concreta *Phase* representando as diferentes fases de um teste. Diferentemente da aplicação HyperBook, no HyperTest a classe *SectionComposite* foi reutilizada do framework por composição, como ela implementa o padrão *Composite* representa uma seção que pode ser composta por outras seções. Dessa forma a aplicação disponibiliza a instanciação em tempo de execução de objetos da classe *SectionComposite* para representar as provas de um teste,

[22] O conceito de Teste foi convencionado para a criação da aplicação HyperTest como sendo um documento multimídia dividido em três níveis de seções: provas subdivididas em fases, por sua vez subdivididas em telas.

esses objetos agregam instâncias da classe *Phase*, fazendo com que uma prova contenha várias fases. As fases agregam objetos da classe *Screen* que é uma subclasse concreta estendida da classe *SectionElement* do framework. Objetos da classe *Screen* representam as várias telas que compõe uma fase. Como a classe *Test* é subclasse de *Document,* ela herda o relacionamento com as superclasses *Section* e *SectionElement*.

A aplicação HyperTest exigiu além das classes de unidades de informação como *Sentence, Text, Button, Image, Sound* e *Movie*, que foram todas reutilizadas do framework, outras classes para compor um teste. As classes concretas *MultipleChoice*, *ChoiceDefault* e *MarkerDefault* foram estendidas das classes abstratas do framework *QuestionComposite*, *ChoiceComposite* e *Marker*. Elas implementam as funcionalidades de enunciado de questão de múltipla escolha, as várias alternativas da questão e o elemento para representar a seleção das alternativas respectivamente. Essas unidades de informação podem ser compostas por outras unidades de informação. Da mesma forma que a aplicação HyperBook, o protocolo de controle do relacionamento entre as unidade de informação e as possíveis ações sobre as mesmas também é herdado do framework. Apenas se fez necessário criar subclasses concretas da classe *Action* para disponibilizar na aplicação algumas ações como: saltar para uma determinada tela (*GoToScreen*) e corrigir questões (*CorrectQuestion*).

Através da Figura 5-25 e Figura 5-26 pode-se observar a substancial quantidade de reuso obtido para o desenvolvimento da aplicação HyperTest baseado no framework HyperToolBuilder. A partir da análise dos diagramas das figuras observa-se que seria necessário a criação de aproximadamente trinta e cinco classes para o desenvolvimento da aplicação,[23] sem a utilização do framework. Com a utilização do framework foram reusadas 11 classes do mesmo através de herança de classes abstratas e 24 classes foram reusadas por composição, através da instanciação de classes concretas do framework.

[23] Nem todas as classes do framework que aparecem no diagrama são reutilizadas para a ferramenta de autoria HyperTest. Além disso a contagem realizada inclui apenas as classes do domínio do problema, não incluindo classes de Visão e Controle do modelo MVC

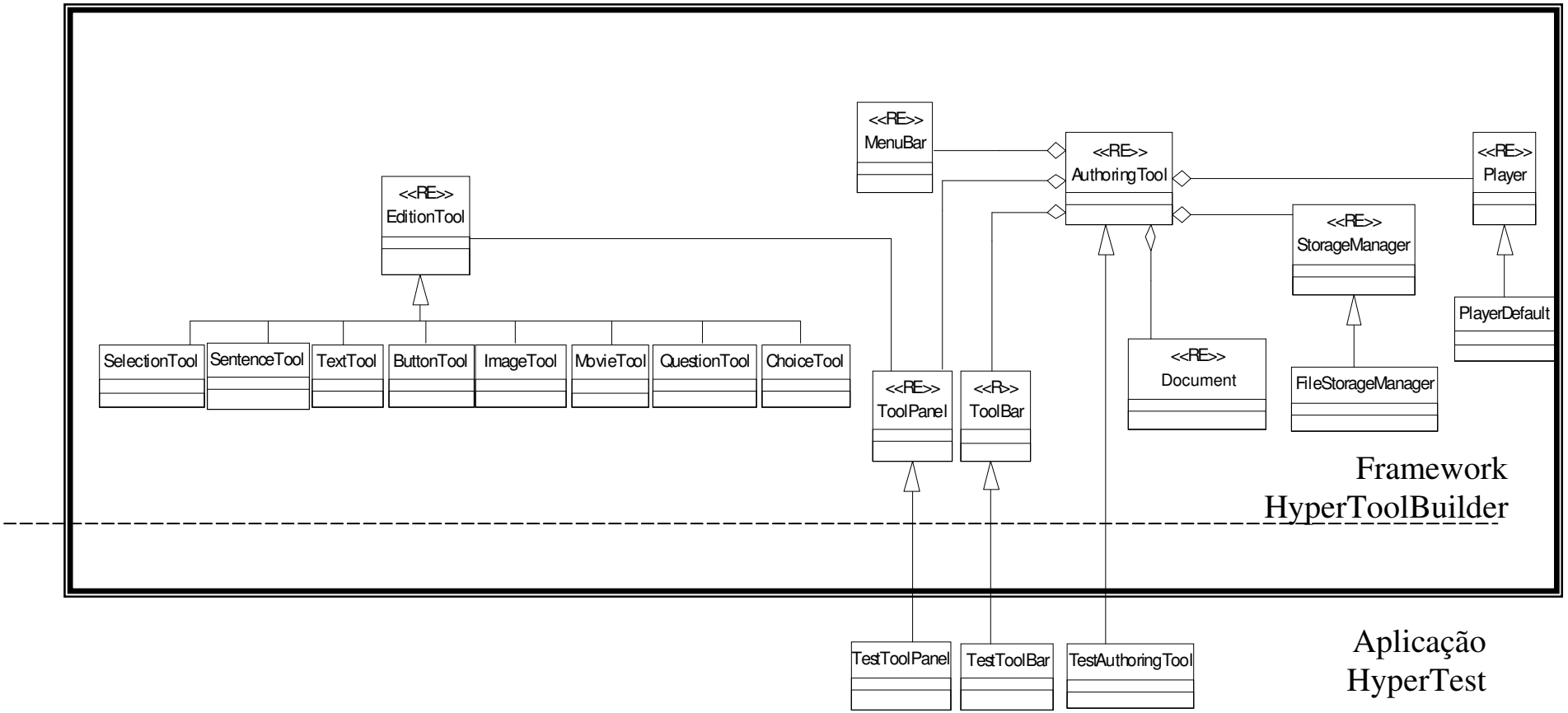


Figura 5-25Modelo de Objetos da ferramenta de autoria HyperTest, desenvolvida sob o framework HyperToolBuilder

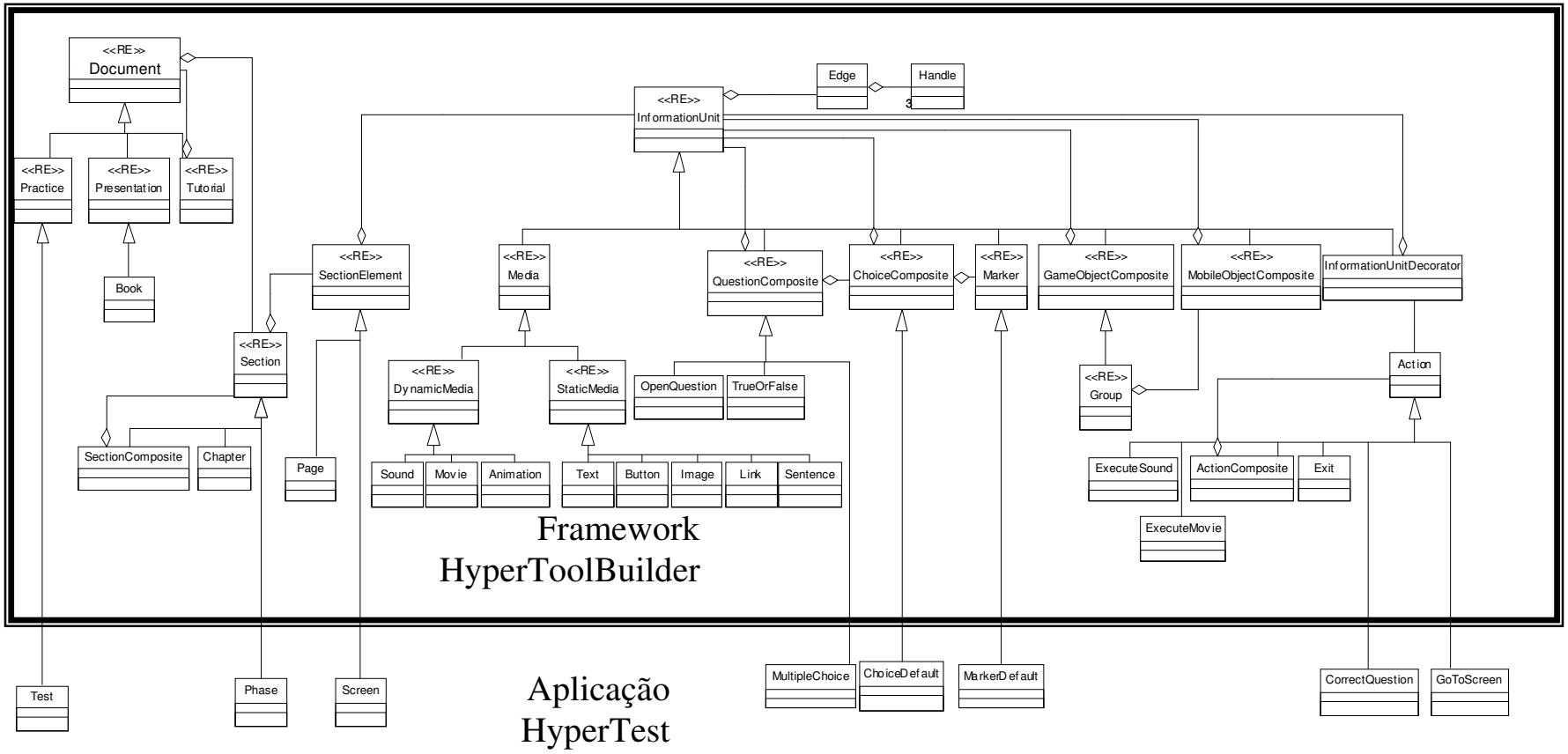


Figura 5-26Modelo de Objetos do documento(teste) da ferramenta de autoria HyperTest, desenvolvida sob o framework HyperToolBuilder

5.5 Criação de novas ferramentas de autoria

As ferramentas de autoria HyperBook e HyperTest descritas nas seções anteriores ilustram exemplos de aplicações que podem ser desenvolvidas sob o framework HyperToolBuilder. Mas conforme foi apresentado na seção 4.1.2, outras aplicações de ferramentas de autoria são previstas pelo projeto do HyperToolBuilder. A seguir são apresentadas algumas considerações sobre o desenvolvimento de algumas outras aplicações:

5.5.1 Ferramentas de autoria para apresentações multimídia

Para criar ferramentas de autoria para apresentações o usuário seguirá praticamente os mesmos passos da criação do HyperBook. A diferença de um livro eletrônico para uma apresentação multimídia está na estrutura de documento, a apresentação possui apenas telas que correspondem as páginas de um livro, mas não possui divisão em capítulos.

O usuário do framework deverá criar uma subclasse de *Presentation* para representar um documento de apresentação, após criar uma subclasse de *Section* e instanciar apenas um objeto dessa classe e associá-lo à subclasse de *Presentation*. Nessa instância serão associadas várias instâncias de uma subclasse de *SectionElement*, criada pelo usuário do HyperToolBuilder, para compor as várias telas de uma apresentação.

O usuário do framework deverá se preocupar em deixar transparente para os usuários autor e leitor a existência de uma única seção, os mesmos visualizarão apenas os elementos de seção. Para fazer isso o usuário do framework deverá criar uma subclasse de *PresentationView*, fazendo com que essa classe não exiba a divisão do documento em seção.

Além disso, criar uma subclasse de *MenuBar* e outra de *ToolBar* implementando nas mesmas botões para inserção e remoção apenas de elementos de seção, excluindo os botões de inserção e remoção de seções.

5.5.2 Ferramentas de autoria para jogos

O framework foi projetado prevendo a possível criação de ferramenta de autoria para apenas um tipo de jogo, o jogo de arrastar unidades de informação para determinados grupos – por exemplo, arrastar imagens de maçã, banana e uva para o grupo frutas.

Para criar uma ferramenta de autoria para esse tipo de jogo o usuário do framework terá que criar subclasses concretas de *MobileObjectComposite* e *Group* (apresentadas na seção 4.2.1.3) para realizar as funções dos objetos móveis e dos grupos respectivamente. O

usuário do framework terá também que criar ações que possibilitem aos objetos de subclasses concretas de *MobileObjectComposite* serem movidos e fixados nos objetos das subclasses concretas de *Group*. As ações previstas são: habilitar movimento de objetos, desabilitar movimento, fixar objetos em um grupo, levar objetos para grupo, somar pontos e voltar objeto ao lugar. As classes *MobileObject* e *Group* implementam o padrão de projeto *Composite* que faz com que tanto os objetos móveis quanto os grupos podem ser unidades de informação compostas por outras unidades de informação, dessa forma um objeto móvel pode ser um grupo de imagens mais um texto, por exemplo. Deverão ser criadas ferramentas para esses dois novos tipos de unidades de informação, subclasses concretas de *EditionTool*, e as mesmas serem adicionadas na barra de ferramentas, instâncias de subclasses de *EditionTool* deverão ser adicionadas na subclasses concreta de *ToolPanel*.

O usuário do framework deverá instanciar um objeto da classe *Game* que representará o documento que terá a estrutura de acordo com a divisão de seções e elementos de seção criadas pelo mesmo. Ele irá compor o documento através da criação de subclasses de *ElementSection* e *Section* e irá definir os métodos *createSection()* e *createSectionElement()*, para a instanciação de objetos dessas subclasses e o método de inicialização irá montar a estrutura do documento. Além disso, o usuário deverá definir o método abstrato *addPoints()* da classe *Section* para totalizar os pontos de cada seção do jogo.

5.5.3 Ferramentas de autoria para tutoriais

As ferramentas de autoria para tutoriais representam um dos grupos de aplicações mais complexo a ser desenvolvido sob o framework. Um documento de tutorial está previsto no framework através da classe abstrata *Tutorial*. Essa classe implementa o padrão de projeto *Composite* e dessa forma um tutorial será uma composição de vários outros documentos multimídia. Para compor um tutorial deve-se primeiramente instanciar objetos de subclasses de *Document*, mais especificamente de *Presentation* e *Practice*, e adicioná-los a um objeto de subclasse concreta de *Tutorial*.

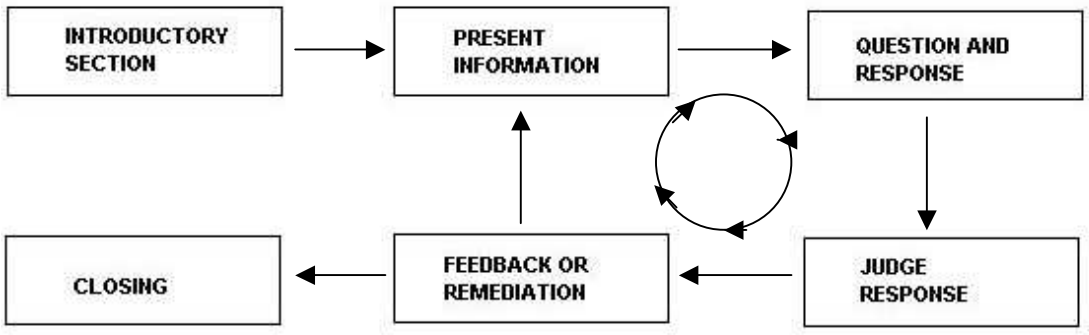


Figura 5-27A estrutura geral e seqüência de um tutorial (ALESSI, 2001)

A estrutura de um documento tutorial, segundo ALESSI (2001), é apresentada na Figura 5-27. Essa abordagem de tutorial apresenta inicialmente informações introdutórias ao usuário (apresentado na Figura 5-27 como Introductory Section), e após iniciando um ciclo de quatro etapas: na primeira etapa apresenta informações sobre um determinado assunto (Present Information – Figura 5-27), na segunda etapa realiza prática das informações através de exercícios ou jogos (Question and Response – Figura 5-27), na terceira etapa julga e avalia as práticas desenvolvidas(Judge Response- Figura 5-27) e na quarta etapa dá um retorno ao usuário de suas práticas e encaminha o mesmo para outra apresentação de informações (Feedback or Remediation – Figura 5-27), fechando o ciclo. Após serem percorridas todas apresentações e práticas do tutorial, ou por opção do usuário, serão apresentadas informações de encerramento do tutorial (Closing – Figura 5-27).

O usuário do framework usando esse conceito de tutorial deverá criar um objeto de uma subclasse de *Presentation,* para compor um documento para uma seção introdutória; outro objeto de subclasse de *Presentation,* para compor o documento de finalização do tutorial. Além disso, criar vários objetos de subclasse de *Presentation* e *Practice* para apresentar informações e as práticas respectivamente. Poderiam ser usadas classes já existentes no framework das aplicações HyperBook e HyperTest, usando um livro para apresentar informações e um teste para praticá-las.

Finalmente o usuário terá que definir os seguintes métodos abstratos da classe *Tutorial* para a execução do documento de tutorial: *presentIntroductorySection(), presentInformation(), presentQuestionAndResponse(), judgeResponse(), presentFeedBack* e

presentClosing(). Esses métodos deverão implementar as funcionalidades de controle de apresentação dos vários documentos que compõem o tutorial.

5.6 Resultados obtidos da utilização de uma aplicação

Alunos do curso do Ensino Médio Normal do Colégio Cenecista Sepé Tiarajú de Santo Ângelo (RS) foram submetidos a testes de utilização de uma ferramenta de autoria para livros eletrônicos – HyperBook - que foi instanciada a partir do framework HyperToolBuilder. O grupo consistiu em 31 alunos sem experiência na utilização de ferramentas de autoria.

O autor desse trabalho deu uma explanação sobre os conceitos e o modo de utilização da ferramenta de autoria, e propôs a tarefa aos alunos de construírem em grupos de quatro componentes livros eletrônicos sobre assuntos de seus interesses. Funcionalidade, usabilidade e operacionalidade da ferramenta de autoria HyperBook foram testadas.

A partir da utilização prática do HyperBook se observou uma série de pequenas novas funcionalidades a serem acrescentadas ao framework, como:

- Recursos para recortar, copiar e colar unidades de informação de um local do documento para outro;
- A possibilidade de criar e manipular camadas em um elemento de seção para que várias unidades de informação possam ser empilhadas. Com isso ter-se-ia a possibilidade de enviar uma unidade de informação para frente ou para trás de outra;
- A possibilidade de edição dos elementos de seção em tela cheia. A ferramenta HyperBook implementada faz uso de uma área inferior da tela e outra à direita para a exibição das abas para a seleção das páginas e capítulos, respectivamente. Dessa forma o tamanho em pixels da página no modo de autoria é menor que o tamanho da mesma no modo de visualização (leitura), já que o modo de visualização acontece em tela cheia. Esse problema poderia ser resolvido dando a possibilidade ao usuário-autor de ocultar essas áreas. Assim, o autor seria capaz de usar toda a tela do computador (tela cheia) para que o tamanho da página do modo de autoria seja fiel ao tamanho da página no modo de visualização do livro. Outra forma para resolver esse problema seria fazer com que a própria ferramenta redimensionasse automaticamente de um tamanho de página menor do modo de autoria para um tamanho de página maior(tela cheia) no modo de visualização;

- Recursos para desfazer comandos já executados;
- Recursos para girar unidades de informação, para que as mesmas possam ser apresentadas inclinadas, por exemplo;
- Recursos para mover uma seção de uma posição para a outra no documento; da mesma forma ter a possibilidade de trocar a ordem dos elementos de seção dentro de uma seção;
- Colocar imagens no fundo de um elemento de seção e também de unidades de informação;
- Modificar a cor da moldura e dos handles de acordo com a cor do fundo do elemento de seção;
- Criar efeitos no momento de abertura de cada elemento de seção na execução do documento; ter efeitos também na apresentação de cada unidade de informação na execução do documento;
- Adicionar um documento de livro já armazenado em outro documento;
- Possibilitar a inserção de textos criados em outros processadores de texto;

5.7 Considerações Finais

Após o término da implementação das aplicações HyperBook e HyperTest baseadas no HyperToolBuilder e da experiência de uso do HyperBook pode-se concluir que foi possível gerar aplicações sobre o framework.

Através da análise realizada sobre a instanciação das aplicações a partir do framework, nas seções 5.2. e 5.4, pode-se constatar que ocorreu um alto grau de reuso no desenvolvimento dessas aplicações. Constata-se um alto reuso, já que nas duas aplicações todas as classes necessárias estavam previstas no framework, sendo que todas foram reutilizadas. Fazendo uma média entre as duas aplicações se constata um reuso de classes de aproximadamente 31% por herança e 69% por composição, isso corresponde a um bom resultado já que quanto maior o percentual de reuso por composição mais adequado é o framework. Outra consideração importante é que todo o relacionamento e protocolo de comunicação entre as classes do framework são herdados pela aplicação.

A partir disso chega-se à conclusão de que o framework HyperToolBuilder é capaz de suportar a geração de aplicações do domínio de ferramentas de autoria para documentos multimídia, requisito mínimo para a verificação de sua aplicabilidade e que a reutilização do framework para o desenvolvimento das aplicações propiciou uma substancial diminuição do esforço no desenvolvimento das mesmas.

6. CONCLUSÃO

6.1 Resultados Obtidos

Pode-se dizer que o objetivo geral desse trabalho foi atingido através da conclusão da análise, projeto e implementação da primeira versão do framework HyperToolBuilder, apresentado no capítulo 4. O HyperToolBuilder é um framework orientado a objetos que provê uma infra-estrutura para o desenvolvimento de ferramentas de autoria, baseadas no paradigma de páginas ou cartões, para documentos multimídia. A construção do framework HyperToolBuilder foi realizada utilizando-se principalmente a abordagem de projeto dirigido por exemplo (JOHNSON, 1996) e a abordagem de projeto dirigido por pontos flexíveis (PREE, 1995) em conjunto com a aplicação de metapadrões (PREE, 1995) e padrões de projeto (GAMMA, 1994).

Os principais resultados obtidos deste trabalho são:

A disponibilização do projeto arquitetural de um framework orientado a objetos, que pode ser reutilizado para a criação de diferentes ferramentas de autoria para documentos multimídia. O projeto de classes pode ser reutilizado independentemente de linguagem de programação. Os principais pontos adaptáveis do framework são as diferentes estruturas de documentos e unidades de informação que podem ser criadas pelo usuário do mesmo. Além disso, o framework provê o controle do fluxo de execução das colaborações existentes entre suas classes, sendo que as classes reutilizadas através de especialização e composição para a construção das aplicações são chamadas pelas classes do framework;

A disponibilização de um protótipo de implementação do framework HyperToolBuilder em linguagem de programação JAVA. Programadores com conhecimento dessa linguagem podem fazer reutilização não apenas de projeto, mas também de código. Na implementação foram utilizados projetos de classes da própria linguagem Java, a fim de minimizar os conceitos a serem entendidos pelo implementador;

Uma abordagem de criação e manipulação de documentos multimídia usando tecnologia de orientação a objetos voltada para o reuso;

Um relato de experiência no processo de desenvolvimento de artefatos reutilizáveis para aplicações com recursos multimídia. O trabalho apresenta exemplos de aplicação prática do uso de metapadrões e padrões de projeto;

A disponibilização de duas aplicações de ferramentas de autoria para documentos multimídia – HyperBook e HyperTest – desenvolvidas a partir do framework HyperToolBuilder;

A experiência de uso da ferramenta de autoria HyperBook desenvolvida sob o framework por usuários leigos em informática. Essa experiência possibilita uma realimentação do framework através de sugestões de melhoria das aplicações.

O uso do HyperToolBuilder para o desenvolvimento de ferramentas de autoria não apenas promoverá o reuso em alta escala, mas também reduzirá o nível de complexidade de desenvolvimento dessas aplicações que incluem características complexas como vídeo, música, gráfico e hipertexto. Esse trabalho produziu recursos de projeto e código de software para a futura criação de novas ferramentas de autoria. Permite que desenvolvedores não necessitem produzir software do ponto zero, facilitando consideravelmente o desenvolvimento de tais aplicações, uma vez que a grande maioria dos recursos necessários já está disponível.

O framework fornece subsídios para que pequenas empresas, desenvolvedores de software e acadêmicos da área da computação possam desenvolver novos softwares atraentes, personalizados e que promovam o uso da criatividade por parte dos usuários. Ele contribui para o avanço do conhecimento científico na área de desenvolvimento de software multimídia, pois constitui-se em um projeto e implementação reutilizável de software nessa área. O mesmo deverá servir de suporte para a pesquisa de trabalhos futuros nessa área, já que possui uma estrutura muito flexível para isso.

6.2 Limitações

Após ter levado em consideração inúmeros aspectos de ferramentas de autoria para documentos multimídia para a construção do framework, percebe-se que o mesmo apresenta uma estrutura bastante flexível para a criação de tais aplicações. Porém, o HyperToolBuilder possui algumas limitações, entre elas:

- Necessidade de adaptar o framework para que o mesmo suporte as funcionalidades verificadas através do teste da ferramenta de autoria HyperBook, apresentadas no capítulo anterior;
- Falta de uma avaliação efetiva do framework. Até o momento o framework HyperToolBuilder ainda não foi suficientemente avaliado para verificar se ele é completamente adequado ao desenvolvimento de qualquer ferramenta de autoria para documentos, já que o mesmo foi reutilizado para produzir apenas dois tipos de ferramentas de autoria. Outras aplicações como ferramentas de autoria para apresentações multimídia, jogos e tutorias precisam ser desenvolvidas para a evolução do mesmo.
- A falta de uma avaliação sobre decisões de projeto tomadas durante o desenvolvimento do HyperToolBuilder. Durante o projeto algumas decisões foram tomadas para que a sua implementação fosse concluída. Por exemplo, foi utilizado o padrão de projeto Decorator para a associação entre as unidades de informação e suas possíveis ações. Será que essa é a melhor abordagem para a associação entre os dois conceitos ? Essa questão somente poderá ser respondida a partir da observação do uso desses recursos em várias aplicações diferentes de ferramentas de autoria.
- A falta do desenvolvimento de uma ferramenta de autoria para a criação de tutorias. Tutorias são documentos bem mais complexos que os livros e testes já implementados pelas aplicações desenvolvidas sobre o framework. Uma ferramenta de autoria para tutorias traria novos ganhos de flexibilidade ao framework.

6.3 Trabalhos futuros

O framework HyperToolBuilder constitui um suporte para a construção de ferramentas de autoria para documentos multimídia, que deve ser estendido para futuros esforços de pesquisa.

A seguir são descritas extensões que podem ser agregadas ao trabalho ora desenvolvido:

- Adicionar funcionalidades ao framework para que a partir do mesmo se possa produzir ferramentas de autoria colaborativas. A possibilidade de autoria através de redes de

computadores é uma característica que está cada vez mais presente em aplicações multimídia;

- Embutir no framework uma linguagem de programação (scripting) que possa ser reutilizada por todas as ferramentas geradas a partir dele;
- Disponibilizar no framework recursos para que possam ser criadas ferramentas de autoria com novos mecanismos de interação, novas interfaces gráficas, mecanismos de edição e criação de efeitos de vídeo digitais, recursos para estruturação, indexação e resumo de textos, e possibidade de subtituição de um recurso por outro (substituir um arquivo de som por um texto caso não seja possível executar o som) e novos conceitos de entrega dos documentos;
- Adicionar a capacidade ao framework de gerar aplicações que auxiliem em váriass fases da criação de um documento multimídia (na fase da pré-autoria, por exemplo). Essa versão do apenas provê recursos para o processo de construção e execução de documentos multimídia.
- Adicionar conceitos e funcionalidades ao framework para o mesmo possa gerar ferramentas de autoria que possuam recursos para separar a descrição e manipulação da estrutura de conteúdo, de apresentação e da estrutura conceitual de documentos multimídia.
- Adicionar capacidade ao framework de desenvolver tutoriais inteligentes, que envolveria outros conceitos como o tratamento do conhecimento do estudante, conhecimento do especialista, interface com o usuário, etc... A tarefa de desenvolver este tipo de software desde o início (sem uso de um framework) seria muito mais trabalhosa do que sendo auxiliado por uma estrutura pré-elaborada;
- Identificar e controlar os usuários leitores, bem como armazenar as etapas de execução do documento por parte do usuário-leitor;
- Fornecer documentação e *cookbooks* para vários tipos de público para que a utilização do framework realmente aconteça; descrever o que pode ser feito a partir do framework para as pessoas que estão decidindo qual framework usar; fornecer documentação que ensine de forma sumária a usar o framework para as pessoas que desejam aprender aplicações típicas a partir do mesmo; fornecer documentação completa sobre detalhes do projeto do framework para as pessoas que pretendem

desenvolver aplicações mais complexas. Ou ainda, criar um ambiente gráfico para facilitar o uso e desenvolvimento de aplicações a partir do HyperToolBuilder;

- Disponibilizar modelos (templates) de documentos, seções e elementos de seções. Modelos de documentos, seções e elementos de seções auxiliariam o usuário do framework a fornecer recursos prontos ao usuário-autor nas ferramentas de autoria;

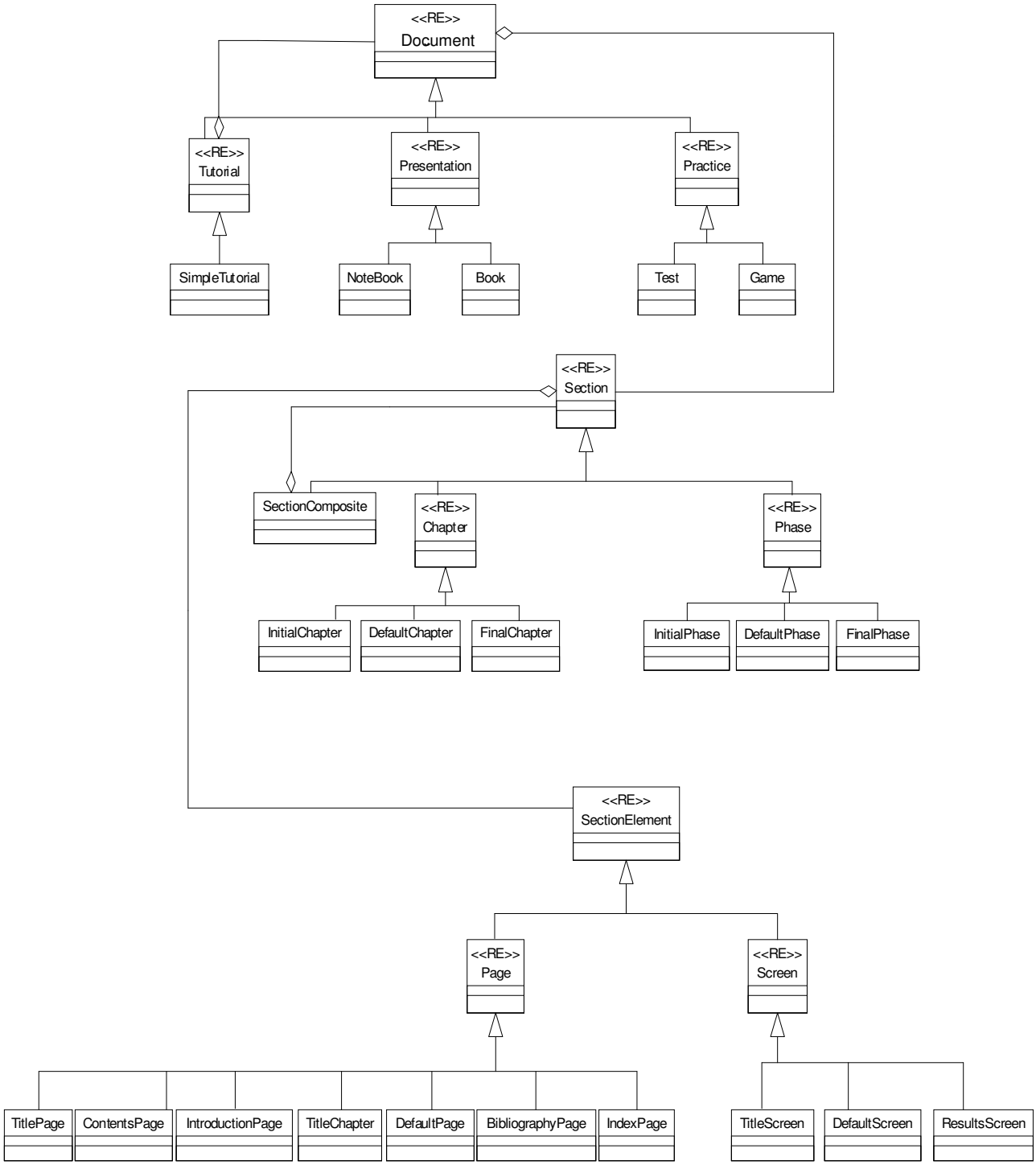
6.4 Considerações finais

Significativas mudanças estão sendo promovidas pela tecnologia multimídia. Através da combinação de textos, gráficos de alta resolução, animações, vídeo, sons e utilização de telas sensíveis ao toque, os documentos multimídia interativos provêem uma nova abordagem para acessar grande quantidade de conhecimento com qualidade. Além disso, dando a possibilidade de distribuí-lo para várias pessoas ao mesmo tempo.

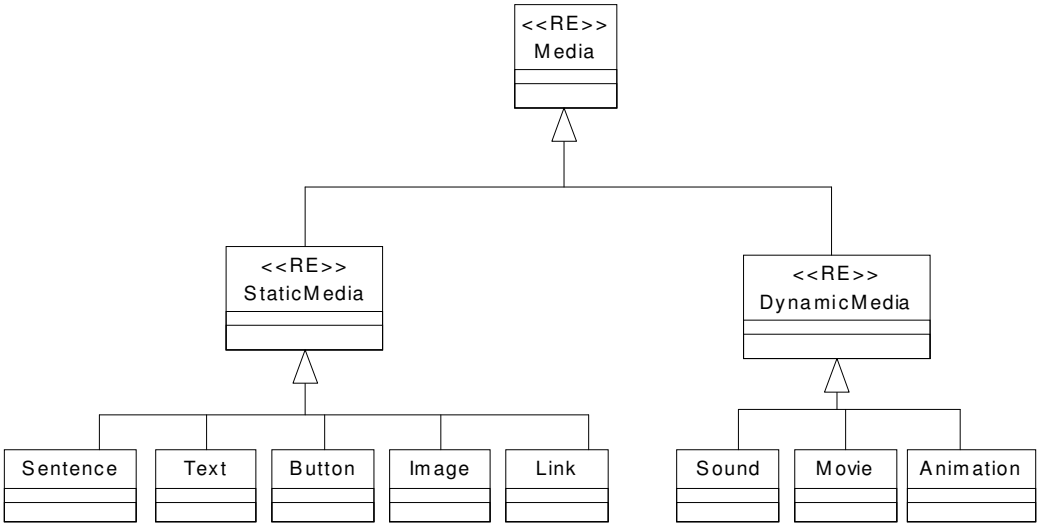
Porém os autores de documentos multimídia necessitam fazer melhor uso das mídias, utilizar novas metáforas de autoria multimídia, criar documentos com novas técnicas de interação e novos conceitos de entrega desses documentos. A partir desse panorama se produziu um estudo a respeito de algumas aplicações de ferramentas de autoria para a produção desses documentos, abstraindo aspectos julgados importantes e se construiu um artefato de software reutilizável, fornendo um suporte para facilitar o surgimento de novas aplicações, com novos paradigmas e novos recursos. O framework HyperToolBuilder materializa um conjunto de abordagens propostas para facilitar o desenvolvimento de ferramentas de autoria, e constitui em uma contribuição para a reflexão em torno da reutilização de software no desenvolvimento de aplicações multimídia.

O aprofundamento da pesquisa introduzida por essa dissertação poderá auxiliar a tornar a abordagem voltada à promoção de reuso de software uma prática amplamente difundida, já que apresenta um exemplo prático da abordagem de frameworks orientados a objetos em um domínio específico. Além disso, o framework desenvolvido contribui para diminuir de forma considerável o esforço no desenvolvimento de ferramentas de autoria para documentos multimídia.

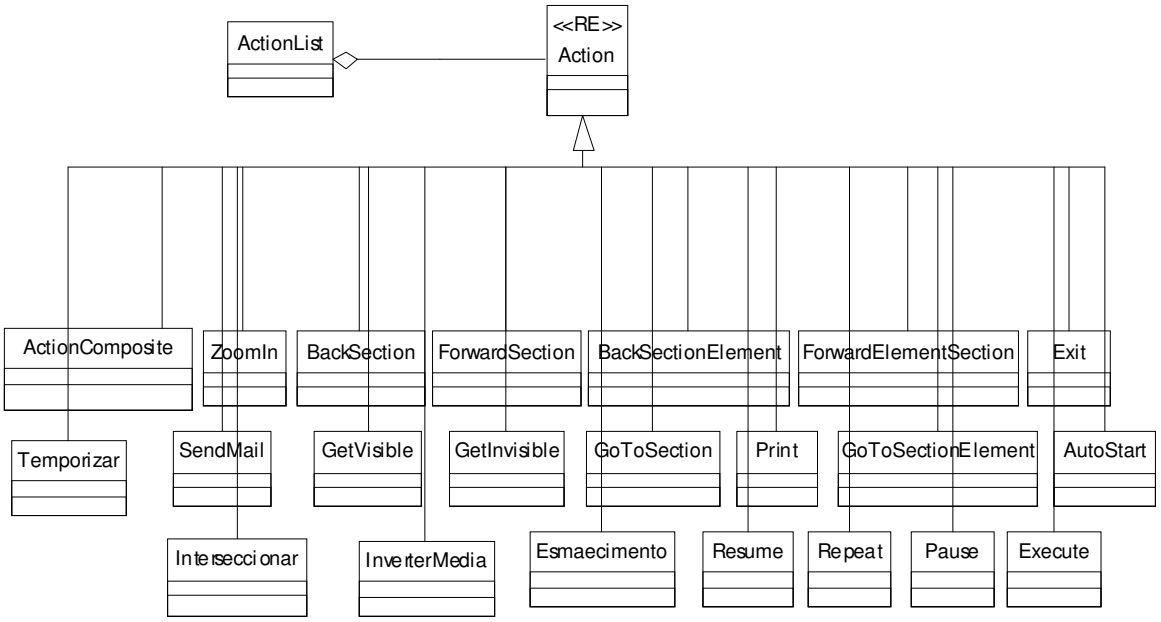
ANEXO I Diagrama de classes da estrutura de documentos de ensino



ANEXO II Diagrama de classes das mídias



ANEXO III Diagrama de classes das ações



REFERÊNCIAS BIBLIOGRÁFICAS

ACKERMANN, P. **Direct Manipulation of Temporal Structures in a Multimedia Application Framework**. In proc. of the ACM Multimedia. 1994.

ALESSI, Stephen M., Stanley R. Trollip. **Multimedia for Learning – Methods and Development** – 3rd ed. Allyn & Bacon, 2001.

APPLE. **Multimedia Demystified - A Guide to the World of Multimedia from Apple Computer**, Inc. Random House/New Media Series, 1994.

ARANGO, G.; PRIETO-DIAZ, **R. Domain analysis concepts and research directions**. In: PRIETO-DIAZ, R.; ARANGO, G. **Domain analysis and software modeling**. Los Alamitos: IEEE Computer Society Press, 1991.

AUTHORWARE. **Inc. Authorware Professional Manual**. Redwood City, CA, 1989.

BLAKOWSKI, G. and R. Steinmetz. **A Media Synchronization Survey: Reference Model, Specification, and Case Studies**. IEEE Journal on Selected Areas in 1996.

BOTAFOGO, R., D. Mossé. **The MORENA Model for Hypermedia Authoring and Browsing**. In proc. of the IEEE Int. Conf. on Multimedia Computing and Systems, pp. 42-49, Washington, DC, May 1995.

BUFORD, John F. Koegel. **Multimedia Systems**. ACM Press - SIGGRAPH Series - New York, New York, 1994. Apud WILLRICH, Roberto. Conception Formelle de Documents Hypermédias Portables. Toulouse (França): Universidade Paul Sabatier, 1996. Tese de Doutorado.

BULTERMAN, Dick C.A, Lynda Hardman. **Multimedia Authoring Tools: State of the Art and Research Challenges**. CWI:Centrum voor Wiskunde em Informatica, Kruislan 413, 1098 SJ Amsterdam. Springer-Verlag, 1995.

BUSCHMANN, F., 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.

CAMPBELL, Roy H., Nayeem ISLAM. **A technique for documenting the frameworks of an object-oriented system**. In Proceedings of Second International Workshop on Object-Oriented in Operation Systems, pages 288-300, Setembro 1992.

CAMPO, Marcelo Ricardo. **Compreensão Visual de Frameworks através de Introspeção de Exemplos**. PhD Thesis. Porto Alegre: UFRGS/II/ CPGCC. Março de 1997.

COAD,P, E. Yourdon. **Análise baseada em objetos**. Rio de Janeiro: Campus, 1992.

COAD, P, E. Yourdon. **Projeto Baseado em objetos**. Rio de Janeiro: Campus, 1993.

COLEMAN, D. et al. **Object-oriented development: the Fusion method**. Englewood Cliffs: Prentice Hall, 1994.

DENNIS, Brian M., Michael A. Harrison. **Technical Issues in Hypermedia Authoring Systems**. Proceedings of the 40th IEEE Computer Society International Conference (COMPCON'95), 1995.

DIAZ, M., P. Sénac. **Time Streams Petri Nets, a Model for Multimedia Streams Synchronization**. In proc. Of the First International Conference on Multi-media Modeling - Vol. 1, Singapore, World Scientific, Tat-Seng Chua & T. L. Kunii (Eds.), pp. 257-273, November 1993.

DRAPEAU, G.D., H. Greengiled, H. MAEstro**. A Distributed Multimedia Authoring Environment**. In proc. of the USENIX Summer ' 91 Conference, pp. 315-328. Nashville TN, 1991.

EGGENSCHWILER,Thomas, 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.

EYES 2.0 Reference Manual. Center for Productivity Enhancement. University of Massachusetts-Lowell. 1992.

FACH, Peter W. **Design Reuse through Frameworks and Patterns**. IEEE Transactions on Software Engineering. New York, October 2001.

FAQ. **Multimedia Authoring Systems FAQ**. Version 2.13. Disponível em: <http://www.tiac.net/users/jalisglar/MMASFAQ.HTML>. Acesso em: jun. 1997.

FAYAD, Mohamed E., Douglas C. Schmidt**. Object-oriented application frameworks**. Communications of the ACM, 40(10):32-38, October 1997.

FLUCKIGER, François. **Understanding Networked Multimedia: Applications and Technology**. Prentice Hall International (UK) Limited, 1995.

FREIBERGER, Evandro César. **Suporte ao Uso de Frameworks Orientados a Objetos com base no Histórico do Desenvolvimento de Aplicações**. Santa Catarina: Programa de Pós-Graduação em Ciência da Computação da UFSC, 2002. Dissertação de Mestrado.

FURLAN, José Davi. **Modelagem de Objetos através da UML – the Unified Modeling Language** – São Paulo: Makron Books, 1998.

GAMMA, E. **Objektorientierte Software – Entwicklung am Beispiel von ET++: Design – Muster, Klassenbibliotek**, Werkzeug. Doctoral Thesis, Universidade de Zurique, 1991. Publicado por Springer Verlag, 1992.

GAMMA, E., **Design Patterns: Abstraction and Reuse of Object – Oriented design**. In Proceedings of the ECOOP'93 Conference. KaiserLautern, Germany: Publicado por Springer Verlag, 1993.

GAMMA, E. Helm R.; Johnson R.; Vlissides J. **Design Patterns – Microarchitetures for Reusable Object-Oriented Software**. Reading, Massachussets: Addison-Wesley, 1994.

GAMMA, Erich. **Padrões de Projeto: soluções reutilizáveis de software orientado a objetos**. Porto Alegre: Bookman, 2000.

GIBBS, S. **Composite Multimedia and Active Objects**. In proc. of the OOPSLA ' 91. Phoenix, Arizona. October, 1991.

HARDMAN, L., D.C.A Bulterman. Authoring **Support for Durable Interactive Multimedia Presentations**. STAR Report in Eurographics ' 1995.

HOLLAND, Ian M. **Specifying reusable components using contracts**. In O. Lehrmann Hadson, editor, European Conference on Object-Oriented Programming, pages 187-308. Springer-Verlag, 1992.

HUDSON, S.E., C.H. Hsi. **A Framework for Low Level Analysis and Synthesis to Support High Level Authoring of Multimedia Documents**. Graphics Visualization and Usability Center Technical Report GVU-93-14. Georgia Institute of Technology, 1993.

ISO 8613 Information Processing - Text and Office Systems - Office Document Architecture (ODA) and Interchange Format (ODIF), 1998.

JACOBSON, I. et al. **Object-oriented software engineering – a use case driven approach**. Reading: Addison-Wesley, 1992.

JBUILDER 7 Documentation. Borland Software Corporation. 1997-2002. Disponível em: <http://info.borland.com/techpubs/jbuilder/index7.html> Acesso em: fev. 2003.

JFONTCHOOSER v0.22 updated 5/2/2003. Disponível em <http://www.cruftworks.com/software/JFontChooser/> Acesso em: set. 2003.

JOHNSON, R. E., Brian Foote. **Designing reusable classes**. Journal of Object-Oriented Programming, 1(2):22-35, June/July 1988.

JOHNSON, R. E. **Documenting frameworks using patterns**. SIGPLAN Notices, New York, v.27, n.10, Oct 1992. Trabalho apresentado na OOPSLA, 1992.

JOHNSON, R. E. **How do design frameworks**. 1993 Disponível por FTP anônimo em <st.cs.uiuc.Edu> (dez.98).

JOHNSON, R. **How to develop frameworks**. Tutorial Notes of ECOOP 96. Linz, Austria: July, 1996.

KEEBOOK Creator. Disponível em: <http://www.keebook.com>. Acesso em: 02 out. 2003.

KLAS, W., E.J. Neuhold, M. Schrefl. **Using an Object-Oriented Approach to Model Multimedia Data**. Computer Communication 13(4):204-216, 1990.

LARMAN, Craig. **Utilizando UML e padrões: uma introdução à análise e ao projeto orietados a objetos**. Trad. Luiz A. Meirelles Salgado. – Porto Alegre: Bookman, 2000.

LEWIS, Ted; Andert, G.; Calder, P.; Gamma, E.; Pree, W.; Rosenstein, L.; Schmucker, K.; Weinand, A.; Vlissides, J.M. **Object-Oriented Application Frameworks**. Manning Publications, 1995.

LITTLE, T.C.D., A. Ghafoor. **Synchronization and Storage Models for Multimedia Objects**. IEEE Journal on Selected Areas in Communication 8(3):413-427, April 1990.

MARTIN, J., J. ODELL. **Análise e projeto orientados a objetos**. São Paulo: Makron Books, 1995.

MICROSOFT POWERPOINT. Microsoft Office 2003. Microsoft Corporation. Estados Unidos, 2003. Dispnonivel em: < http://office.microsoft.com/home/default.aspx> Acesso em: 03 out. 2003.

MULTIMEDIA BUILDER. Disponível em: <http://www.mediachance.com/oldindex.html> Acesso em: 03 out. 2003.

MURATA, T**. Petri Nets: Properties, Analysis and Applications**. IEEE 77(4):541-580. April, 1989.

NEWCOMB, S. et al. **The HyTime Hypermedia/Time-based Document Structuring Language**. Communication of the ACM 34(11):159-166, 1991.

OMG Unified Modeling Language Specification (Action Semantics). Object Management Group. UML V1.4 AS adopted December 2001.Disponivel em: <http://www.omg.com> Acesso em: fev. 2003.

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

PERCEPTION. Disponível em: <http://www.questionmark.com/perception/help/v3/v3manuals/getstart/v3_percgetstart_TOC.html> Acesso em: 04 out. 2003.

PREE, Wolfgang. **Design patterns for object-oriented software development**. Reading: Addison-Wesley, 1995.

RATIONAL Rose Enterprise Edition. RATIONAL SOFTWARE CORPORATION.. 2002. Disponível em: <http:// www.rational.com> Acesso em: set. 2003.

ROGERS, Gregory F. **Framework – Based Software Development in C++**. New Jersey:Prentice – Hall, 1997.

RUMBAUGH, J. et al. **Modelagem e projetos baseados em objetos**. Rio de Janeiro: Campus, 1994.

SILVA, Ricardo Pereira e. **Suporte ao Desenvolvimento e uso de Frameworks e Componentes**. Porto Alegre: Instituto de Informática da UFRGS, 2000a. Tese de Doutorado.

SCHLOSS, G.A., M.J. Wynblatt. **Building Temporal Structures in a Layered Multimedia Data Model**. In proc. ACM Multimedia ' 94, pp. 271-278, 1994.

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

SCHMIDT, Doug, Paul STEPHENSON. **Experience using design patterns to envolve 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.

SCHMID, Hans Albrecht**. Design patterns for constructing the hot spots of a manufacturing framework.** Journal of Object-Oriented Progamming, 9(3), jan 1996.

SÉNAC, P., M. Diaz, A Léger, P. de Saqui-Sannes. **Modeling Logical and Temporal Synchronization in Hypermedia Systems**. IEEE Journal on Selected Areas in Communication 14(1):84-103, 1996.

SINTES, Tony. **Aprenda Programação Orientada a Objetos em 21 Dias**. São Paulo: Pearson Education do Brasil. Makron Books, 2002.

JAVA 2 SDK. **Standard Edition Documentation - Version 1.4**. API Specification. Sun Microsystems. 1995-2003. Disponível em: <http://java.sun.com/j2se/1.4.1/download.html> Acesso em: fev. 2003.

JAVA TUTORIAL. Sun Microsystems**.** 2003. Disponível em: <http://java.sun.com/docs/books/tutorial/index.html> Acesso em: set. 2003.

JAVA MEDIA FRAMEWORK API. Sun Microsystems**.** 2003. Disponível em: <http://java.sun.com/products/java-media/jmf/index.html> Acesso em: set. 2003.

TALIGENT. **Building object-oriented frameworks**. [S.l.]: Taligent, 1994. White paper. Apud SILVA, Ricardo Pereira e. Suporte aos Desenvolvimento e uso de Frameworks e Componentes. Porto Alegre: Instituto de Informática da UFRGS, 2000a. Tese de Doutorado.

TALIGENT. **Leveraging object-oriented frameworks**. [S.I]: Taligent, 1995. White paper. Apud SILVA, Ricardo Pereira e. Suporte aos Desenvolvimento e uso de Frameworks e Componentes. Porto Alegre: Instituto de Informática da UFRGS, 2000a. Tese de Doutorado.

TOOLBOOK Instructor. Disponível em <http://home.click2learn.com/en/toolbook/> Acesso em: 03 out. 2003.

VISUAL CLASS. Disponível em: <http://www.visualclass.com.br/>. Acesso em: 03 out. 2003.

WANG, H.K., J.L.C. Wu. **Interactive Hypermedia Applications: A Model and Its Implementation**. Software-Practice and Experience 25(9):1045-1063, 1995.

WEINAND, André, Erich Gamma, 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.

WIRFS-BROCKS, Rebecca. Ralph E. Johnson. **Surveying Current Research in Object-Oriented Design**. Communications of the Acm, 33(9):04-124, 1990.

WIRFS-BROCK, A. et al. **Designing reusable designs: Experiences designing object-oriented frameworks**. In: OBJECT-ORIENTED PROGRAMMING SYSTEMS, LANGUAGES AND APPLICATIONS CONFERENCE; EUROPEAN CONFERENCE ON OBJECT-ORIENTED PROGRAMMING, 1991, Ottawa. Addendum to the proceedings… Ottawa: [s.n.], 1991.

WILLRICH, Roberto. **Conception Formelle de Documents Hypermédias Portables**. Tese apresentada no Laboratoire d'Analyse et d'Architecture des Systèmes du C.N.R.S e m vista de obtenção to título de Docteur pela Universidade Paul Sabatier. Toulouse (França), Setembro de 1996.

WILLRICH, R., P. Sénac, M. DIAZ, P. de SAQUI-SANNES. **A Formal Framework for the Specification, Analysis and Generation of Standardized Hypermedia Documents**. Third IEEE International Conference on Multimedia Computing and Systems (ICMCS ´96), pp. 399-406, Japão, 1996.

YANG, C.C., J.H. Huang. **A Multimedia Synchronization Model and Its Implementation in Transport Protocols**. IEEE Journal on Selected Areas in Communications 14(1):212-255, 1996.

ZUKOWSKI, John, Scott Stanchfield**. Fundamentals of JFC/Swing: Part II**. MegaLang Institute 1999. Disponível em: <http://developer.java.sun.com/developer/onlineTraining/GUI/Swing2/index.html> Acesso em: fev. 2003.

ZUKOWSKI, John. **Magic with Merlin: Scrolling tabbed panes.** September 2001. Disponível em: <http://www-106.ibm.com/developerworks/java/library/j-mer0905/> Acesso em: fev. 2003.