RODRIGO PEREIRA MOREIRA

MAW – UMA ABORDAGEM HÍBRIDA DE MODELAGEM PARA APLICAÇÕES *WEBAPPS*

Monografia de graduação apresentada ao Departamento de Ciência da Computação da Universidade Federal de Lavras como parte das exigências do curso de Ciência da Computação para obtenção do título de Bacharel em Ciência da Computação.

LAVRAS
MINAS GERAIS - BRASIL
2007

RODRIGO PEREIRA MOREIRA

MAW – UMA ABORDAGEM HÍBRIDA DE MODELAGEM PARA APLICAÇÕES *WEBAPPS*

Monografia de graduação apresentada ao Departamento de Ciência da Computação da Universidade Federal de Lavras como parte das exigências do curso de Ciência da Computação para obtenção do título de Bacharel em Ciência da Computação.

Área de concentração:
Engenharia e Qualidade de Software

Orientador:
Prof. Dr. Heitor Augustus Xavier Costa

LAVRAS
MINAS GERAIS - BRASIL
2007

RODRIGO PEREIRA MOREIRA

MAW – UMA ABORDAGEM HÍBRIDA DE MODELAGEM PARA APLICAÇÕES *WEBAPPS*

Monografia de graduação apresentada ao Departamento de Ciência da Computação da Universidade Federal de Lavras como parte das exigências do curso de Ciência da Computação para obtenção do título de Bacharel em Ciência da Computação.

Aprovada em 07 de Agosto de 2007

_____
Msc. Ana Rubélia Mendes de Lima Resende

_____
Prof. Dr. Antônio Maria Pereira de Resende

_____
Prof. Dr. Heitor Augustus Xavier Costa
(Orientador)

LAVRAS
MINAS GERAIS - BRASIL

“Para aqueles que uma vez amaram esse mundo e passaram o tempo com seus amigos, juntem-se novamente e devotem seu tempo...”

Final Fantasy VII

Agradecimentos

À Deus, por tudo. Ao orientador Heitor Augustus pela paciência, pela ajuda nesse estudo e principalmente pelas lições de vida acadêmica. Agradeço também à minha família pelo apoio e por estarem comigo em tudo, mesmo longe e aos meus amigos por estarem sempre presentes.

Resumo

O presente trabalho descreve um estudo sobre três abordagens de modelagem conhecidas na literatura para implementação de aplicações Web: OOHDM (*Object-Oriented Hypermedia Design Method*), WebML (*Web Modeling Language*) e UWE (*UML based Web Engineering*). Após análise destas abordagens de modelagem, propôs-se uma segmentação em submodelos caracterizando-os em Modelo Conceitual, Modelo Navegacional e Modelo de Interface. Em seguida, os modelos das três abordagens de modelagem foram adequados para atender a proposta dos submodelos. Uma abordagem híbrida de modelagem para aplicações Webapps, MAW, foi proposta baseando-se nos submodelos mais adequados das três abordagens de modelagem e adaptando, quando necessário, a integração dos submodelos.

Palavras Chaves: Modelagem de Software, Aplicações Web, OOHDM, UWE, WebML.

ABSTRACT

This work describes a related study to three modeling for implementing Web applications: OOHDM (Object-Oriented Hypermedia Design Method), WebML (Web Modeling Language) e UWE (UML based Web Engineering). After analyzing these modeling, segmentation in sub-models was proposed, characterizing it in Conceptual Model, Navigational Model, and Interface Model. So, three modeling were adapted to support the sub-models proposed. A hybrid modeling to Webapps application – MAW – was proposed and it is based in the sub-model more appropriate of the three modeling, adapting the integration of the sub-models.

Words Keys: Software Modeling, Web Application, OOHDM, UWE<WebML.

Sumário

1 INTRODUÇÃO.....1

1.1 Motivação.....1

1.2 Objetivo.....3

1.3 Metodologia de Desenvolvimento.....3

1.4 Estrutura da Monografia.....3

2 ENGENHARIA DE SOFTWARE.....5

2.1 Considerações Iniciais.....5

2.2 Engenharia de Software Tradicional.....5

2.3 Engenharia de Software para Web.....9

2.4 Considerações Finais.....12

3 UML – UNIFIED MODELING LANGUAGE.....14

3.1 Considerações Iniciais.....14

3.2 Histórico da UML.....14

3.3 Definição.....15

3.4 Estruturas da UML.....16

3.4.1 Elementos Básicos.....16

3.4.2 Tipos de Relação.....18

3.4.3 Diagramas da UML.....19

3.5 Mecanismos de Extensão.....29

3.6 Considerações Finais.....30

4 ABORDAGENS DE MODELAGEM PARA O DESENVOLVIMENTO DE SOFTWARE PARA WEB.....31

4.1 Considerações Iniciais.....31

4.2 OOHDM.....31

4.2.1 Modelo Conceitual.....33

4.2.2 Projeto da Navegação.....36

4.2.2.1 Classes Navegacionais.....37

4.2.2.2 Contextos Navegacionais.....39

4.2.3 Projeto de Interface Abstrata....40
4.2.4 Implementação....42
4.3 WebML....42
4.3.1 Modelo de Dados....43
4.3.2 Modelo de Hipertexto....43
4.3.2.1 Modelo de Composição....43
4.3.2.2 Modelo Navegacional....45
4.3.3 Modelo de Apresentação....45
4.3.4 Modelo de Personalização....46
4.4 UWE....46
4.4.1 Modelo Conceitual....47
4.4.2 Modelo Navegacional....48
4.4.3 Modelo de Apresentação....52
4.5 Comparação entre as Abordagens de Moddagem....55
4.6 Considerações Finais....58

5 INTEGRAÇÃO DAS ABORDAGENS DE MODELAGEM ESTUDADAS....59

5.1 Considerações Iniciais....59
5.2 Desenvolvimento da Proposta da MAW....59
5.3 Adequação das Abordagens de Modelagem....60
5.4 Modelagem de Aplicações WebApps – MAW....61
5.4.1 Comparação e Escolha dos Submodelos das Abordagens de Modelagem....61
5.4.1.1 Modelo Conceitual....61
5.4.1.2 Modelo Navegacional....61
5.4.1.3 Modelo de Interface....62
5.4.2 Transição entre os Submodelos do MAW....62
5.4.3 Visão Geral de MAW....63
5.5 Considerações Finais....65

6 ESTUDO DE CASO – PROJETO RAD....66

6.1 Considerações Iniciais....66
6.2 Estudo de Caso....66

6.2.1 Modelo Conceitual – Subsistema de Login.....66
6.2.2 Modelo Navegacional – Subsistema Login.....67
6.2.3 Modelo de Interface – Subsistema Login.....68
6.3 Considerações Finais.....70

7 CONSIDERAÇÕES FINAIS.....71

7.1 Conclusões.....71
7.2 Contribuição.....71
7.3 Trabalhos Futuros.....72

REFERÊNCIAS BIBLIOGRÁFIAS.....73

Lista de Figuras

Figura 2.1: Ciclo de Vida do Software (Fonte: [04]) 6
Figura 2.2: Exemplo de Prototipação (Fonte: [05]) 7
Figura 2.3: Modelo Incremental (Fonte: [06]) 8
Figura 2.4: Modelo Espiral (Fonte: [07]) 9
Figura 3.1: Histórico UML (Fonte: [09]) 15
Figura 3.2: Elementos de Estrutura (Fonte: [09]) 17
Figura 3.3: Elementos de Comportamento, Agrupamento e Anotação (Fonte: [09]) 17
Figura 3.4: Tipos de Relação (Fonte: [09]) 19
Figura 3.5: Exemplo de Diagrama de Classes (Fonte: [12]) 20
Figura 3.6: Exemplo de Diagrama de Objetos (Fonte: [12]) 20
Figura 3.7: Exemplo de Diagrama de Casos de Uso (Fonte: [12]) 21
Figura 3.8: Exemplo de Diagrama de Seqüencia (Fonte: [12]) 22
Figura 3.9: Exemplo de Diagrama de Comunicação (Fonte: [12]) 23
Figura 3.10: Exemplo de Diagrama de Atividades (Fonte: [12]) 23
Figura 3.11: Exemplo de Diagrama de Estados (Fonte: [12]) 24
Figura 3.12: Exemplo de Diagrama de Componentes (Fonte: [12]) 25
Figura 3.13: Exemplo de Diagrama de Implementação (Fonte: [12]) 25
Figura 3.14: Exemplo de Diagrama de Pacote (Fonte: [12]) 26
Figura 3.15: Exemplo de Diagrama de Temporal (Fonte: [14]) 27
Figura 3.16: Exemplo de Diagrama de Tempo (Fonte: [14]) 27
Figura 3.17: Exemplo de Diagrama de Estrutura Composta (Fonte: [16]) 28
Figura 3.18: Exemplo do Diagrama Visão Geral da Interação (Fonte: [17]) 28
Figura 4.1: Exemplo de Atributo Múltiplo (Fonte: [19]) 34
Figura 4.2: Exemplo Mecanismo de Abstração – Generalização (Fonte: [19]) 34
Figura 4.3: Exemplo de Modelo Conceitual – OOHDM (Fonte: [19]) 35
Figura 4.4: Exemplo de Modelo Navegacional – OOHDM (Fonte: [19]) 37
Figura 4.5: Hierarquia de Classes Navegacionais (Fonte: [18]) 37
Figura 4.6: Exemplo de Nó (Fonte: [19]) 38
Figura 4.7: Exemplo de Elo (Fonte: [19]) 38
Figura 4.8: Elementos Gráficos do Esquema Contextual (Fonte: [19]) 40
Figura 4.9: Exemplo de ADV (Fonte: [21]) 41

Figura 4.10: Exemplo de Modelo de Dados (Fonte: [24]) 43
Figura 4.11: Exemplo de Data Unit (Fonte: [25]) 44
Figura 4.12: Exemplo de Mult Data Unit (Fonte: [25]) 44
Figura 4.13: Exemplo de Index Unit (Fonte: [25]) 44
Figura 4.14: Exemplo de Scroller Unit (Fonte: [25]) 44
Figura 4.15: Exemplo de MultiChoice Index (Fonte: [25]) 45
Figura 4.16: Exemplo Entry (Fonte: [25]) 45
Figura 4.17: Exemplo de Link Automático (Fonte: [24]) 45
Figura 4.18: Exemplo de Link de Transporte (Fonte: [24]) 45
Figura 4.19: Modelos da UWE (Fonte: [01]) 47
Figura 4.20: Modelo Conceitual (Fonte: [28]) 48
Figura 4.21: Modelo Navegacional de Espaço (Fonte: [28]) 49
Figura 4.22: Exemplo de Classe de Navegação 50
Figura 4.23: Exemplo de Índice (Fonte: [30]) 50
Figura 4.24: Exemplo de Visita Guiada (Fonte: [30]) 51
Figura 4.25: Exemplo de Interrogação (Fonte: [30]) 51
Figura 4.26: Exemplo de Menu (Fonte: [30]) 52
Figura 4.27: Modelo Navegacional (Fonte: [28]) 53
Figura 4.28: Exemplo de Classe de Apresentação (Fonte: [28]) 54
Figura 4.29: Elementos para Modelagem de Apresentação (Fonte: [30]) 55
Figura 4.30: Exemplo de Modelo de Apresentação (Fonte: [30]) 56
Figura 5.1: Diagrama de Conexão dos Submodelos 64
Figura 6.1: Modelo Conceitual Login – RAD 67
Figura 6.2: Modelo Navegacional de Espaço Login – RAD 68
Figura 6.3: Modelo Navegacional de Estrutura Login – RAD 68
Figura 6.4: Modelo de Interface Login – RAD 69
Figura 6.5: Modelo de Interface Login – RAD (Continuação) 70

Lista de Tabelas

Tabela 4.1: Características Herdadas e Criadas (Fonte: [08]) 32
Tabela 4.2: Metodologia dos Processos OOHDM (Fonte [19]) 32
Tabela 4.3: Resumo dos Modelos de Navegação 57
Tabela 4.4: Resumo dos Modelos de Interface 57
Tabela 5.1: Adequação dos Modelos 60
Tabela 5.2: Modelo MAW 64

Lista de Abreviaturas e Siglas

ADV – *Abstract Data Views*

CASE – *Computer Aided Software Engineering*

HDM – *Hypermidia Design Model*

OMG – *Object Management Group*

OOHM – *Object-Oriented Hypermedia Design Method*

RAD – Registro de Atividades de Docentes

RFP – *Request for Proposals*

FTF – *Finalization Task Force*

UML – *Unified Modeling Language*

UWE – *UML based Web Engineering*

WebApp – *Web based Application*

WebML – *Web Modeling Language*

WIS – *Web Information Systems*

1 INTRODUÇÃO

O uso de modelos e de métodos ajudam a disciplinar o processo de desenvolvimento de produtos de software, tornando possível especificar e projetar a aplicação independentemente de sua implementação e, além disso, definir uma arquitetura que facilite sua evolução e manutenção [01]. O processo de criação de um produto de software e a construção de um *website* necessitam de ferramentas que possibilitem seu desenvolvimento de um forma organizada e expressiva a qualquer entidade que faça parte desse processo.

Entretanto, esses modelos e métodos, ao serem abordados para a *Web,* demonstram diferenças em relação a visão mais adequada ao desenvolvimento de aplicações para *Web*, o que leva a um questionamento maior sobre qual desses modelos deve ser usado. Essa necessidade, apesar de aparentar ser uma decisão de foco na aplicação, apresenta uma variação de formalismo que em muitos casos dificulta a escolha da modelagem a ser aplicada.

Desta forma, esse trabalho surge com o intuito de apresentar um estudo sobre os métodos e os modelos para a modelagem de aplicações *Web*, que tenham maior ênfase na literatura, e um modelo híbrido que convirja os formalismos de modelagem.

1.1 Motivação

O avanço da *Internet* tem motivado o crescimento de uma nova geração de aplicações baseadas na *Web* (*WebApp*[1] – *Web based Application*) que combinam navegação e interatividade em um grande espaço de documentos heterogêneos. A *Web* mostra-se como um dos mais efetivos e atrativos meios de divulgação, de negociação e de disponibilização de bens e de serviços.

Organizações comerciais exploram o potencial da *Web* e dos sistemas de informação baseados na *Web* (WIS – *Web Information Systems*) para se apresentarem ao público e, ao mesmo tempo, venderem seus produtos e seus serviços com mais rapidez, ampliando o seu universo de consumidores. Além disso, devido à globalização da economia mundial, cada vez mais empresas migram seus produtos de software corporativos para plataformas baseadas na *Web* [01].

---

[1] Conjunto de denominações encontradas na literatura, como aplicação hipermídia na *Web*, hiperdocumento, hipertexto, *hipermidia*, *hiperbase*, *site* e *Website.*

Com o seu rápido crescimento em seu escopo e em extensão de seu uso, a *Web* tem afetado aspectos das vidas das pessoas. Por representar uma evolução do produto de software convencional, algumas preocupações adicionais motivaram as pesquisas relacionadas à Engenharia de Aplicações *WebApps* (*Web Engineering*), mantendo o objetivo de aplicar princípios de engenharia para desenvolver aplicações *WebApps* de qualidade [02].

Entretanto, processo de desenvolvimento de aplicações voltadas para a *Internet* diferem no processo de desenvolvimento de produtos de software comuns. Este é composto, essencialmente, pelas atividades de análise, projeto, implementação e teste, ou seja, atuais metodologias de Engenharia de Software não contêm abstração razoáveis de especificar as *WebApps*, que mudam e crescem rapidamente em requisitos, conteúdo e funcionalidade durante seu tempo de vida útil [01 e 02].

Contudo, tendo em vista facilitar e administrar a complexidade do processo de desenvolver e manter as aplicações *Web*, recomenda-se utilizar metodologias que dão suporte à construção do modelo de análise, modelo de projeto e modelo de implementação. Desta forma, o responsável pelo desenvolvimento da aplicação é conduzido a um desenvolvimento planejado, pois estes modelos têm como objetivo sistematizar a construção e a manutenção de produtos de software.

Diante deste panorama e das tendências tecnológicas, várias pesquisas são realizadas objetivando propor modelos, métodos e ferramentas formais e automatizadas para auxiliar o processo de desenvolvimento de *WebApp*.

Surge, assim, oportunidade para especificar e implementar novos modelos baseados em métodos específicos conhecidos ou propor algumas adequações destes modelos, o que remetem a motivação para estudá-los e compreendê-los com o intuito de promover esse novo foco de pesquisa.

A motivação para o estudo sobre modelagens para *Web* se deve pelo interesse de abordar como o dinamismo da *Web* interfere nesses modelos e como eles estão aptos a serem usados para o desenvolvimento de *WebApp*, pois, ao se compreender essas técnicas, as soluções, as aplicações *Web* tendem a ser formuladas com mais clareza de estrutura, permitindo melhor qualidade.

1.2 Objetivo

O objetivo desse trabalho é propor uma abordagem de modelagem híbrida, utilizando partes de três abordagens de modelagem amplamente conhecidas: WebML, UWE e OOHDM.. Para isso, foram realizados estudos comparativos destas três abordagens de modelagem para o desenvolvimento de *WebApp*.

1.3 Metodologia de Desenvolvimento

A pesquisa desenvolvida apresenta natureza básica, pois remete a um estudo comparativo e à geração de novos conhecimentos sobre modelagens de desenvolvimento de *WebApp* com foco na Engenharia de Software na *Web*. Com relação ao objetivo, possui a característica descritiva, pois busca entender o processo de construção dessas modelagens. Considerando os seus procedimentos, qualifica-se como operacional, pois realiza-se uma investigação sistemática para definição da qualidade e das deficiências da abordagens de modelagens para o desenvolvimento de *WebApp*.

O projeto foi desenvolvido a partir de levantamento bibliográfico na *Internet* e em bibliotecas, tendo sido consultados livros, monografias, artigos científicos clássicos e atuais relacionados ao tema da monografia.

Após o estudo das características dos modelos, realizou-se uma análise comparativa com o intuito de definir a qualidade e as deficiências de cada abordagem de modelagem estudada, sendo cada uma dividida em módulos para efeito de comparação. Posteriormente, apresentou-se uma proposta de abordagem de modelagem híbrida com os módulos considerados mais aderentes para a construção de aplicações *WebApps*.

1.4 Estrutura da Monografia

Este trabalho está organizado em sete capítulos.

O capítulo 2 apresenta uma visão geral sobre a Engenharia de Software Tradicional e a Engenharia de Software para *Web*.

O capítulo 3 apresenta os conceitos gerais sobre UML.

O capítulo 4 apresenta um estudo sobre três abordagens de modelagem para Engenharia de Software na *Web*: WebML, UWE e OOHDM.

O capítulo 5 apresenta a definição de uma proposta de abordagem de modelagem híbrida criada a partir do estudo realizado das abordagens anteriores, MAW – Modelagem para Aplicações *Webapps*.

O capítulo 6 apresenta um estudo de caso com o uso da modelagem MAW em uma parte de um sistema real. Para isso, foram construídos o Modelo Conceitual, o Modelo Navegacional e o Modelo de Interface de MAW.

O capítulo 7 apresenta as conclusões obtidas na realização do trabalho, suas contribuições e sugestões de trabalhos futuros.

2 ENGENHARIA DE SOFTWARE

2.1 Considerações Iniciais

As atuais metodologias de Engenharia de Software não têm habilidade para modelar *WebApps*, pois o seu ciclo de vida possui um dinamismo difícil de alcançar. Desta forma, a Engenharia *Web* é o processo utilizado para criar *WebApp* de alta qualidade.

A necessidade de métodos para o projeto e a criação de *WebApps* está relacionada ao fato do processo de desenvolvimento ser estruturado e permitir o reuso e o rastreamento das decisões tomadas, além de facilitar a construção de ferramentas automatizadas [01].

O foco deste capitulo é expor as diferenças entre a Engenharia de Software Tradicional e a Engenharia de Software Web, abordando suas principais características.

2.2 Engenharia de Software Tradicional

Visando melhorar a qualidade dos produtos de software e aumentar a produtividade no processo de desenvolvimento, surgiu a Engenharia de Software. A Engenharia de Software trata de aspectos relacionados ao estabelecimento de processos, de métodos, de técnicas, de ferramentas e de ambientes de suporte ao desenvolvimento de software.

Segundo [03], Engenharia de Software estabelece e usa sólidos princípios de engenharia para obter um produto de software economicamente viável, que seja confiável e que funcione eficientemente em máquinas reais. Entretanto, o mesmo autor afirma que cada leitor tende a ampliar essa definição, pois não ficam explícitos os aspectos de qualidade de software a serem seguidos, quais os princípios sólidos da engenharia que devem ser aplicados e como é vista a contradição entre economicamente viável com confiável.

De qualquer forma, [04] destaca que, embora várias definições tenham sido dadas à Engenharia de Software, elas reforçam a exigência da disciplina de engenharia no desenvolvimento de produtos de software. Estas definições abrangem um conjunto de três elementos fundamentais: métodos, ferramentas e procedimentos. Os métodos detalham "o que fazer" para construir o produto de software, as ferramentas proporcionam apoio automatizado ou semi-automatizado aos métodos e os procedimentos constituem o elo de ligação que mantém juntos os métodos e as suas ferramentas, possibilitando um processo de desenvolvimento claro e eficiente e visando garantir ao desenvolvedor e a seus clientes a

construção de um produto de software de qualidade.

Na década de 80, o hardware deixou de ser o item mais caro na implementação de um produto de software. Por outro lado, o custo relacionado ao software cresceu e se tornou o principal item no orçamento da computação. Isso se deve principalmente à crescente complexidade dos problemas a serem resolvidos pelo produto de software.

A chave para vencer esses problemas e essas dificuldades é a larga utilização de uma abordagem de engenharia ao desenvolvimento de produtos de software aliada a uma contínua melhoria das técnicas e das ferramentas, no intuito de melhorar a produtividade da equipe de desenvolvimento.

No processo de desenvolvimento de produtos de software, um conjunto de etapas deve ser definido, o qual é denominado paradigma da Engenharia de Software [04]; também conhecido como modelo de ciclo de vida de software (Figura 2.1).

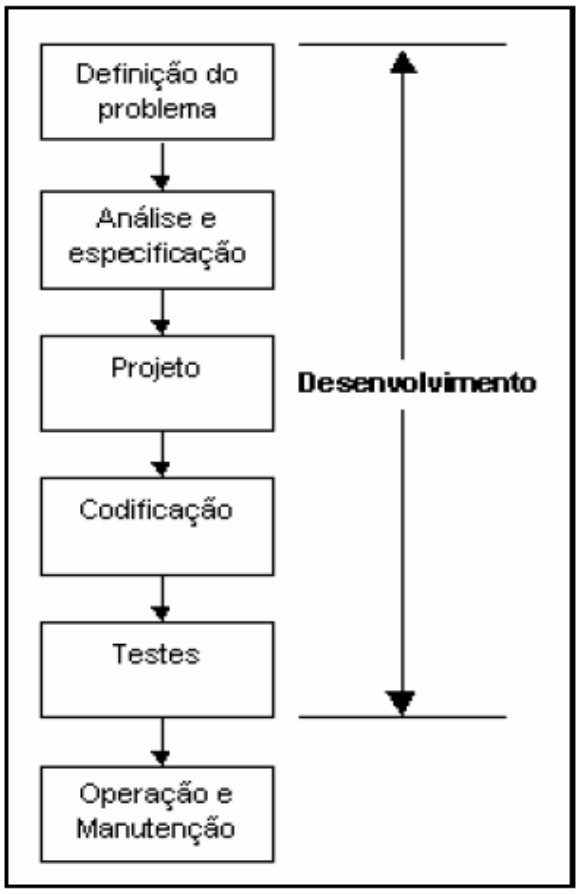


Figura 2.1: Ciclo de Vida do Software (Fonte: [04])

Alguns dos modelos que se destacam são: i) cascata; ii) modelo incremental; iii) modelo espiral; e iv) prototipação. Deve ser lembrado ainda que é possível combinar os

modelos para obter um melhor resultado.

O modelo cascata (Figura 2.1), também conhecido como Linear e Clássico, descreve um método de desenvolvimento que é linear e seqüencial. Quando uma fase de desenvolvimento é completada, o desenvolvimento prossegue para a próxima fase e não há retorno. Cada fase de desenvolvimento prossegue em uma ordem estrita, sem qualquer sobreposição ou passos iterativos [03].

A prototipação tem bons resultados, principalmente quando o cliente não tem exatidão na declaração do problema. A construção de protótipos em projetos deve ser extensamente utilizada, pois são sistemas complexos e necessitam de alto nível de colaboração do usuário no processo de desenvolvimento [04].

A Figura 2.2 mostra prototipação inicia-se com a coleta dos requisitos; em seguida, o desenvolvedor e o cliente formalizam um projeto rápido com os objetivos globais do produto a ser desenvolvido. O projeto rápido leva a construção de um protótipo que é avaliado pelo cliente e, depois, passa por uma fase de refinamento dos requisitos. O protótipo é modificado incorporando as sugestões de mudança e o cliente usa o protótipo novamente, repetindo o processo até que ele seja validado.

Figura 2.2: Exemplo de Prototipação (Fonte: [05])

O modelo incremental, segundo [03], combina elementos do modelo cascata sendo aplicado de maneira interativa. O modelo de processo incremental é interativo igual à prototipagem, mas, diferente da prototipagem, o incremental tem como objetivo apresentar

um produto operacional a cada incremento realizado.

A Figura 2.3 demonstra que o produto de software é desenvolvido em incrementos e que em cada um deles é usado um modelo em cascata, ou seja, pode ser visto como a aplicação do modelo em cascata várias vezes.

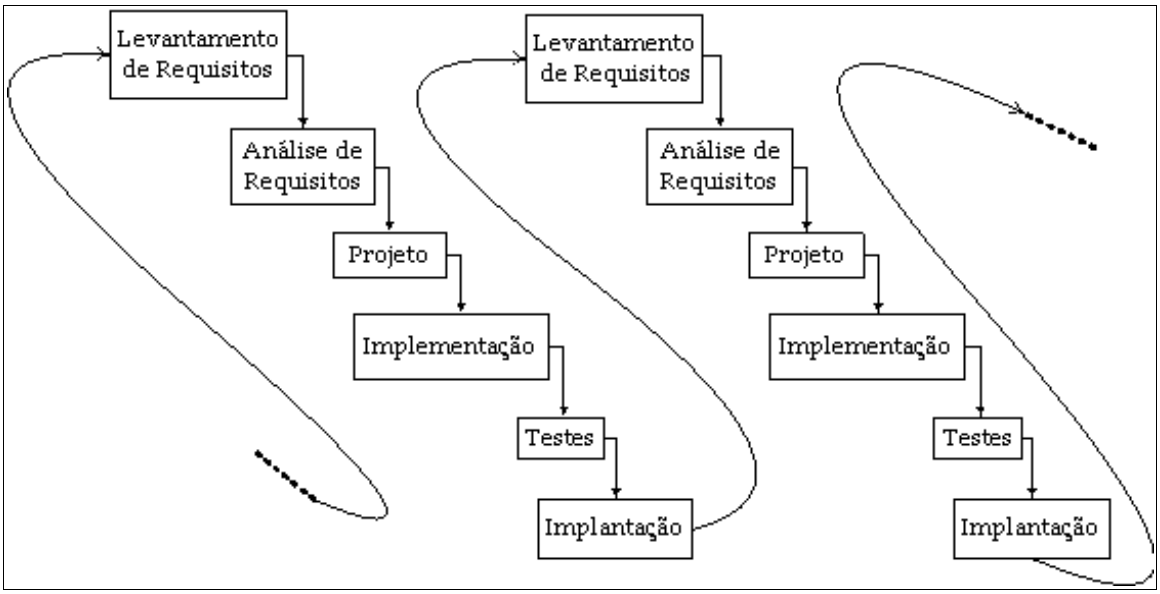


Figura 2.3: Modelo Incremental (Fonte: [06])

O modelo espiral (Figura 2.4) é baseado no princípio do desenvolvimento incremental, no qual novas funções são adicionadas a cada ciclo de desenvolvimento. Análise, especificação, projeto, implementação e homologação são repetidas a cada ciclo, gerando uma nova versão do produto de software e permitindo um *feedback* mais imediato do usuário [04]. Com cada iteração ao redor da espiral (iniciando-se ao centro e avançando para fora), versões progressivamente mais completas do produto de software são construídas. Durante as interações ao redor da espiral, os objetivos, as alternativas e as restrições são definidos e os riscos são identificados e analisados.

Existem outros modelos citados por [03] os quais são:

- Modelo Incremental;
- Modelo RAD;
- Modelo de Desenvolvimento Concorrente;
- Modelo de Métodos Formais.

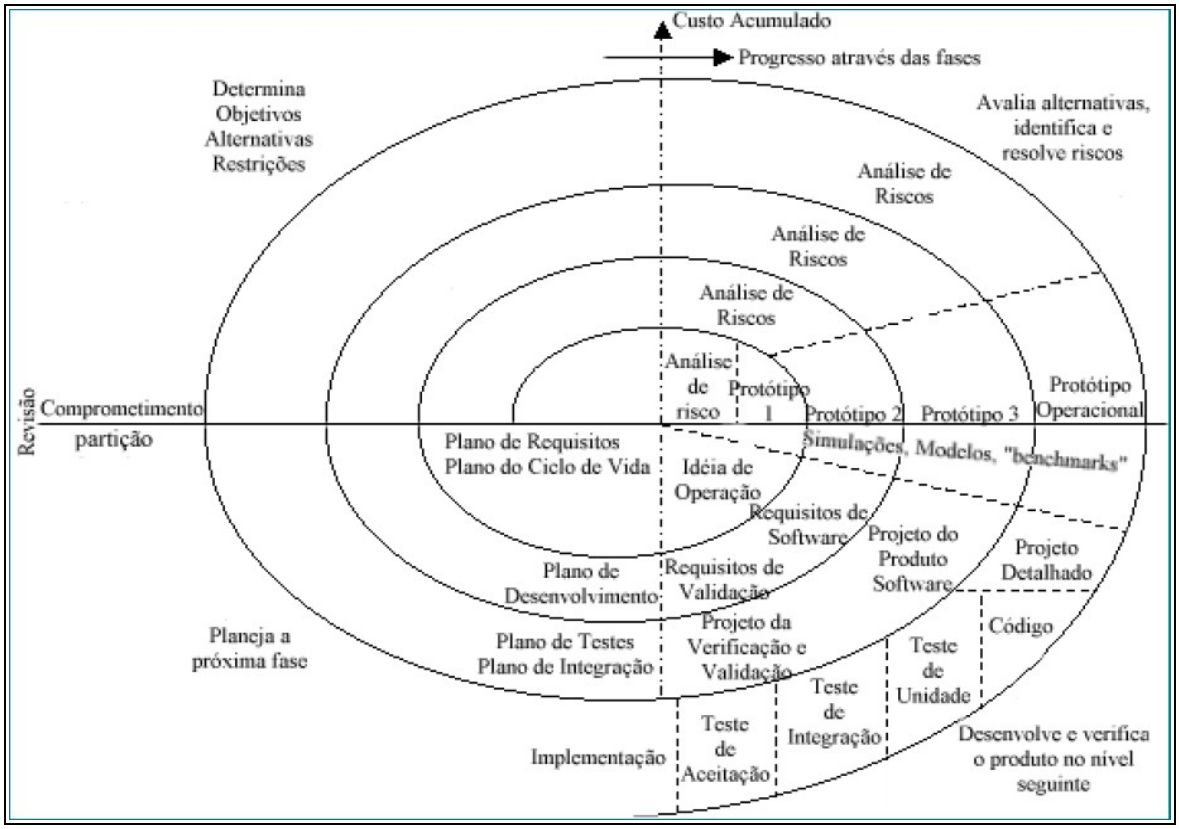


Figura 2.4: Modelo Espiral (Fonte: [07])

Independentemente do paradigma a ser utilizado, três fases genéricas dividem o processo de desenvolvimento [04]:

- Definição: focaliza o "o quê" (análise do sistema, planejamento do projeto de software e análise de requisitos);
- Desenvolvimento: focaliza o "como" (projeto de software, codificação e realização de testes do software);
- Manutenção: focaliza as "mudanças" (correção, adaptação e melhoramento funcional).

Cabe destacar a existência de diversas metodologias de Engenharia de Software que criaram novos paradigmas, combinando e aproveitando os melhores conceitos de outras metodologias. Nesse ponto, deve-se ressaltar o *Unified Process* (UP – Processo Unificado) como um dos principais processos de desenvolvimento de software utilizados em conjunto com a linguagem de notação UML (*Unified Modeling Language*).

2.3 Engenharia de Software para *Web*

Quando se fala em Engenharia de Software, deve-se ter claro que ela destina-se a

aplicar métodos e tecnologias da engenharia para planejar, especificar, desenhar, implementar, validar, testar, medir, manter e melhorar o produto de software.

Segundo [01], o avanço da Engenharia *Web* deve promover um enfoque sistemático, disciplinado e quantificável para o desenvolvimento e a evolução de *WebApps*, com alta qualidade e a um custo efetivo.

A Engenharia de Software para *WebApps* é importante, pois auxilia no processo de desenvolvimento nas áreas envolvidas no projeto: hipermídia, banco de dados, *design*, artes, comunicação, computação gráfica, linguagens computacionais, etc.

Desde a Crise de Software, ocorrida na década de 70, os programadores buscam soluções que facilitem o processo de criação e de manutenção de produtos de software complexos. Com o surgimento de aplicações *Web* e a sua popularização, vários paradigmas de programação tiveram que ser mudados. Isso não afirma que a programação para a construção de *WebApps* seja muito diferente da programação tradicional, porém devem ser tomados certos cuidados ao planejar tais produtos de software [08].

As principais diferenças entre aplicações *Web* e as tradicionais referem-se a questões sobre navegação, organização da interface e implementação. Algumas dessas características são [08]:

- A maioria das aplicações *Web* é orientada à documentação, contendo páginas *Web* estáticas e dinâmicas;
- As aplicações *Web* são focadas na aparência, exigindo criatividade na apresentação visual e a incorporação de recursos multimídia;
- A maioria das aplicações *Web* é dirigida a conteúdo, portanto desenvolvê-las inclui desenvolver o conteúdo a ser apresentado;
- Um *site Web*, geralmente, atende a uma multiplicidade de perfis de usuários, precisando satisfazer às necessidades de usuários com diferentes habilidades e capacidades, aumentando a complexidade da interação homem-máquina, a interface com o usuário e a apresentação das informações;
- A natureza e as características da mídia da *Web* não estão bem entendidas quanto a mídia dos produto de software tradicionais;
- Em geral, um projeto *Web* exige maior multiplicidade de tipos e níveis de habilidades (e papéis) do que os participantes de um projeto de software tradicional;
- A maioria das aplicações *Web* precisa ser desenvolvida em um curto espaço de tempo

(*Web Time*), tornando difícil a adoção do mesmo nível de rigor no planejamento e no teste utilizados em produtos de software tradicionais;

- As aplicações *Web* diferem de sistemas tradicionais quanto ao ambiente em que são disponibilizados, pois o meio onde se executam as aplicações baseadas na *Web* é mais imprevisível do que o meio onde se rodam as aplicações de produtos de software tradicionais.

O principal objetivo da Engenharia de Software para *Web* é desenvolver *WebApps* com eficiência e qualidade, devido ao fato que, na maioria das vezes, os aplicativos para *Web* são desenvolvidos partindo-se de pequenas aplicações a nível de departamento e migrando-se para aplicações que atinge toda a empresa, fazendo com que eles sejam difíceis de manter [08].

Segundo [02], a qualidade de aplicações *Web* tem sido avaliada, geralmente, de maneira *ad-hoc*, baseada primariamente em senso comum, intuição e conhecimento de desenvolvedores. No entanto, para atender aspectos relativos ao desenvolvimento *Web* e garantir a qualidade de processos específicos, faz-se necessário definir novas atividades relacionadas à qualidade de processos em Engenharia *Web*.

Para conseguir desenvolver atividades e técnicas de garantia da qualidade que apóiem efetivamente processos de desenvolvimento de aplicações *Web*, é necessário caracterizar esses processos, a fim de conhecer as atividades existentes e os artefatos específicos por eles produzidos.

Como na Engenharia de Software, os objetivos para o desenvolvimento de *WebApp* são os mesmos, pode-se citar [08]:

- Evitar o desenvolvimento *ad-hoc* (falta de modelos de processo);
- Garantir o acompanhamento das evoluções tecnológicas;
- Garantir o cumprimento de prazos através de estimativas de métricas precisas;
- Gerenciar equipes interdisciplinares, facilitando a comunicação entre as diferentes áreas envolvidas no processo;
- Diminuir custos de criação e de manutenção;
- Documentar as fases do projeto;
- Proporcionar navegação entre as informações de forma intuitiva e organizada.

Processos de desenvolvimento para aplicações *Web* devem produzir representações

para projeto de aspectos de aplicações tradicionais, como estrutura e funcionalidade, e para aspectos orientados para *Web*, como navegação e apresentação (com recursos *Web*). Mendes e Mosley *apud* [02] discutem as principais diferenças entre desenvolvimento de produtos de software convencionais e para *Web*. Várias metodologias voltadas para o desenvolvimento de aplicações *Web* têm sido propostas, por exemplo OOHDM, UWE e WebML.

As tecnologias utilizadas na construção de produtos de *software* tradicionais aplicando técnicas de Engenharia de Software são mais estáveis. Isto decorre do fato que o desenvolvimento deste tipo de produto de software ocorre a muito mais tempo; além disso, existem excelentes ferramentas automatizadas disponíveis para análise, projeto, implementação e testes. No desenvolvimento de *WebApp*, as tecnologias estão em constante evolução, embora existam diversas ferramentas automatizadas a nível de implementação e poucas para análise, projetos e outras atividades do desenvolvimento [08].

2.4 Considerações Finais

A Engenharia de Software para desenvolvimento Tradicional e para o desenvolvimento de *WebApps* preocupa-se em facilitar o desenvolvimento e a manutenção de produtos de software complexos, utilizando métodos, técnicas, ferramentas CASE e modelos específicos. Este ferramental visa gerenciar a qualidade de produção, prazos e pessoas envolvidas no projeto. Mesmo que a Engenharia de Software para o desenvolvimento Tradicional e para o desenvolvimento de *WebApps* tenha o mesmo objetivo e a Engenharia de Software para o desenvolvimento de *WebApp* utilize princípios da Engenharia de Software Tradicional, não é possível utilizar as mesmas técnicas para ambas, ou seja, a Engenharia de Software para o desenvolvimento de *WebApp* não é uma cópia da Engenharia de Software Tradicional, como muitos profissionais pensam [08].

Uma das principais diferenças entre as duas formas de desenvolvimento é o ciclo de vida do produto de software, a evolução do produto de software tradicional é muitas vezes menor que a evolução de um *WebApp*, devido ao caráter tecnológico e informacional de um *WebApp* que evolui de maneira constante. O que influi realmente no ciclo de vida de um *WebApp* é o fato de seu desenvolvimento ser uma mistura de *designer* e de programação, entre *marketing* e computação, entre relações internas e externas; enfim, a multidisciplinaridade envolvida na Engenharia de Software para *WebApp* é extremamente envolvente e desafiadora [08].

A abrangência de uma aplicação convencional possui um público muitas vezes definido de usuários, público esse que pode ser controlado, enquanto, no desenvolvimento de um *WebApp,* a classe de usuários pode ser muito maior, as vezes sendo impossível saber exatamente o perfil dos usuários que interagem com o produto de software construído [08].

3 UML – *UNIFIED MODELING LANGUAGE*

3.1 Considerações Iniciais

Ao longo dos anos, a evolução das aplicações em produtos de software tornaram-se mais complexas, necessitando de um padrão que possibilitasse a documentação durante a sua construção. A solução para esse problema resultou na criação de uma linguagem padrão chamada UML (*Unified Modeling Language)* sendo esta apresentada nesse capitulo.

3.2 Histórico da UML

As linguagens de modelagem orientadas a objetos surgiram entre a metade da década de 70 e o final da década de 80, à medida que o pessoal envolvido com metodologia, diante de um novo gênero de linguagens de programação orientadas a objeto e de aplicações cada vez mais complexas, começou a experimentar métodos alternativos de análise e de projeto. A quantidade de métodos orientados a objetos aumentou de pouco mais de 10 para mais de 50 durante o período de 1989 a 1994. Muitos usuários desses métodos tiveram dificuldades para encontrar uma linguagem de modelagem capaz de atender inteiramente as suas necessidades. A resposta para essa questão foi a UML (Figura 3.1) [09].

A criação da UML iniciou oficialmente em outubro de 1994, quando James Rumbaugh se juntou a Grady Booch na Rational. O foco inicial do projeto era a unificação do Método de Booch e OMT (*Object Modeling Technique*) [10]. O esboço da versão 0.8 do Método Unificado foi lançado em outubro de 1995. Mais ou menos na mesma época, Ivar Jacobson se associou à Rational com a finalidade de incorporar OOSE (*Object-Oriented Software Engineering*) no escopo inicial da versão 0.8, resultando o lançamento da versão 0.9 da UML em junho de 1996 [10]. Muitas empresas ficaram interessadas e um consórcio foi criado com várias empresas interessadas em dedicar recursos com o propósito de trabalhar em uma definição mais forte e completa da UML.

As empresas que trabalharam e contribuíam para a definição da UML 1.0 foram: Digital Equipment Corporationm Hewlett-Packard, I-Logix, Intel-Licorp, IBM, ICON Computing, MCI Systemhouse, Microsoft, Oracle, Rational, Texas Instruments e Unisys. O trabalho resultou em uma linguagem de modelagem bem definida, expressiva, poderosa e que poderia ser aplicada em vários tipos de problemas. A UML foi apresentada para a OMG

(*Object Management Group*) em janeiro de 1997, em resposta à sua solicitação de propostas para uma linguagem padrão de modelagem [09].

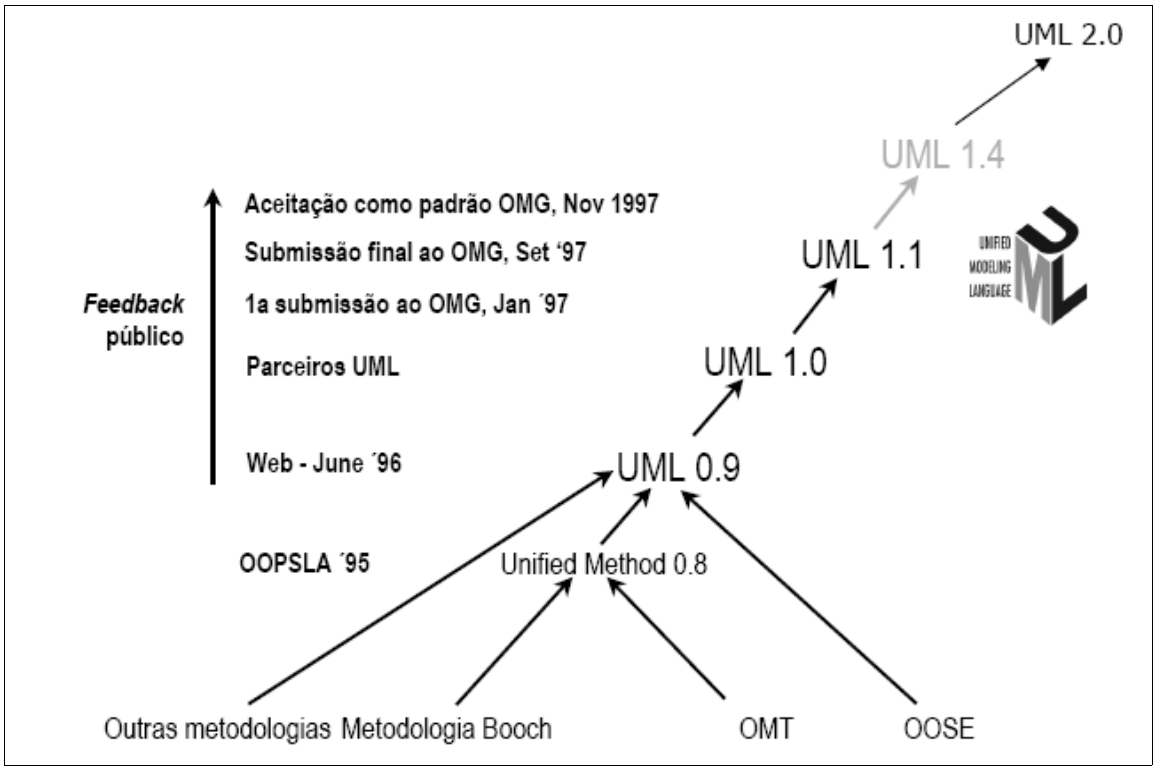


Figura 3.1: Histórico UML (Fonte: [09])

OMG lançou uma RFP (*Request for Proposals*) para que outras empresas pudessem contribuir com a evolução da UML, daí se chegou a versão 1.1. Após alcançar esta versão, OMG passou a adotá-la como padrão e responsabilizar-se pelas revisões através da *Revision Task Force* (RTF), que produziu várias versões com destaque para 1.4. Essas revisões são controladas para não provocar grande mudança no escopo original. Realmente, foi isso que aconteceu, se observar as diferenças entre as versões, pode-se ver que de uma versão para a outra não houve grande impacto, o que facilitou sua disseminação pelo mundo [10]. Entre 2000 a 2003, a versão 2.0 foi produzida e atualizada sendo revisada em 2004 pela FTF (*Finalization Task Force*) e a versão oficial da UML 2.0 foi adotada pelo OMG em2005.

3.3 Definição

A UML é uma linguagem padrão que permite modelar um produto de software orientado a objetos, bem como dotar o mercado mundial de orientação a objetos de uma linguagem única de modelagem, permitindo a troca de modelos de forma natural entre os

desenvolvedores de produtos de software. Com a UML, é possível: i) descrever eficazmente requisitos de um produto de software; ii) caracterizar a arquitetura lógica e física de um produto de software; iii) focalizar na arquitetura ao invés da implementação; e iv) direcionar programadores, aumentando a produtividade e diminuindo os riscos [10 e 12].

A UML suporta as cinco fases de desenvolvimento de produtos de software: análise de requisitos, análise, projeto, implementação e testes. Estas fases não necessariamente devem ser executadas em ordem seqüencial [12].

3.4 Estruturas da UML

A estrutura de conceitos da UML é razoavelmente abrangente consistindo em um conjunto variado de notações, as quais podem ser aplicados em diferentes domínios de problemas e a diferentes níveis de abstração [10].

O vocabulário da UML abrange três tipos de blocos de construção: i) elementos básicos (itens); ii) relacionamentos; e iii) diagramas [12].

3.4.1 Elementos Básicos

Os elementos básicos são organizados segundo a sua funcionalidade ou responsabilidade. Assim, há elementos estrutural, comportamental, de agrupamento e de anotação [10].

A Figura 3.2 ilustra o conjunto dos principais elementos de estrutura [10 e 12], a saber:

- Classe descreve conjunto de objetos que compartilham os mesmos atributos, as mesmas operações, os mesmos relacionamentos e a mesma semântica e pode implementar uma ou mais interfaces;
- Classe ativa tem um ou mais processos ou *threads* e podem iniciar a atividade de controle, ou seja, modela elementos que executam algum tipo de computação;
- Interface é uma coleção de operações que especificam serviços de uma classe ou de um componente;
- Colaboração define interações e são sociedades de papéis e outros elementos que funcionam em conjunto para proporcionar um comportamento cooperativo superior à soma dos outros elementos;

- Caso de uso descreve um conjunto de seqüência de ações realizadas pelo produto de software que proporcionam resultados observáveis de valor para um determinado ator;
- Ator representa um conjunto coerente de papéis que os usuários de casos de uso desempenham quando interagem com a aplicação. Tipicamente, um ator representa um papel que um ser humano, um dispositivo de hardware ou outro produto de software desempenha com o programa principal;
- Componente é a parte física e substituível de um produto de software, que proporciona a realização de um conjunto de interfaces;
- Nó é um elemento físico existente em tempo de execução que representa um recurso computacional, geralmente, com alguma memória e, freqüentemente, capacidade de processamento.

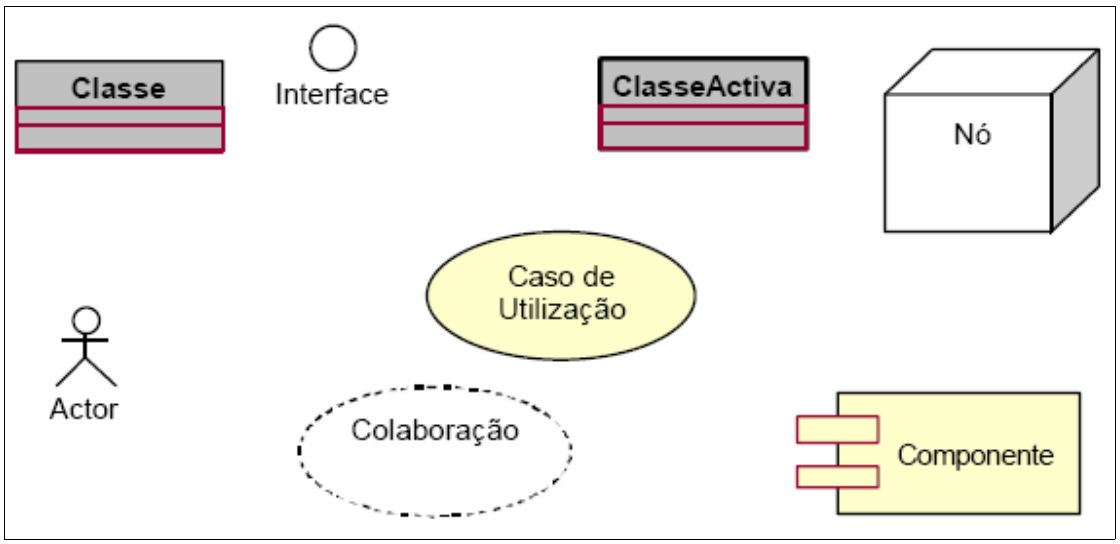


Figura 3.2: Elementos de Estrutura (Fonte: [09])

A Figura 3.3 ilustra outros elementos básicos da UML: i) de comportamento (estados e mensagens); ii) de agrupamento (pacotes); eiii) de anotação (notas).

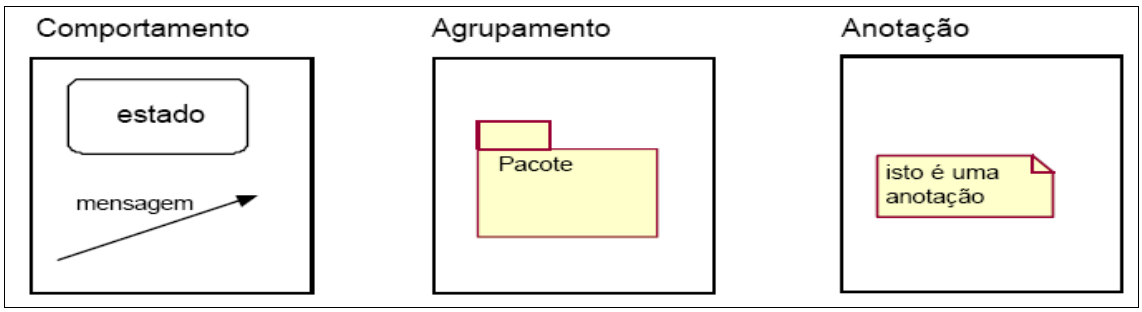


Figura 3.3: Elementos de Comportamento, Agrupamento e Anotação (Fonte: [09])

Estes elementos têm a finalidade de prover a simplificação de produtos de software mais abrangente e/ou atribuir conceitos extras aos diagramas da UML [10 e 12]:

- Elementos de comportamento são partes dinâmicas dos diagramas da UML, representam comportamentos no tempo e no espaço. Existem dois tipos de itens comportamentais:
    1. Interação. É um comportamento que abrange um conjunto de mensagens trocadas entre um conjunto de objetos em determinado contexto para realizar os propósitos específicos;
    2. Máquina de estado. É um comportamento que especifica as seqüências de estados pelas quais objetos ou interações passam durante sua existência em resposta a eventos, bem como suas respostas a esse evento.
- Elementos de agrupamento são partes organizacionais de um diagrama da UML. Eles são os blocos em que os modelos podem ser decompostos. Existe apenas um item de agrupamento chamado pacote, que consiste em um mecanismo de propósito geral para a organização de elementos em grupos;
- Elementos de anotação são partes explicativas dos diagramas da UML. Eles são elementos para comentário, descrição e esclarecimento sobre qualquer elemento do modelo. Existe apenas um item de anotação chamado nota, que é uma representação gráfica de restrições e comentário anexado a um elemento ou a um conjunto de elementos.

3.4.2 Tipos de Relação

As relações são conceitos gerais que apresentam uma sintaxe e uma semântica bem definidas e permitem o estabelecimento de interdependências entre os elementos básicos. A Figura 3.4 ilustra os principais tipos de relações da UML[10 e 12], a saber:

- Dependência é um relacionamento de uso que determina as modificações na especialização de um item, mas não necessariamente o inverso;
- Generalização é o relacionamento entre itens gerais e tipos mais específicos;
- Associação é um relacionamento estrutural que especifica objetos de um item conectado a objeto de outro item;
- Realização é um relacionamento semântico entre classificadores, no qual um dos classificadores especifica um contrato cujo cumprimento é assegurado pelo outro classificador;

- Transição de estado é um relacionamento que direciona a mudança de estado entre objetos.

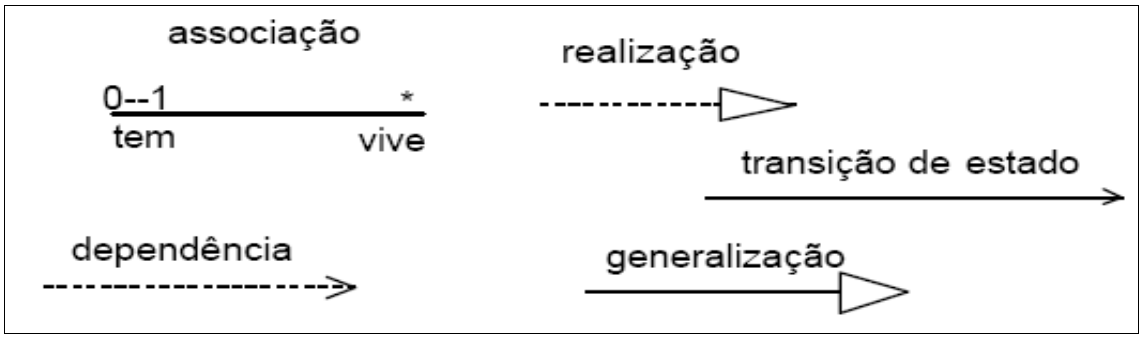


Figura 3.4: Tipos de Relação (Fonte: [09])

3.4.3 Diagramas da UML

Os diagramas são conceitos que traduzem a possibilidade de agrupar elementos básicos e suas relações de uma forma lógica ou de uma forma estrutural. Ou seja, são desenhados para permitir a visualização de um produto de software sobre diferentes perspectivas. Há diferentes tipos de diagramas em UML e, em cada um deles, é usado um subconjunto dos elementos básicos e diferentes tipos de relações que tenha sentido existir. Os diagramas são [09 e 10]:

- Diagrama de Classes descreve a estrutura estática de um produto de software, em particular as entidades existentes, as suas estruturas internas e as relações entre si, ou seja, exibe um conjunto de classes, interfaces e colaborações, bem como seus relacionamentos (Figura 3.5). Além disso, este diagrama promove a especificação, a visualização e a documentação de modelos estruturais e possibilita a construção de produtos de software executáveis por intermédio da engenharia de produção bem como a engenharia reversa;
- Diagrama de Objetos é uma variação do Diagrama de Classes e utiliza quase a mesma notação. A diferença é o Diagrama de Objetos mostrar os objetos que foram instanciados das classes (Figura 3.6). O Diagrama de Objetos é como se fosse o perfil do produto de software em um certo momento de sua execução;

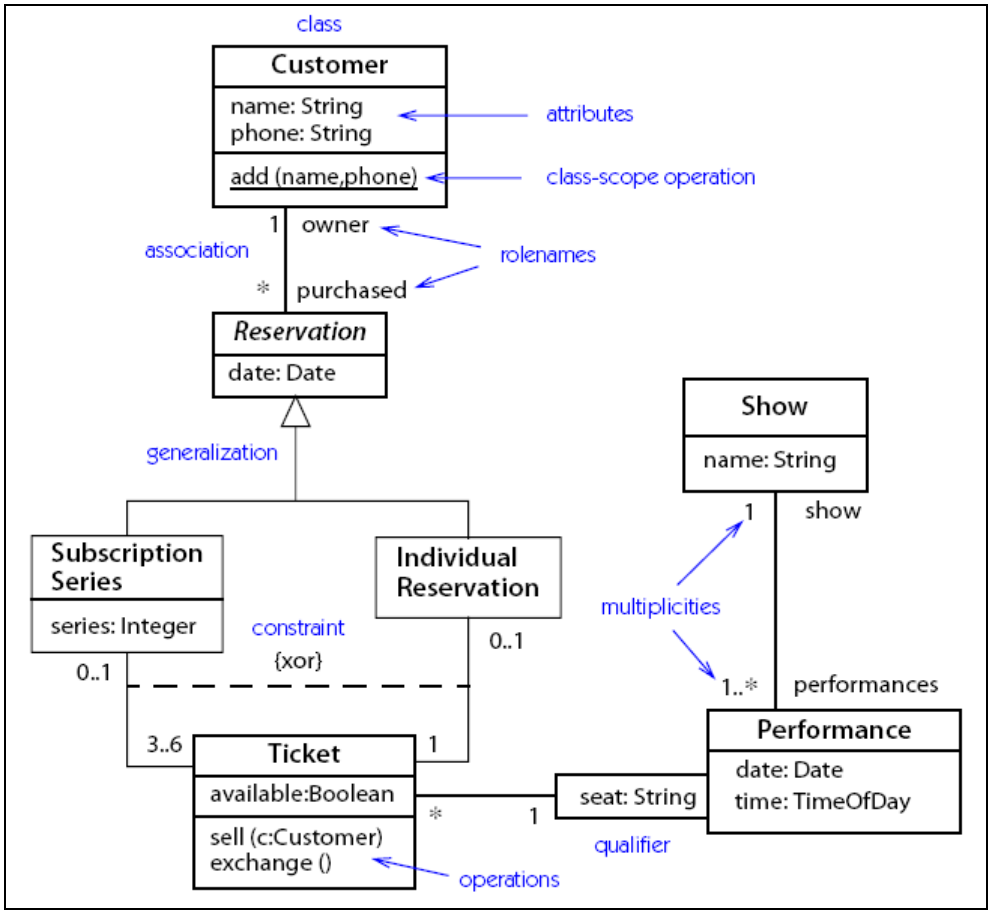


Figura 3.5: Exemplo de Diagrama de Classes (Fonte: [12])

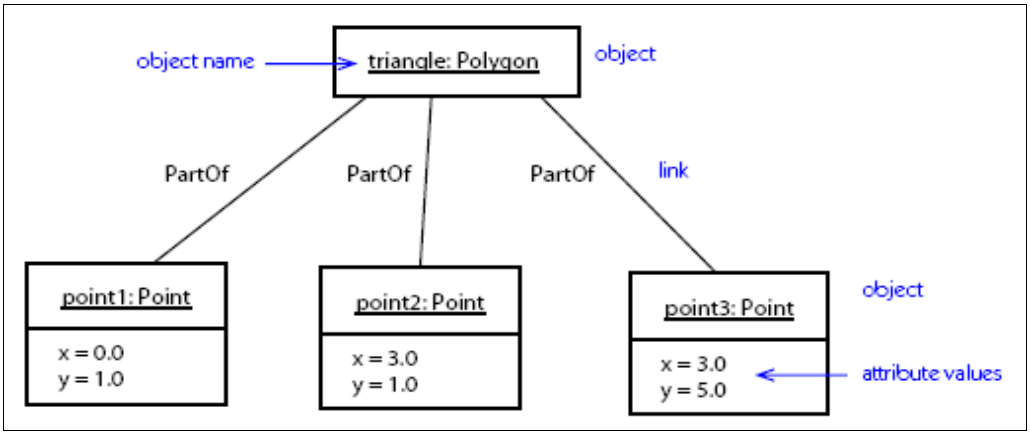


Figura 3.6: Exemplo de Diagrama de Objetos (Fonte: [12])

- Diagramas de Casos de Uso descreve a relação entre os atores e os casos de uso do produto de software (Figura 3.7) e é utilizado para os aspectos dinâmicos do produto de software. Ele tem o objetivo de documentar, visualizar e especificar o comportamento de um elemento. Este diagrama apresenta uma visão global e de alto

nível do produto de software, sendo fundamental a definição correta da sua fronteira, pois esse diagrama faz com que o sistema, o subsistemas e as classes fiquem acessíveis e compreensíveis e apresente uma visão externa sobre como esses elementos podem ser utilizados [09]. Este diagrama é utilizado, preferencialmente, na fase de especificação de requisitos e na modelagem de processos de negócio, pois são importantes para a organização e a modelagem de comportamentos do produto de software;

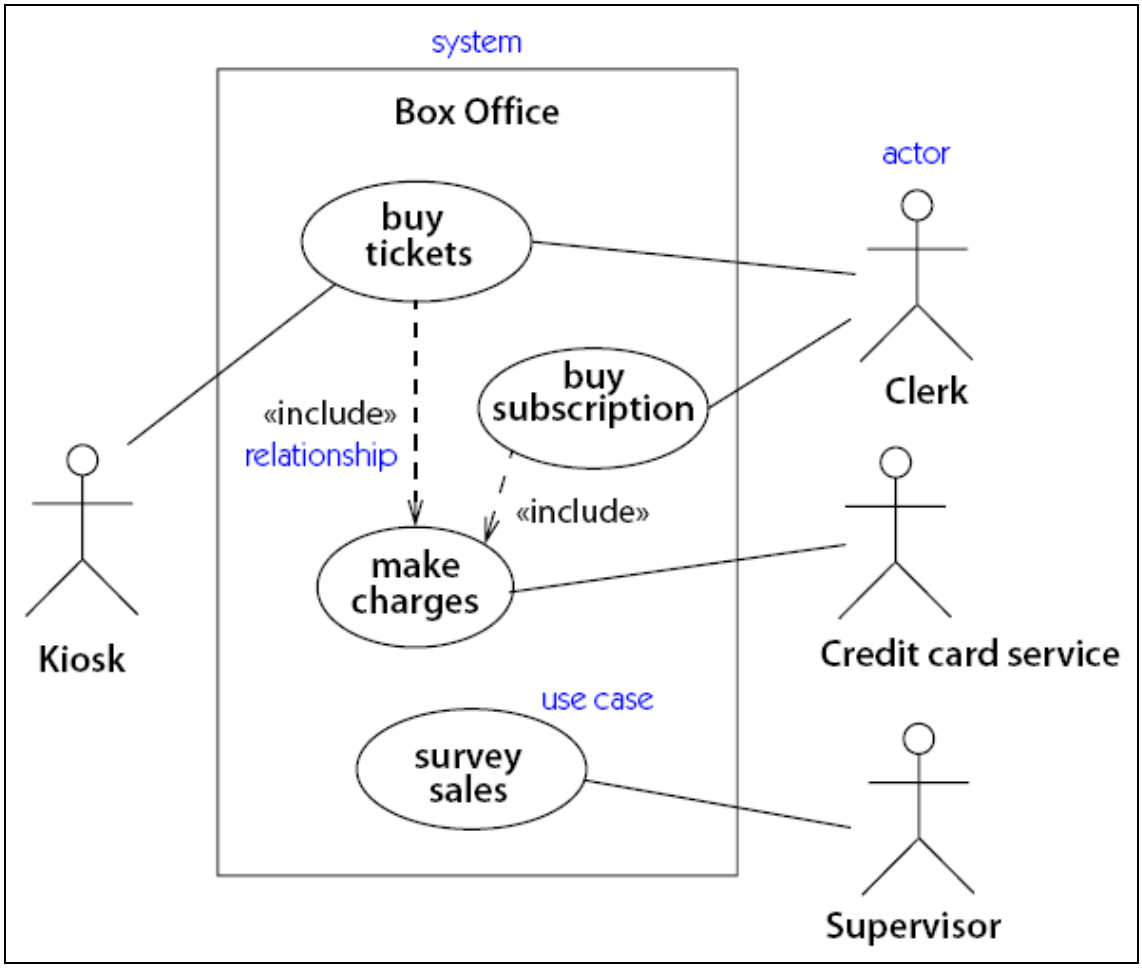


Figura 3.7: Exemplo de Diagrama de Casos de Uso (Fonte: [12])

- Diagrama de Seqüência ilustra interações entre objetos em um determinado período de tempo, sendo sua ênfase na ordenação temporal das mensagens (Figura 3.8). Em particular, os objetos são representados pelas suas linhas de vida e interagem através de troca de mensagens ao longo de um determinado período de tempo. O Diagrama de Seqüência tem duas características estruturais: i) a presença da linha de vida (*lifeline*) do objeto; e ii) a existência de foco de controle;

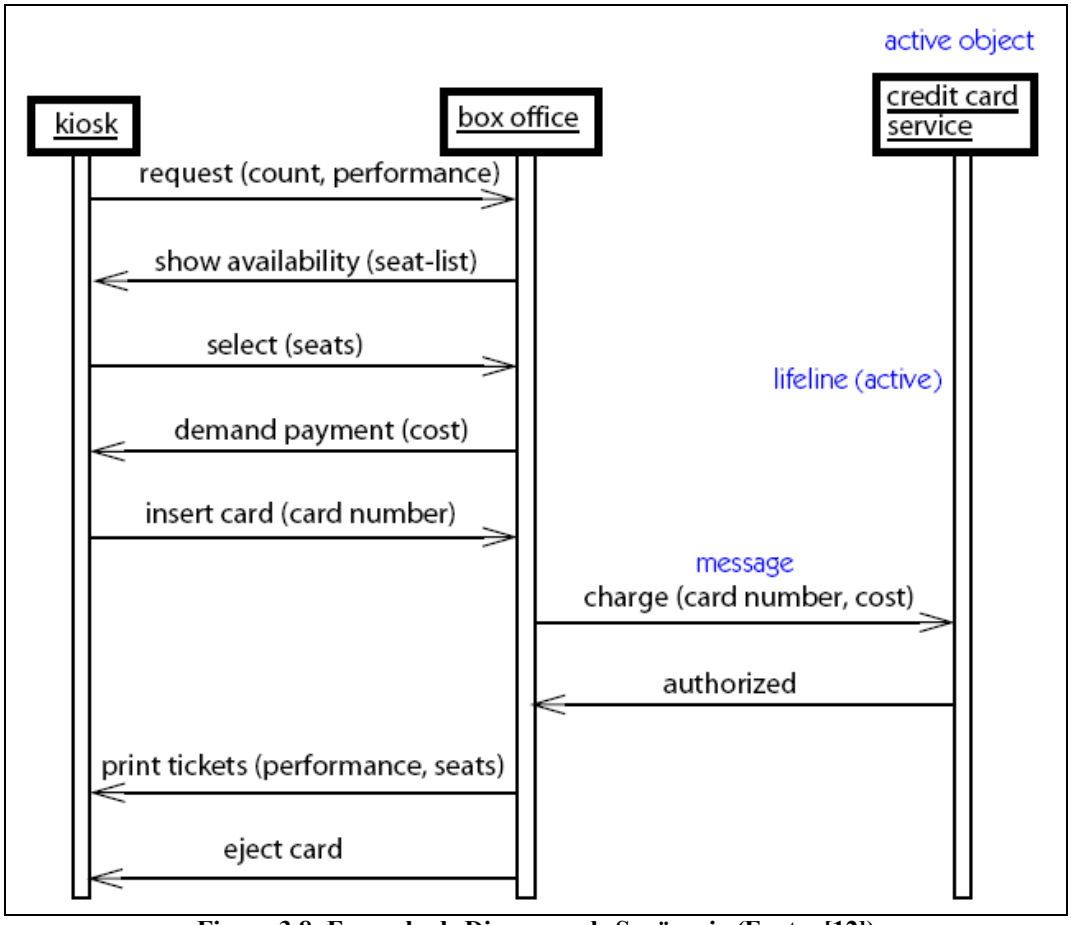


Figura 3.8: Exemplo de Diagrama de Seqüencia (Fonte: [12])

- Diagrama de Comunicação ilustra interações entre objetos com ênfase para a representação das ligações entre objetos (Figura 3.9). Como os Diagramas de Colaboração não mostram o tempo como um elemento explícito (tal como acontece no Diagrama de Seqüência), a seqüência de mensagens e de atividades concorrentes é determinada usando números seqüenciais com diferentes níveis hierárquicos [10];
- Diagrama de Atividades mostra o fluxo de controle de uma atividade para outra, sendo uma atividade considerada como uma execução em andamento não atômica em uma máquina de estados (Figura 3.10). Este diagrama abrange a visão dinâmica do produto de software, é importante para a modelagem da sua função e dá ênfase ao fluxo de controle entre objetos;

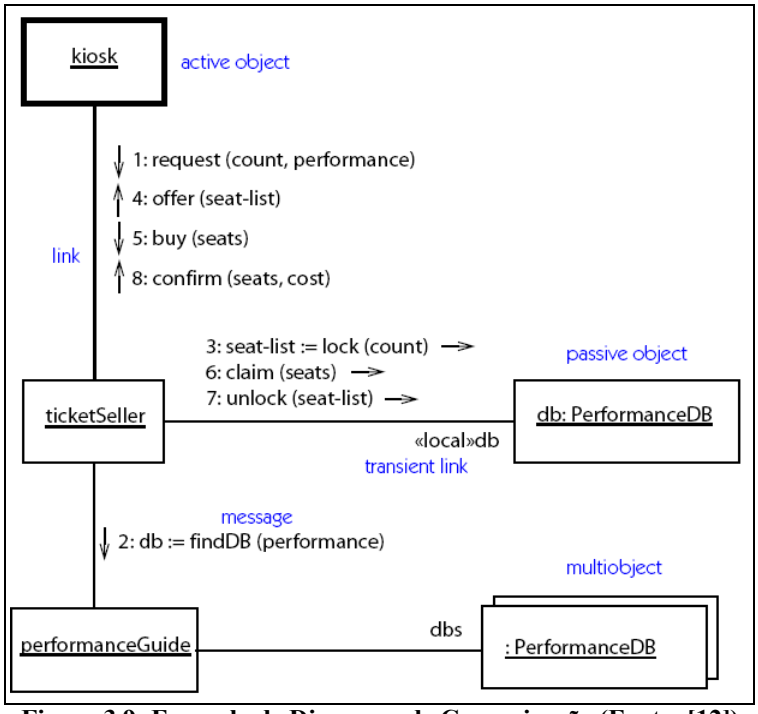


Figura 3.9: Exemplo de Diagrama de Comunicação (Fonte: [12])

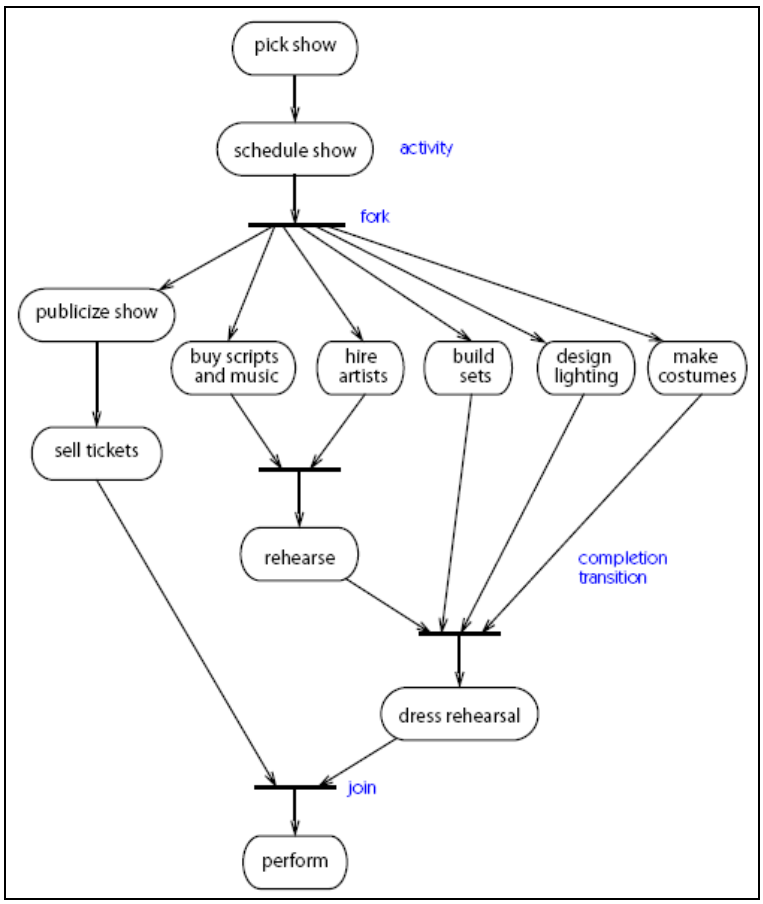


Figura 3.10: Exemplo de Diagrama de Atividades (Fonte: [12])

- Diagrama de Estados exibe a dinâmica interna de uma classe (Figura 3.11). Apenas os eventos e os estados de uma classe são apresentados no diagrama. Entende-se por eventos os fatos que ocorrem na classe, provocados por elementos externos ou internos como condições internas da classe que provocam uma troca de estado. Uma classe pode ter vários estados, caracterizados por situações em que a classe se encontra. O Diagrama de Estados pode possuir estados especiais como o estado inicial e o estado final e outros estados de controle internos;

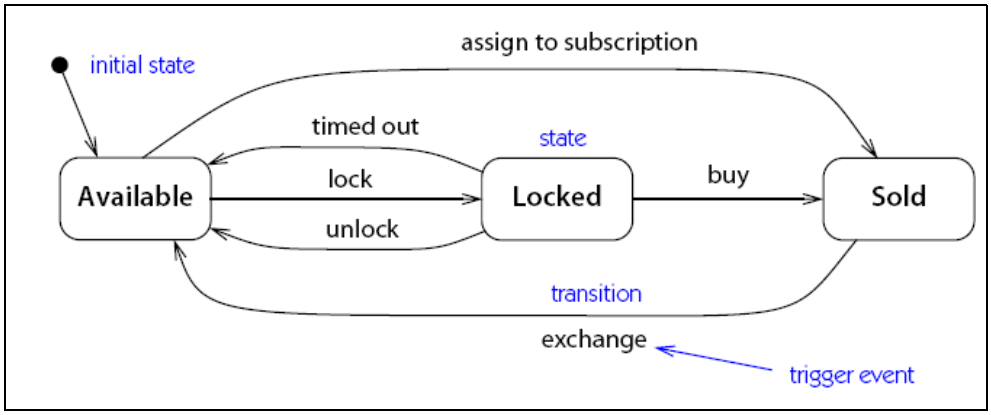


Figura 3.11: Exemplo de Diagrama de Estados (Fonte: [12])

- Diagrama de Componentes mostra os elementos reutilizáveis de produtos de software e sua interdependência, sendo um componente formado por um conjunto de classes que se encontram implementadas nele e as classes, que ele possui, dependem funcionalmente das classes de outro componente, cuja dependência é exibida pelo diagrama (Figura 3.12). O Diagrama de Componentes permite demonstrar a configuração de um produto de software exibindo, graficamente, a dependência entre os diversos arquivos que o compõem;
- Diagrama de Implementação descreve a configuração de elementos de suporte de processamento e de componentes de software, os processos e os objetos existentes nesses elementos (Figura 3.13) [10];
- Diagrama de Pacote é utilizado quando Diagrama de Classes for muito denso (possui muitas informações) para representar o produto de software. Assim, é conveniente utilizar um elemento para organizar em subsistemas. Para isso, utiliza-se os Diagramas de Pacote (Figura 3.14), que representa um grupo de classes (ou outros elementos) que se relacionam com outros pacotes através de uma relação de dependência. Um Diagrama de Pacotes pode ser utilizado em qualquer fase do processo de modelagem e visa organizar os modelos [13];

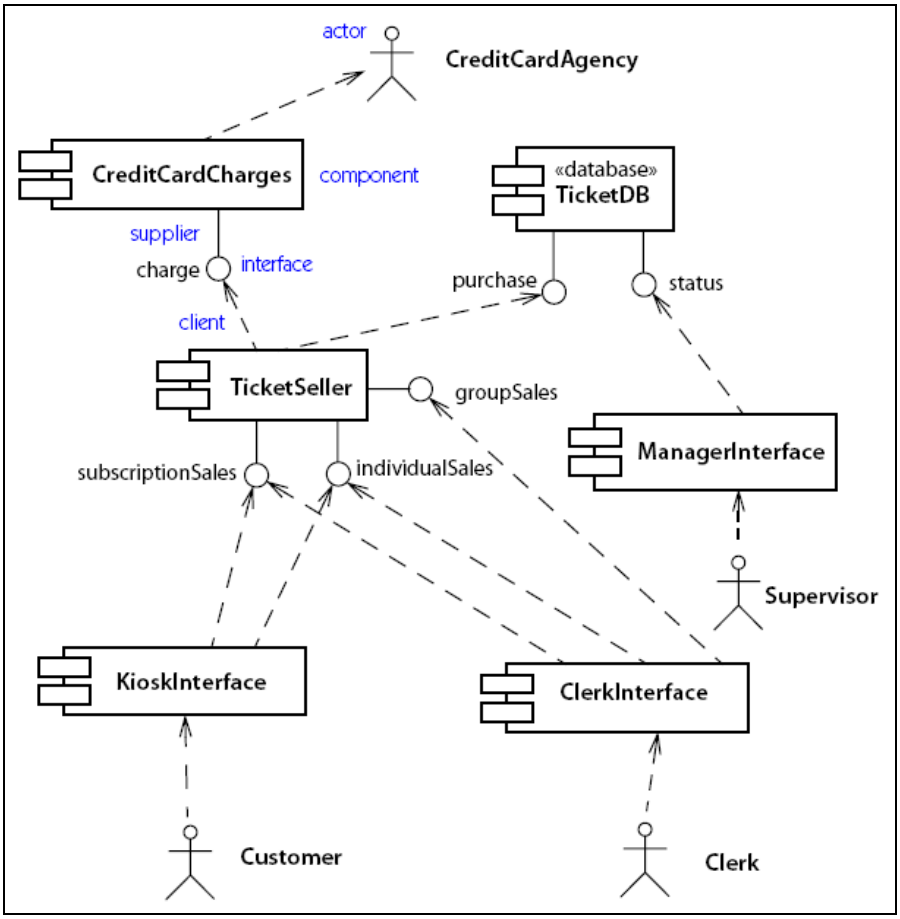


Figura 3.12: Exemplo de Diagrama de Componentes (Fonte: [12])

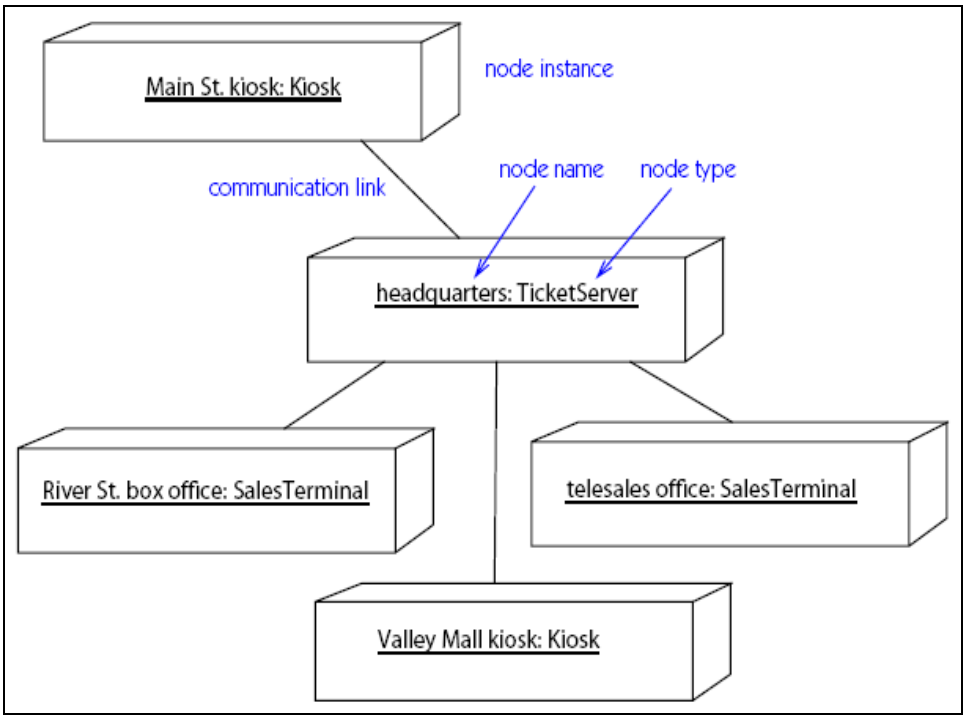


Figura 3.13: Exemplo de Diagrama de Implementação (Fonte: [12])

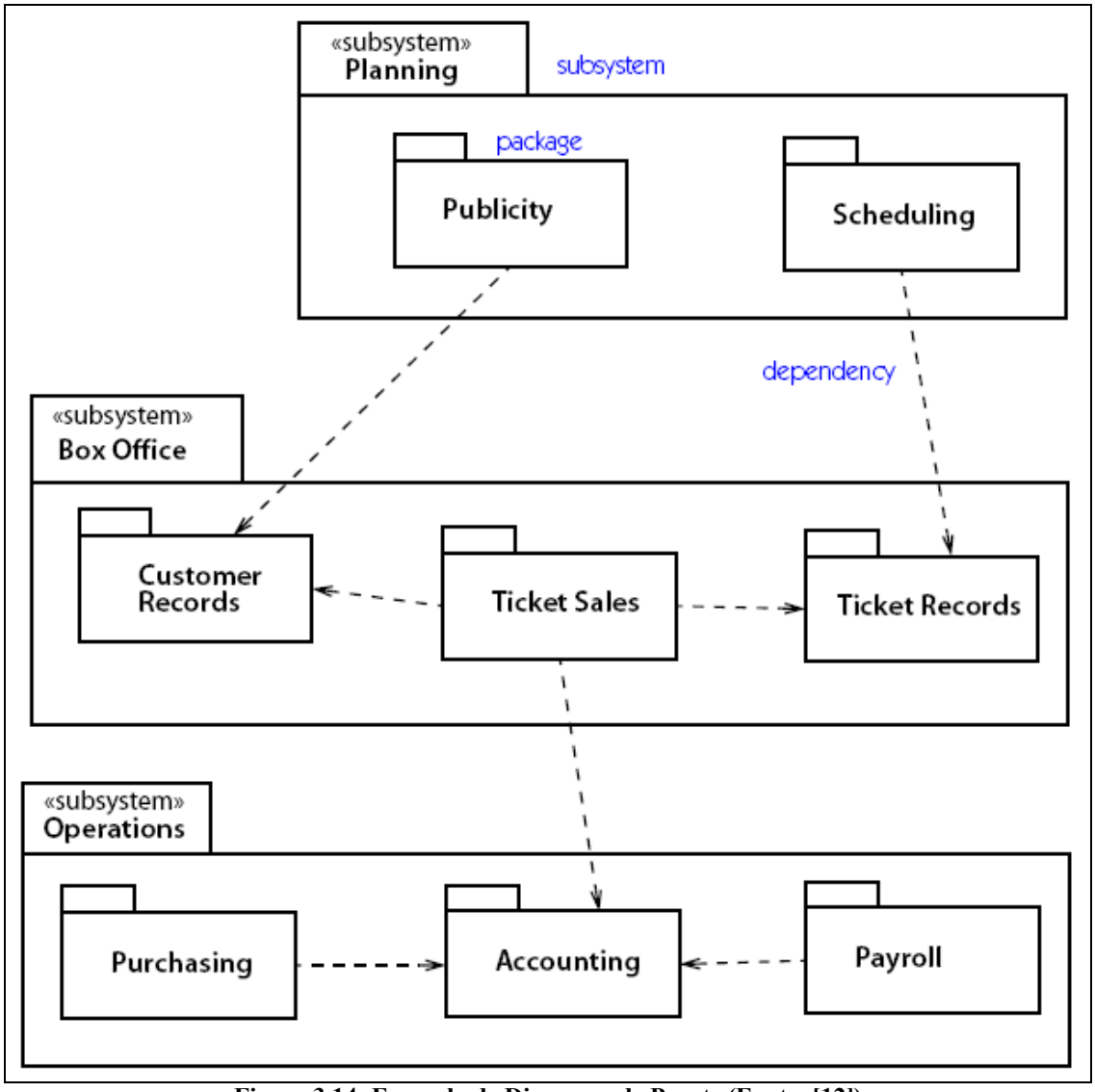


Figura 3.14: Exemplo de Diagrama de Pacote (Fonte: [12])

- Diagrama de Temporal verifica as condições de mudança em função do tempo. Esse diagrama descreve comportamentos e interações de componentes e permite visualizar as alterações nos estados ou nos valores ou em outras condições desses componentes em relação ao tempo. Enquanto um Diagrama de Estados permite representar as alterações do estado de um objeto, um Diagrama de Tempo permite visualizar qual o efeito do tempo no estado de um ou mais objetos relacionando-os entre si. Há dois tipos básicos de Diagrama de Tempo: i) de notação robusta mostrado na Figura 3.15; e ii) de notação concisa apresentado na Figura 3.16. Uma das alterações mais significativas na linguagem para a representação de produtos de software de tempo real é a adição do Diagrama de Tempo [13];

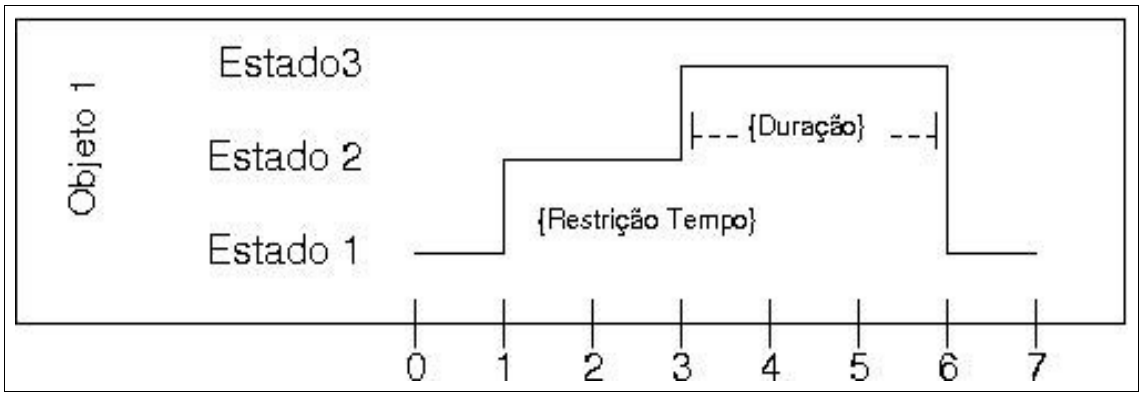


Figura 3.15: Exemplo de Diagrama de Temporal (Fonte: [14])

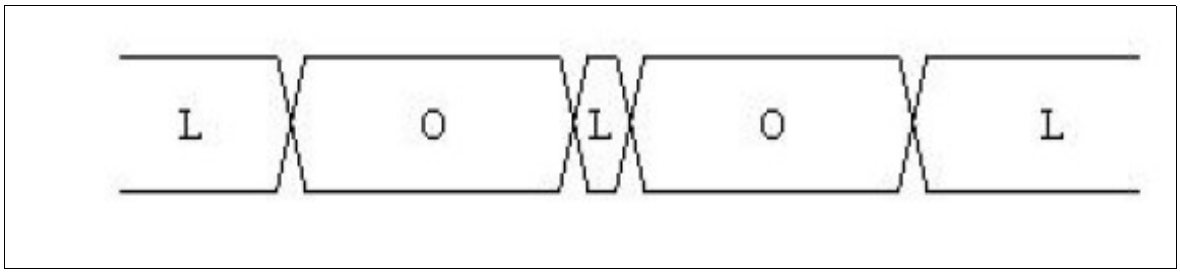


Figura 3.16: Exemplo de Diagrama de Tempo (Fonte: [14])

- Diagrama de Estrutura Composta destina-se à descrição dos relacionamentos entre os elementos, sendo utilizado para descrever a colaboração interna de classes, de interfaces ou de componentes para especificar a funcionalidade (Figura 3.17) [13]. Este diagrama pode representar os relacionamentos entre as classes "todo" e suas "partes" e entre as próprias partes, através de *links* que habilitam a comunicação entre duas ou mais instâncias. Para estes *links*, a UML define dois tipos de conectores [15]:
    1. *Assembly*. Permite que uma classe "parte" supra serviços que outra classe "parte" necessita, conectando duas partes como uma associação;
    2. *Delegation*. Conecta o "todo" com uma de suas "partes", sendo exibido com uma linha saindo da extremidade da classe composta para uma de suas "partes" dentro da classe composta.
- Diagrama de Visão Geral da Interação é a fusão do Diagrama de Atividades e do Diagrama de Seqüência, ou seja, o diagrama mostra um fluxo de atividades e como eles trabalham em uma seqüência de eventos (Figura 3.18). Este diagrama permite que fragmentos das interações sejam facilmente combinados aos pontos e aos fluxos de decisão [10 e 13].

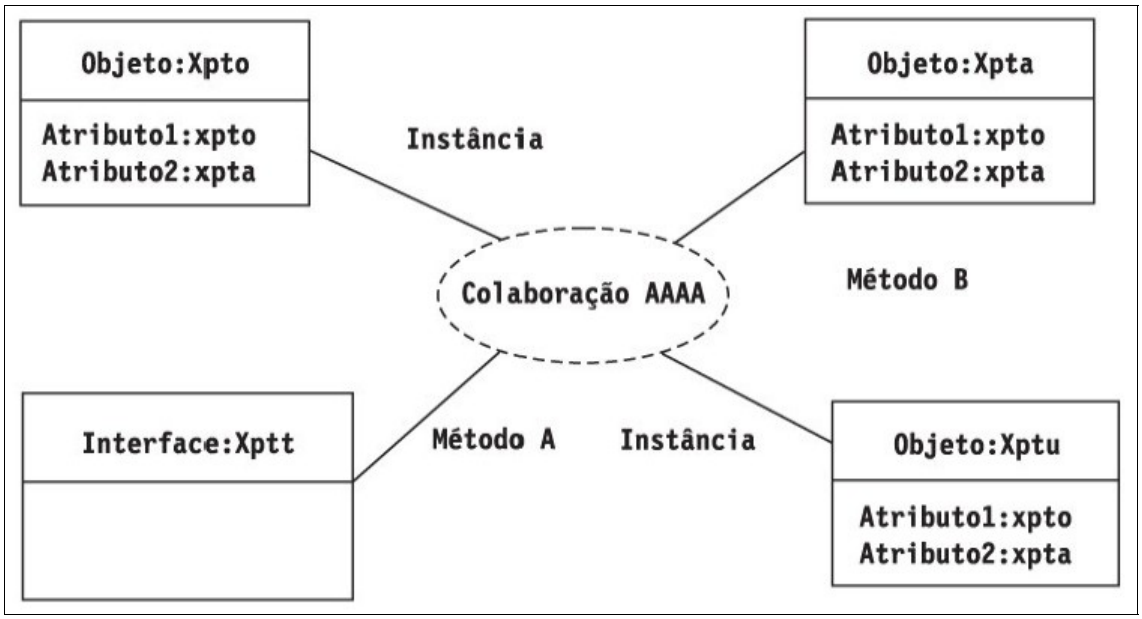


Figura 3.17: Exemplo de Diagrama de Estrutura Composta (Fonte: [16])

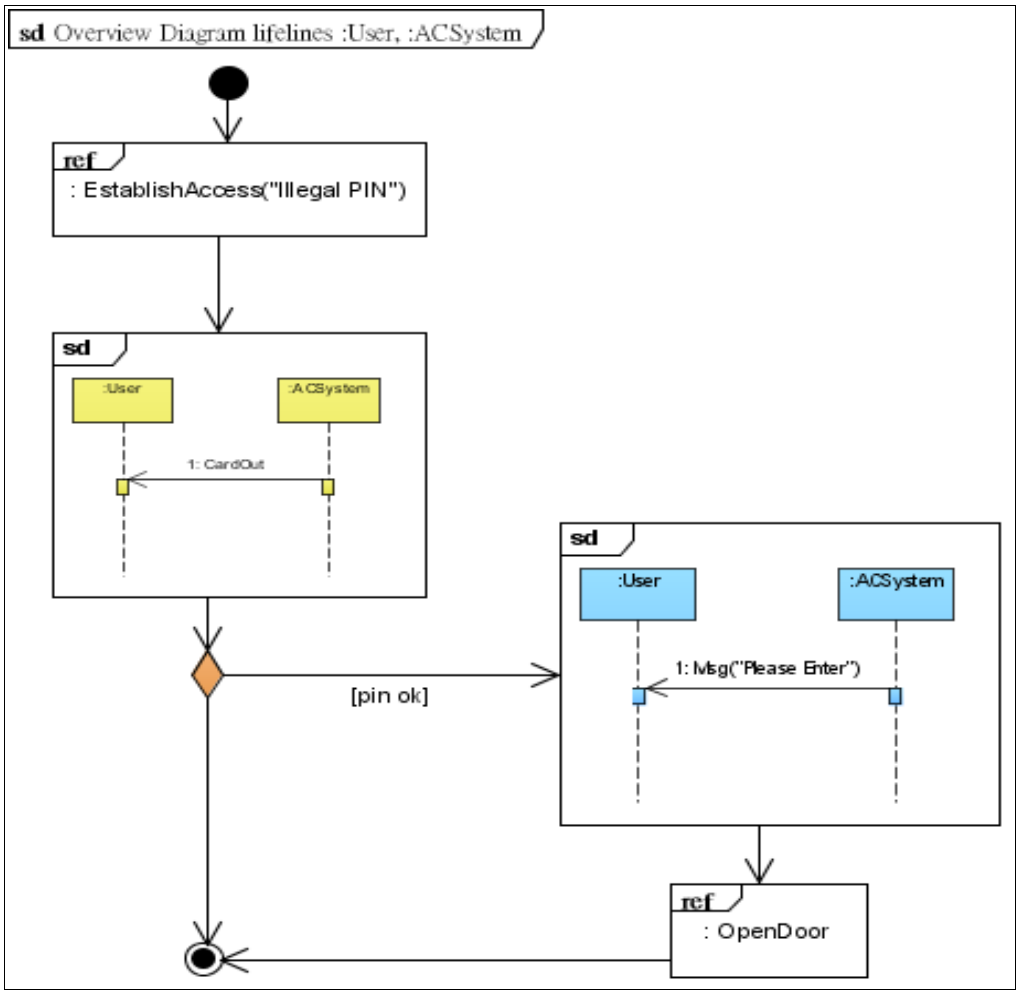


Figura 3.18: Exemplo do Diagrama Visão Geral da Interação (Fonte: [17])

3.5 Mecanismos de Extensão

A UML pode ser estendida ou adaptada para um método específico, uma organização ou um usuário. De acordo com OMG, esses mecanismos são elementos de modelagem que permitem o tratamento visual especializado e definem como criar novas semânticas [10]. A UML dispõe de um rico conjunto de elementos e conceitos de modelagem, os quais foram cuidadosamente definidos para atender aos projetos de software. Porém, seus usuários podem precisar de notações adicionais além daquelas definidas pela UML. Os três mecanismos de extensão podem ser usados em conjunto ou separadamente para representar novos elementos de modelagem que possuem semântica, características e notação próprias [15]. São eles:

- Estereótipos. Define um novo elemento de modelagem (extensão do vocabulário) cuja origem está em elementos de modelagem existentes na UML. Por isso, a semântica de um estereótipo é a extensão daquela definida para os elementos de modelagem existentes. Não há restrições quanto ao uso de um estereótipo de um elemento de modelagem, podendo este ser usado em uma mesma situação com o elemento de modelagem original. Os estereótipos são usados com diversos elementos de modelagem, tais como classes, componentes, relacionamentos e notas. A UML possui um conjunto de estereótipos pré-definidos usados para ajustar os elementos de modelagem ao invés de serem definidos novos elementos. Isso reforça uma característica importante da UML, que é a simplicidade da linguagem. Porém, não inviabiliza a sua especialização, ou seja, maior detalhamento de alguns de seus elementos de modelagem;
- Valor Atribuído (*tagged value*). Utilizado para explicar propriedades de elementos definidos. Este valor representa informações sobre o modelo ou os elementos de modelagem, não o produto de software que está sendo modelado. A regra para o uso de *tagged value* e o valor associado são definidos conforme se segue:

  {*tagged value* = "valor"}

  por exemplo: {autorM ="Jose da Silva"}

  O *tagged value* pode ser colocado dentro de um *container* ou muito próximo a ele. Essencialmente, define regras ou condições para diversos relacionamentos entre elementos de modelagem de um digrama da UML. Existem alguns *tagged value* predefinidos na UML, tais como *invariant*, *precondition* e *postcondition*, que

poderão dar suporte ao uso de contratos entre elementosde modelagem;

- Restrições. É uma extensão das condições ou das restrições propriamente dita e expressa textualmente, com o propósito de declarar as semânticas a um elemento de modelagem, ou seja, caracterizam-se, essencialmente, no fato de adicionar novas regras ou modificar as existentes dos elementos de modelagem.

3.6 Considerações Finais

Com o desenvolvimento de produtos de software mais complexos, é necessário o uso de uma linguagem unificada para a comunicação entre a equipe envolvida (cliente, analista, programador, entre outros) e para a construção da documentação (requisitos e código-fonte).

Neste contexto, a UML é a linguagem de notação da qual o desenvolvedor pode seguir suas etapas para o fluxo correto do desenvolvimento de um projeto de software, pois possui as abstrações para descrever e caracterizar o produto de software, além de suportar as cinco fases de desenvolvimento de produtos de software.

4 ABORDAGENS DE MODELAGEM PARA O DESENVOLVIMENTO DE SOFTWARE PARA *WEB*

4.1 Considerações Iniciais

Tendo em vista que *WebApps* são um tipo de produto de software, várias abordagens de modelagem têm sido propostas para sistematizar a fase de análise, a fase de projeto e a fase de implementação, com foco na modelagem do domínio do problema e na organização estrutural e navegacional.

A modelagem de *WebApps* não é um processo totalmente definido na literatura e existem muitos aspectos que devem ser considerados na produção de uma aplicação com qualidade. A Engenharia de Software Tradicional não atende aos requisitos deste domínio de aplicação, necessitando novos conceitos advindos da Engenharia Softwarepara *Web*.

4.2 OOHDM

Desenvolvido por Gustavo Rossi, OOHDM (*Object-Oriented Hypermedia Design Method*) é um método para construção de aplicações hipermídia e *Web*, sendo uma extensão orientada a objetos de HDM (*Hypermidia Design Model*), incluindo primitivas especiais de modelagem para o projeto navegacional e de interface [18] . Uma das características de aplicações hipermídia é a noção de navegação. Em OOHDM, uma aplicação é trabalhada como uma visão navegacional do modelo conceitual. Segundo Rossi *apud* [19], OOHDM considera o processo de desenvolvimento da aplicação hipermídia como um processo de quatro atividades, desempenhadas em uma mistura de estilos iterativos e incrementais de desenvolvimento; em cada etapa um modelo é construído ou enriquecido.

Para consolidar-se na área de modelagem de aplicações hipermídia, OOHDM herdou de seu precursor algumas idéias e enriqueceu outras, como é possível observar na Tabela 4.1 [08].

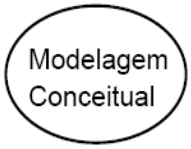
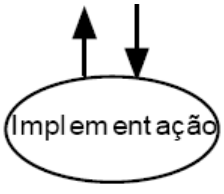
OOHDM trata o processo de desenvolvimento de um *WebApps* com foco na hipermídia em quatro atividades diferentes: i) Modelagem Conceitual; ii) Projeto Navegacional; iii) Projeto de Interface Abstrata; e iv) Implementação.

A Tabela 4.2 apresenta as quatro etapas do projeto OOHDM, os produtos que são gerados em cada uma delas e os produtos e os interesses definidos para o projeto. Em cada uma das atividades, uma parte do projeto é desenvolvida, de forma que é necessária a conclusão de uma etapa para o início da próxima (Rossi *apud* [19]).

Tabela 4.1: Características Herdadas e Criadas (Fonte: [08])

Características Herdadas do *HDM*	Características Criadas ou Enriquecidas
Reconhecimento da modelagem ser independente do ambiente e do modelo de referência.	Não é apenas uma abordagem de modelagem, mas uma metodologia, pois aborda muitas atividades.
Reforça as estruturas hierárquicas, oferecendo a possibilidade da construção de agregados.	Na fase de modelagem conceitual, primitivas são mais ricas. Como são orientadas a objetos, elas suportam: agregação, generalização e herança.
Inclui o conceito de perspectiva de atributo, onde cada classe folha implementa uma visão possível de atributo.	É possível fazer uso do esquema conceitual para definir perfis de usuários diferentes.

Tabela 4.2: Metodologia dos Processos OOHDM (Fonte [19])

Atividades	Produtos	Mecanismos	Interesses do Projeto
	Classes, subsistemas, relacionamentos, perspectivas de atributos.	Classificação, composição, agregação generalização e especialização.	Modelagem da semântica do domínio de aplicação.
	Nós, elos, estruturas de acesso, contextos de navegação, transformações navegacionais.	Mapeamento entre objetos conceituais e de navegação. Padrões de navegação para a descrição da estrutura geral da aplicação.	Leva em conta o perfil do usuário e a tarefa; ênfase em aspectos cognitivos e arquiteturais.
	Objetos de interface abstrata, reações a eventos externos, transformações de interface.	Mapeamento entre objetos de navegação e objetos de interface.	Modelagem de objetos perceptíveis, implementa metáforas escolhidas. Descrição de interface para objetos navegacionais.
	Aplicação em execução.	Aqueles fornecidos pelo ambiente alvo.	Desempenho, completitude.

Segundo [18], a maior contribuição oferecida pelo OOHDM consiste na geração da modelagem navegacional que, para a sua construção, leva em consideração os tipos de

usuários e o conjunto de atividades que poderão ser desenvolvidas.

4.2.1 Modelo Conceitual

Na Modelo Conceitual, é construído um modelo do domínio da aplicação, utilizando os princípios (objetos conceituais e abstrações) da modelagem orientada a objetos e uma notação similar a UML [01]. O produto desta fase é o esquema de classes da aplicação, baseando-se nas classes primitivas, relacionamentos e mecanismos de abstração. As principais diferenças em relação a UML são o uso de atributos de valores múltiplos, ou seja, podem representar várias perspectivas sobre determinada entidade real do domínio do problema (conceito de perspectiva de HDM) e o uso de direções nos relacionamentos.

Este esquema conceitual é, basicamente, um conjunto de classes e de objetos e os relacionamentos são responsáveis em fornecer a ligação entre eles. As classes são descritas através de atributos e de métodos, de acordo com o comportamento dessas classes na implementação. Os relacionamentos possuem cardinalidade, mas podem conter atributos e, em algumas situações, comportamentos. Quando o Modelo Navegacional for gerado, as classes são transformadas em nós e os relacionamentos em elos.

Os atributos são responsáveis por representar as propriedades essenciais ou de conceitualização sobre os objetos. Os tipos ou as classes de atributos representam um relacionamento implícito ou a aparência desse atributo. Cada uma dessas aparências é classificada como uma perspectiva do atributo, ou seja, um atributo pode ser visto de diferentes formas, de acordo com a situação.

No caso de um atributo possuir várias perspectivas, é utilizado "[perspectiva1, perspectiva2 ....]" para identificar essa característica. Caso exista apenas uma, a perspectiva *default* é identificada com o uso de "+", sendo que este deve estar presente nas instâncias. A Figura 4.1 demonstra um exemplo de representação de múltiplas perspectivas. Nesta figura, observa-se que na classe *Pessoa* o atributo *descrição* apresenta múltiplas perspectivas; assim, além da descrição textual, que é o atributo *default* (simbolizado pelo sinal de +), existe uma descrição através de imagem [19].

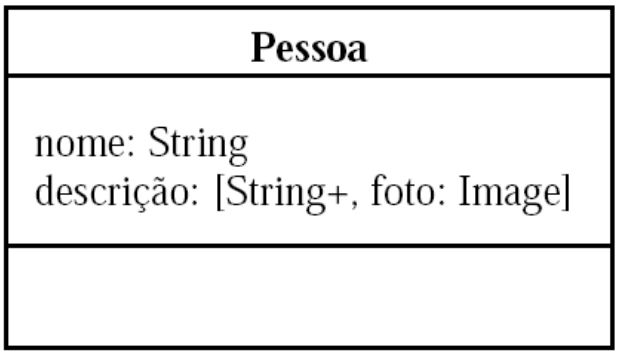


Figura 4.1: Exemplo de Atributo Múltiplo (Fonte:[19])

Os mecanismos de abstração oferecidos são a agregação, a generalização/especialização e o subsistema. A agregação, mecanismo próprio para trabalhar com complexidade, é útil para descrever classes complexas como agregadas de classes mais simples. A generalização é utilizada na construção de hierarquias de classe e para o emprego de herança para compartilhar dados. O subsistema é um mecanismo de empacotamento para abstração de modelos em domínios complexos.

A Figura 4.2 mostra que, ao ocorre Generalização de classes, os atributos da superclasse são herdados pela classe especializada; assim, ela possui os atributos da superclasse, inclusive o atributo *default*, e mais os atributos específicos de sua classe. Caso a classe especializada também possua um atributo *default*, este passa a ser a perspectiva obrigatória e o da superclasse passa a ser opcional [19].

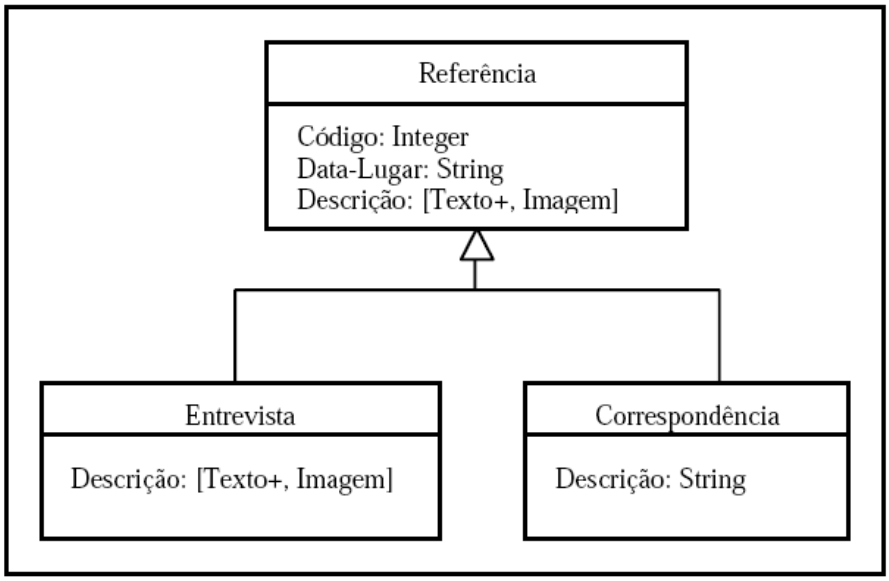


Figura 4.2: Exemplo Mecanismo de Abstração – Generalização (Fonte: [19])

O comportamento esperado para o Modelo Conceitual é dependente das

características esperadas para uma aplicação, principalmente quando se trata de uma aplicação que seja capaz de adaptar-se ao usuário de forma dinâmica, permitindo inúmeras atividades computacionais.

A Figura 4.3 exibe o Modelo Conceitual para o material didático sobre exposição de quadros. O modelo mostra a relação entre o autor e a obra a ele relacionada e o evento no qual estão inseridos.

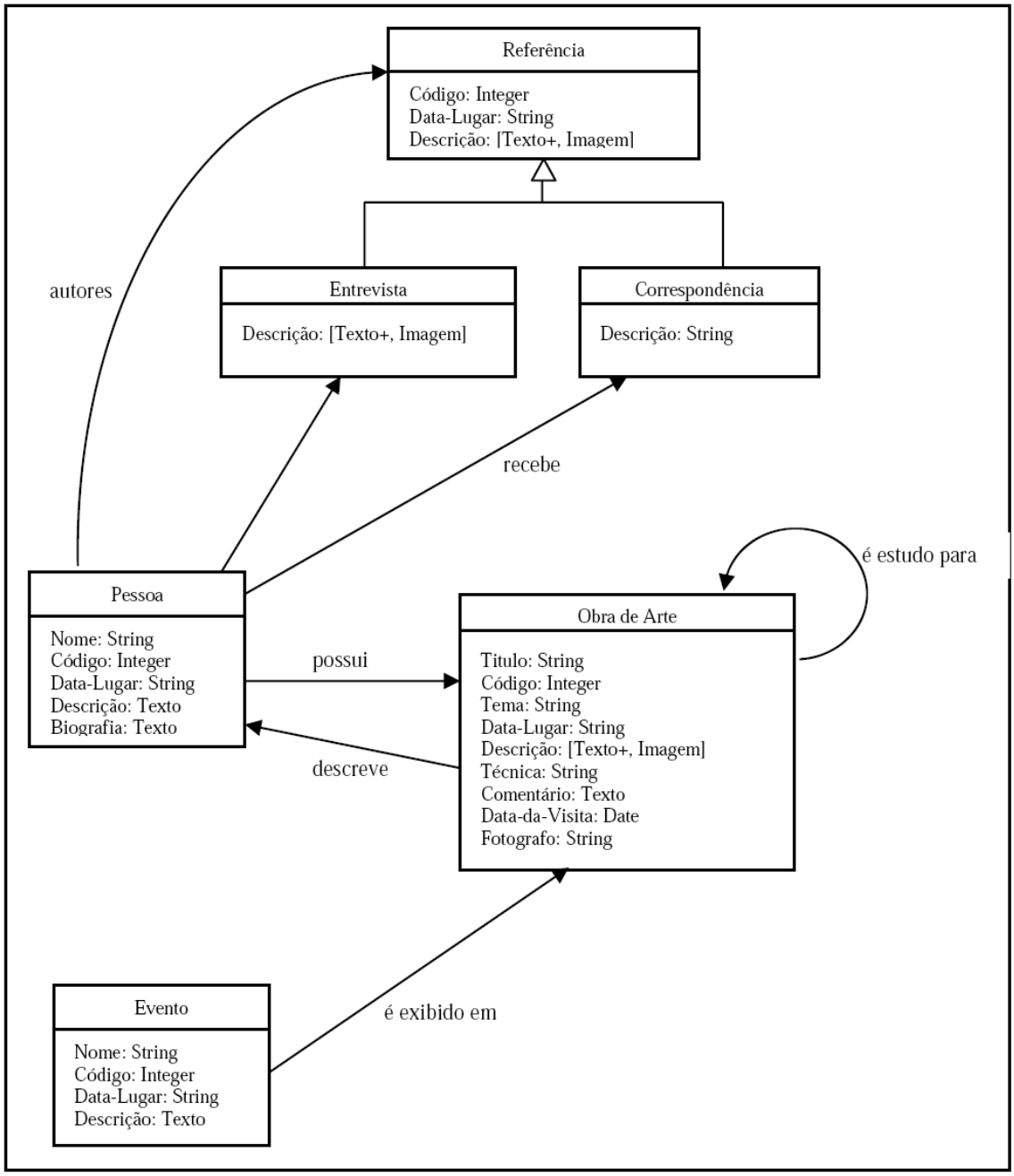


Figura 4.3: Exemplo de Modelo Conceitual – OOHDM (Fonte: [19])

4.2.2 Projeto da Navegação

No Projeto da Navegação ou Modelo Navegacional, um dos principais desafios em *WebApps* é definir as informações que farão parte da aplicação, bem como a navegação entre elas. Em OOHDM, pode-se definir o Modelo de Navegação a partir do Modelo Conceitual ou diretamente (sem que a modelagem conceitual tenha sido construída). Geralmente, quando é feita reengenharia de um aplicativo existente, o Modelo Navegacional é definido a partir do Modelo Conceitual ou quando existem vários perfis de usuários para uma mesma aplicação [19 e 20].

Uma das principais inovações de OOHDM é reconhecer que os objetos navegacionais são diferentes dos objetos conceituais, pois os objetos em que o usuário navega não são os objetos do Modelo Conceitual, mas outros tipos de objetos os quais são construídos de um ou mais objetos conceituais. Segundo Schwabe *apud* [20], a maior contribuição de OOHDM está no Modelo Navegacional. De acordo com Rossi *apud* [19], em OOHDM, foi proposta a construção de um Modelo Conceitual compartilhado no qual os objetos e os relacionamentos do domínio da aplicação são focalizados. Mais tarde, diferentes visões navegacionais são derivadas desse modelo, levando em conta os perfis de futuros usuários. O Modelo Conceitual funciona como um repositório compartilhado de modelagem, a partir do qual são construídas diferentes visões do domínio do problema.

A construção do Modelo Navegacional deve levar em consideração diversos pontos como quais visões são permitidas aos usuários, quais os objetos que podem ser navegados por ele e quais os relacionamentos existentes entre os objetos e os atributos. O desenvolvimento de um Modelo Conceitual bem elaborado contribui para um bom Modelo Navegacional. A partir do Modelo Conceitual, são criadas as visões que definem um conjunto de classes e contextos, que são responsáveis, respectivamente, em mostrar os relacionamentos entre os objetos navegacionais e a navegação em geral.

A Figura 4.4 exibe um exemplo parcial de um Contexto Navegacional para material didático. Esse modelo define os caminhos que o usuário deverá percorrer, desde o *menu* principal até acessar os nós da aplicação. Segundo esta figura, o usuário pode, a partir do Menu Principal, acessar o nó Exemplo e, a partir dele, percorrer o elo de ligação até o nó Imagem ou navegar para os outros nós que podem se ligar ao nó Exemplo.

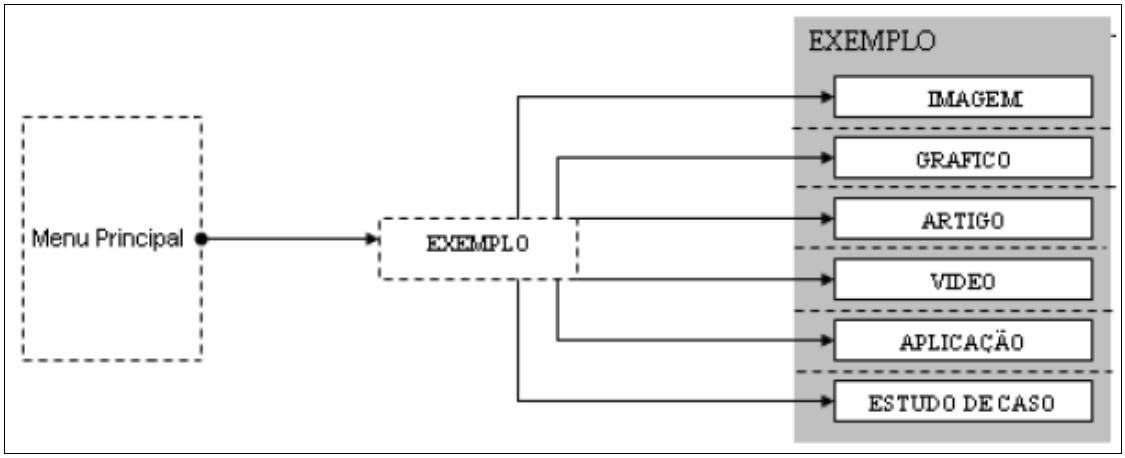


Figura 4.4: Exemplo de Modelo Navegacional – OOHDM (Fonte: [19])

4.2.2.1 Classes Navegacionais

As classes navegacionais são responsáveis pela navegação do usuário, especificando quais objetos são vistos pelos usuários. Essas classes navegacionais são especificadas com especializações de classes básicas que definem a semântica dos objetos navegacionais.

A Figura 4.5 mostra a hierarquia das classes navegacionais em OOHDM, na qual nó, âncora, elo, destino de elo e transformação são considerada as classes básicas.

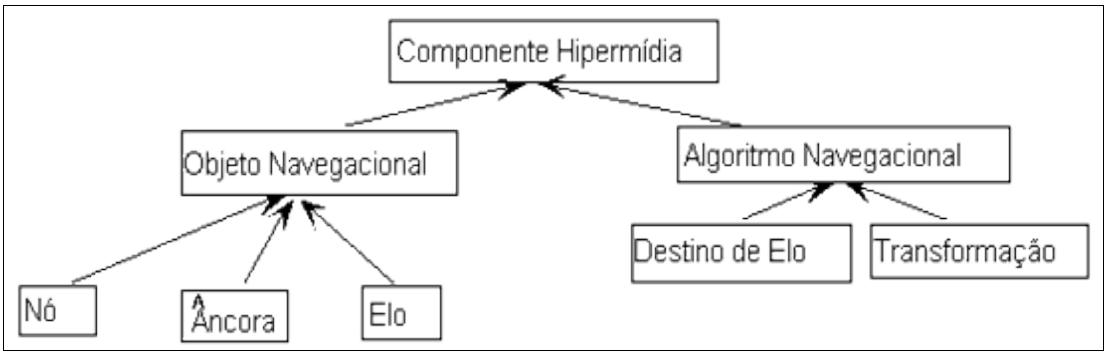


Figura 4.5: Hierarquia de Classes Navegacionais (Fonte: [18])

As classes básicas de OOHDM são descrita da seguinte forma [19]:

- Nó. Contém informações básicas nas aplicações hipermídia. A estrutura dos nós depende da semântica da aplicação e dos interesses dos usuários que utilizarão essa aplicação. A Figura 4.6 representa um nó (para diferenciar um nó de uma classe conceitual, há um linha vertical na primeira parte a direita);

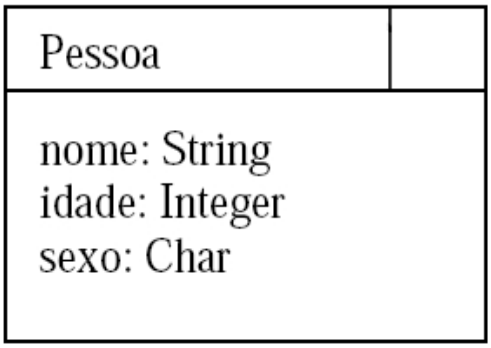


Figura 4.6: Exemplo de Nó (Fonte: [19])

- Elo: É derivado dos relacionamentos do Modelo Conceitual e permitem a ligação entre os objetos navegacionais, podendo ligar um objeto a outro ou a vários outros objetos. O comportamento do elo é expresso pela definição das semânticas navegacionais. A Figura 4.7 mostra que os elos são representados através de setas.

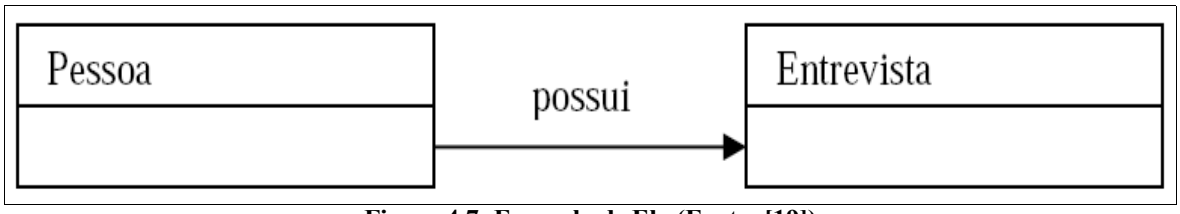


Figura 4.7: Exemplo de Elo (Fonte: [19])

- Âncora: É gatilho responsável por disparar uma navegação depois de selecionada. As âncoras são utilizadas de acordo com o contexto em que o nó correspondente é navegado. As âncoras de acesso aos elementos de um contexto apresentam a seguinte sintaxe:

  nome-âncora: âncora(índice(nome-contexto(parâmetros-contexto)))

  De acordo com [18], as âncoras podem ser representadas como objetos da interface, porém elas não precisam ser necessariamente representadas na interface, apesar de comumente isso acontecer. As âncoras são utilizadas de acordo com o contexto em que o nó correspondente énavegado;
- Transformação: É a classe abstrata oferecendo diferentes algoritmos para as transformações navegacionais. Suas sub-classes definem transformações simples ou mais sofisticadas. Esta classe e suas sub-classes oferecem uma implementação simples (orientada a objetos) dos Diagramas de Navegação. Quando as transformações dependem do estado dos nós de origem, pode ser necessário definir uma hierarquia de Estados de Nós para uma implementação limpa e completa do Diagrama Navegacional;

- Destino de Elo: É a classe abstrata de algoritmos que calculam o ponto final de um elo.

4.2.2.2 Contextos Navegacionais

Os Contextos Navegacionais expressam a Estrutura Navegacional da aplicação. Eles são constituídos por um conjunto de elos, de nós e de outros Contextos navegacionais aninhados, incluindo um caminho pré-definido entre os elementos que o compõe. Eles contribuem para a organização dos objetos navegacionais, proporcionando um espaço navegacional consistente, que auxilia o usuário para que ele não se perca durante sua navegação e evita a apresentação de informações redundantes, ajudando o usuário a escolher a maneira como navegar de forma consciente e controlada.

A definição de um Contexto Navegacional promove a identificação do comportamento dos objetos que fazem parte desse contexto. Os elementos que contribuem para a definição de um contexto são [19]:

- Classes Contexto: Definem as características que fazem a distinção do Contexto Navegacional, podendo conter atributos de diferentes tipos, tais como tipos de mídia para apresentar informações sobre os contextos, índices que surgem no momento que o objeto de contexto é acessado e âncoras que levam a elos de contextos ou estruturas de acessos;
- Classes em Contexto: Apresentam informações adicionais sobre os nós pertencentes a um contexto. As classes em contexto possuem informações significativas ao contexto, por exemplo sentido na travessia existente entre objetos navegacionais; assim, elas definem as reações de uma âncora no momento em que ela é selecionada;
- Elos de Contexto: São responsáveis em relacionar os elos dentro de um mesmo contexto. Esses elos são acessados a partir das âncoras definidas em classes de nó ou nas classes em contexto, obtendo como resultado a abertura do objeto destino dentro do contexto desejado.

Segundo [18], os elementos que compõem um hiperdocumento são mais facilmente compreendidos quando são apresentados em um contexto, minimizando os problemas de desorientação em aplicações complexas. Esses contextos são comumente induzidos pelas Classes Navegacionais. Na Figura 4.8, estão os elementos gráficos utilizados para a representação de Contextos Navegacionais.

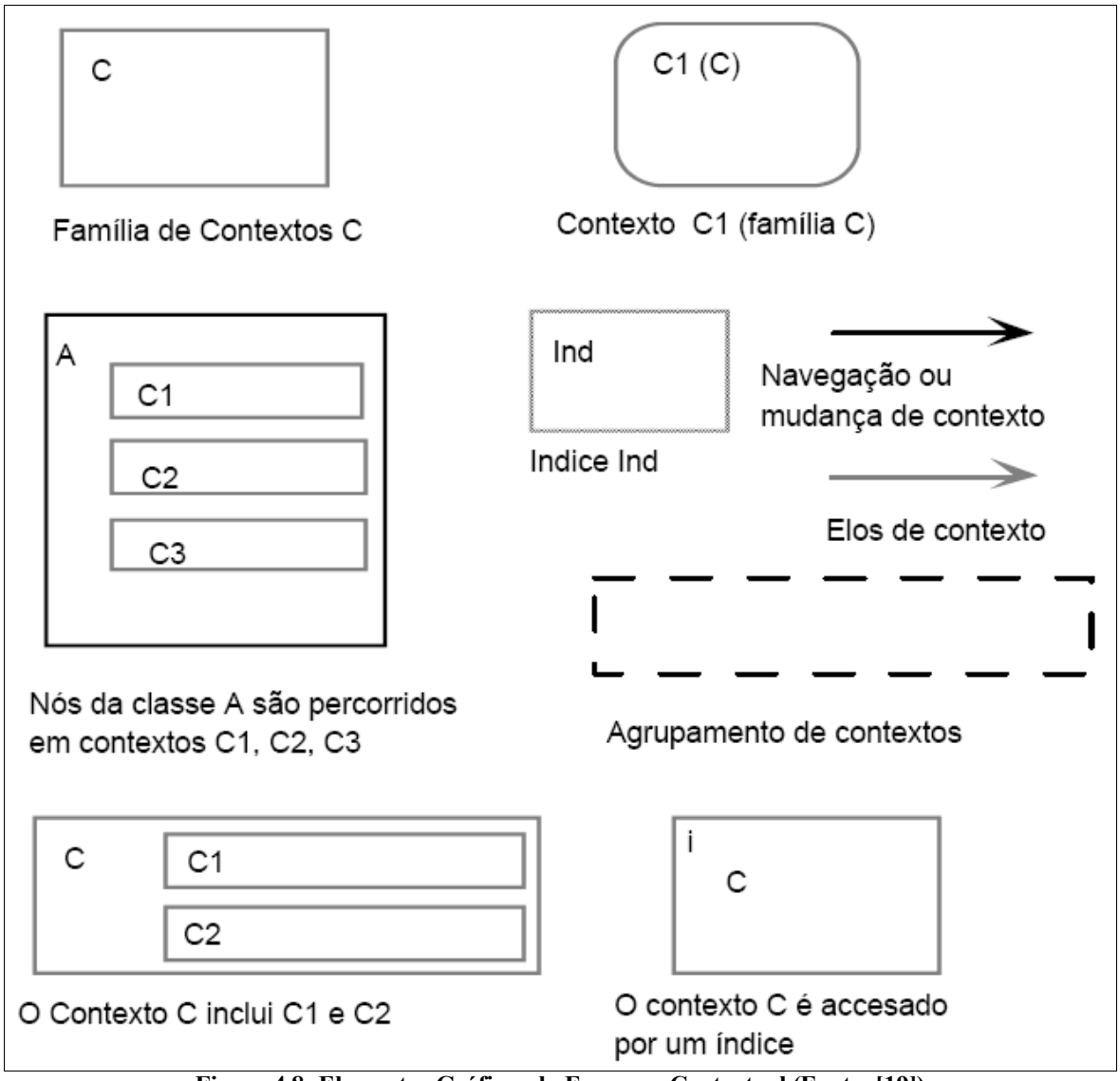


Figura 4.8: Elementos Gráficos do Esquema Contextual (Fonte: [19])

4.2.3 Projeto de Interface Abstrata

No Projeto de Interface Abstrata, OOHDM usa o Modelo ADVs (*Abstract Data Views* – Visões de Dados Abstratas) (Cowan; Lucena *apud* [01]), criado para especificar formalmente a separação entre a interface e os componentes de software. Uma ADV oferece um método de projeto independente da implementação, proporcionando maior reusabilidade de componentes de projeto e de interface. Para especificar os padrões dinâmicos da interface, OOHDM usa o formalismo visual ADV*charts* (Carneiro *et al. apud* [01]).

Na Figura 4.9, Texto-Ancorado é especificado como possuindo dois estados: i) navegação-Desautorizada; e ii) navegação-Autorizada. O Botão1 e o Botão2 servem como controles para a autorização e a desautorização da navegação Texto-Ancorado. Quando o

evento Mouse-Clicado é produzido e o foco está no Botão1, a transição 2 é desencadeada e o evento navegação-Autorizada é gerado; isto é chamado de comunicação disseminada (*broadcast*). Quando o usuário clica no Botão2, o evento desautorizar-Navegação é produzido (comunicação disseminada) e a transição 5 é desencadeada. A transição 4 é desencadeada quando TextoAncorado está no estado navegação-Desautorizada e o evento autorizar-Navegação ocorre. No estado navegação-Autorizada, quando o usuário clica o *mouse* em alguma âncora, isto é, quando o evento MouseClicado ocorre e o foco está em algum membro da coleção âncoras, a transição 6 é desencadeada.

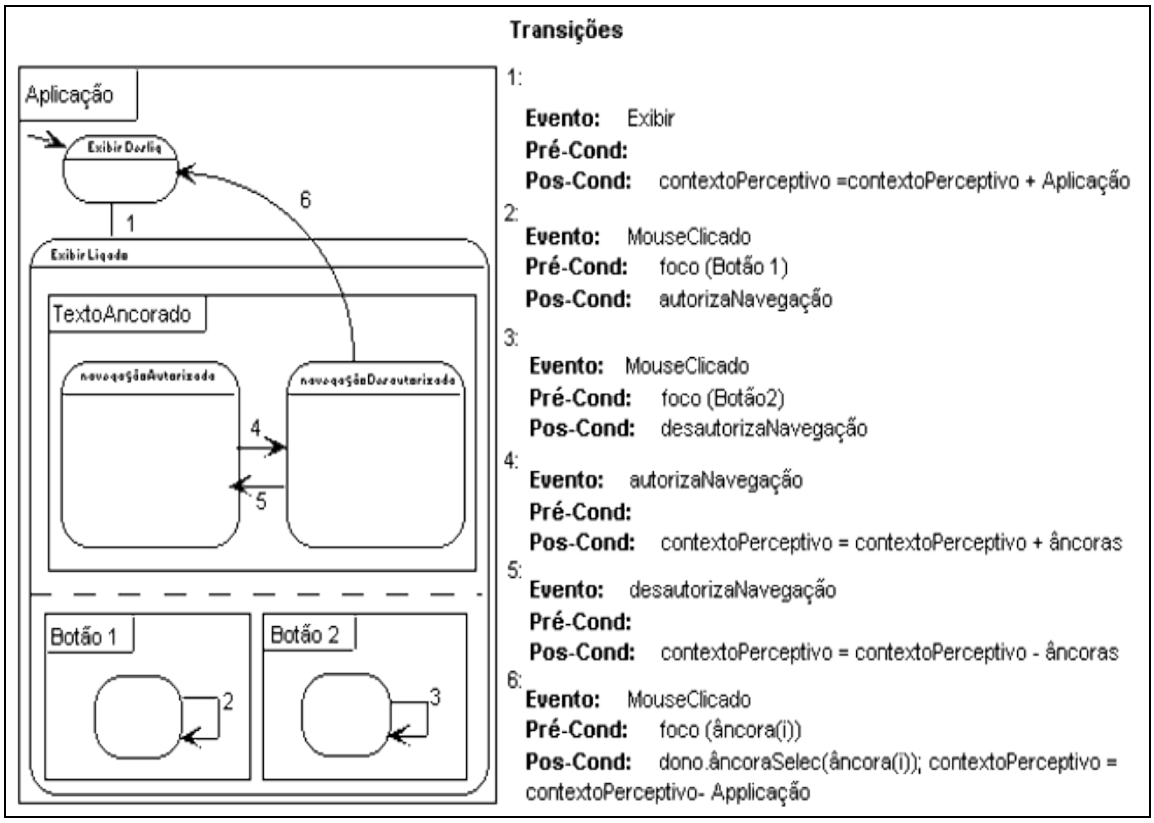


Figura 4.9: Exemplo de ADV (Fonte: [21])

Para especificar um Projeto de Interface Abstrata, é necessário definir metáforas de interface e descrever suas propriedades estáticas e dinâmicas e seus relacionamentos com o Modelo Navegacional de uma forma independente de implementação. Para isso, é necessário especificar [22]:

- A aparência de cada objeto navegacional percebido pelo usuário, isto é, a representação de seus atributos (incluindo as âncoras). O mesmo objeto navegacional pode ter diferentes representações de interface em diferentes situações. Por exemplo, um nó pode ter a representação de uma fotografia de um momento ou a de um ícone

em um mapa que atue como um índice para momentos;

- Outros objetos de interface para oferecer as diversas funções do aplicativo, como barras de *menus*, botões de controle e *menus*;
- Os relacionamentos entre os objetos de interface e os objetos navegacionais, tais como o modo com que um evento externo (por exemplo, o clicar do *mouse*), afetará a navegação;
- As transformações de interface que ocorrem pelo efeito da navegação ou de eventos externos no computador de diferentes objetos de interface;
- A sincronização de alguns objetos de interface deve ser considerada, especialmente quando há meios dinâmicos, como áudio e vídeo envolvidos.

Visualmente, os ADVs são apresentados como caixas, os estados como caixas arredondadas e as transições como setas. As semânticas dos ADVs e dos estados são um pouco diferentes. Enquanto um ADV é um objeto abstraindo estrutura e comportamento, os estados são usados apenas para expressar aspectos comportamentais da interface.

4.2.4 Implementação

Esta é a fase final de OOHDM. Nesta etapa, os modelos construídos anteriormente são traduzidos para um ambiente de implementação específico. OOHDM possui um ambiente de implementação específico para a *WebApp* denominado OOHDMWeb, que cria índices em páginas HTML (*HyperText Markup Language*) representando os contextos a partir das informações armazenadas em tabelas [08].

4.3 WebML

WebML (*Web Modeling Language*) é uma abordagem utilizada para modelar aplicações *WebApp*, guiada por modelos, sendo semelhante a SHDM (*Semantic Hypermedia Design Method*) e a OOHDM. Os principais objetivos de WebML são [23]:

- Expressar a estrutura de uma aplicação *Web* em uma descrição de alto nível, que pode ser utilizada para pesquisa, evolução e manutenção;
- Fornecer múltiplas visões do mesmo conteúdo;
- Separar o conteúdo de informação dos aspectos de navegação e apresentação, podendo evoluir separadamente; e
- Armazenar a estrutura da informação em um repositório, que pode ser utilizado para a geração dinâmica da *WebApp* .

Para o desenvolvimento de um *site* com WebML, são utilizados quatro modelos distintos: i) Modelo de Dados; ii) Modelo de Hipertexto (Modelo de Composição e Modelo Navegacional); iii) Modelo de Apresentação; e iv) Modelo de Personalização [08].

4.3.1 Modelo de Dados

WebML utiliza notações clássicas para representar o Modeo de Dados: modelo Entidade e Relacionamento (MER), ODMG (*Object Database Management Group*) e Diagrama de Classes UML [08]. Existem dois elementos fundamentais no Modelo de Dados [08]: i) entidades, representam um grupo de objetos com características comuns e podem ser organizadas em hierarquias de generalização; e ii) relacionamentos, representam o significado das conexões entre as entidades.

A Figura 4.10 mostra um exemplo do modelo dos dados, representando a informação sobre albums musicais, que são compostos pelos artistas, sobre quem são fornecidas algumas revisões. Cada álbum pode conter diversas trilhas.

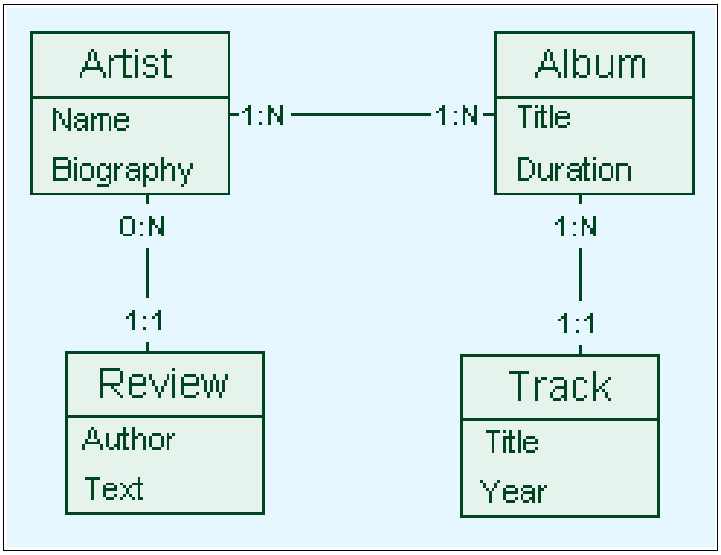


Figura 4.10: Exemplo de Modelo de Dados (Fonte: [24])

4.3.2 Modelo de Hipertexto

O Modelo de Hipertexto descreve um ou mais hipertextos que podem ser publicados no *site*. Cada hipertexto diferente define uma *site view* que consiste de dois sub-modelos: i) Modelo de Composição; e ii) Modelo Navegacional.

4.3.2.1 Modelo de Composição

No Modelo de Composição, são especificadas as páginas que compõem o *site* e estes

são compostos por unidades de elementos visuais (*units*). *Units* representam as informações descritas no Modelo de Dados e podem ser de seis tipos: i) *data*; ii) *multi-data*; iii) *index*; iv) *scroller filter*; v) *MultChoice* Index; e vi) *direct-units*. As páginas fazem parte do Modelo de Composição, pois são nelas que *units* devem ser inseridas [25]:

- *Data,* publica informação sobre um único objeto de uma entidade, por exemplo uma instância de uma entidade (pessoa, user, objeto e etc) (Figura 4.11);
- *Multi-data*, é parecida com a *data unit,* mas publica informação sobre um conjunto de objetos de uma entidade, por exemplo, as instâncias de uma entidade (pessoa, user, objeto e etc) (Figura 4.12);

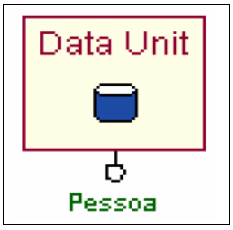


Figura 4.11: Exemplo de *Data Unit*
(Fonte: [25])

Figura 4.12: Exemplo de *Mult Data Unit*
(Fonte: [25])

- *Index,* publica uma lista de objetos de uma entidade. A diferença entre a *index unit* e a *multidata unit* é a primeira permitir que o objeto publicado seja selecionado (Figura 4.13);
- *Scroller,* fornece comandos para acessar os elementos de um conjunto ordenado de objetos (Figura 4.14);

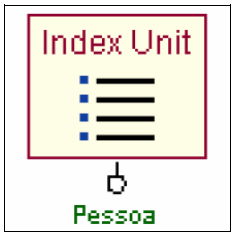


Figura 4.13: Exemplo de *Index Unit*
(Fonte: [25])

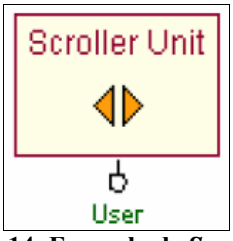


Figura 4.14: Exemplo de *Scroller Unit*
(Fonte: [25])

- *MultChoice Index*, é uma variação da *index unit*, onde cada elemento da lista possui um *ckech box,* permitindo a seleção de vários objetos (Figura 4.15);
- *Entry, a*presenta campos de edição para entrada de dados (Figura 4.16).

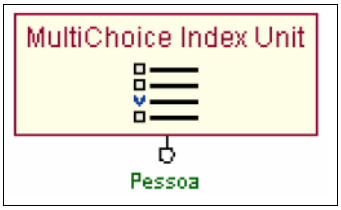


Figura 4.15: Exemplo de *MultiChoice Index* (Fonte: [25])

Figura 4.16: Exemplo *Entry* (Fonte: [25])

4.3.2.2 Modelo Navegacional

A navegação em um *WebApps* é definida através de ligações (*links*) que podem ser definidos entre *units* em uma única página, entre *units* estabelecidas em páginas diferentes ou entre páginas. Há quatro tipos de *links* no Modelo Navegacional, dois que funcionam com a interação do usuário e dois que funcionam sem a interação do usuário [25]:

- *Links* contextuais. Conectam *units* de uma forma coerente com relação à semântica expressa pelo esquema estrutural. *Link* contextual carrega informações (contexto) de uma unidade origem para uma unidade destino;
- *Links* não-contextuais. Conectam páginas de forma totalmente livre, isto é, independente de *units* que elas contêm e do relacionamento entre elas;
- *Links* automáticos (Figura 4.17). Se *link* entre duas *units* é automático, significa que o usuário não precisa interagir com *unit* de origem para que o conteúdo da *unit* de destino seja apresentado;
- *Link* de transporte (Figura 4.18). Usado quando é necessário passar informação de contexto entre *units* dentro de uma mesma página, mas sem a intervenção do usuário. Esse tipo de *link* não é apresentado para o usuário na forma de uma âncora ou botão na tela.

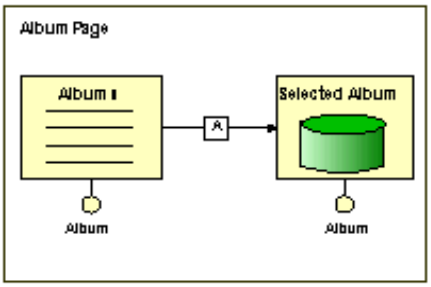


Figura 4.17: Exemplo de Link Automático (Fonte: [24])

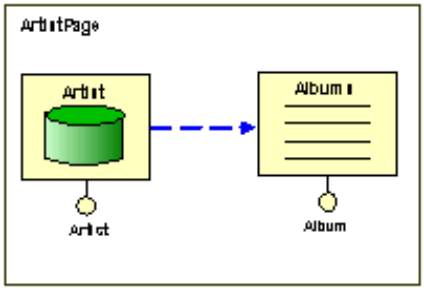


Figura 4.18: Exemplo de Link de Transporte(Fonte: [24])

4.3.3 Modelo de Apresentação

WebML não possui um modelo específico para apresentar graficamente as páginas

em nível conceitual, podendo ser representadas utilizando a linguagem XML (*eXtensible Markup Language*). A apresentação das páginas é considerada como uma transformação de documentos, constrói-se a especificação WebML de uma página em uma página escrita em uma linguagem de implementação concreta como: JSP ou ASP NET.

4.3.4 Modelo de Personalização

Usuários e grupos de usuários são explicitamente modelados na forma de entidades pré-definidas chamadas *User*, *Group* e *Module*. As características dessas entidades podem ser usadas para armazenar o conteúdo de um grupo específico ou do usuário individualmente, ver sugestões, listar favoritos e otimizar recursos gráficos.

4.4 UWE

UWE (*UML based Web Engineering*) é uma extensão da linguagem de modelagem UML para aplicações *Web* proposta por Koch, Hennicker e Kraus. Ela é baseada no RUP (*Rational Unified Process*) [01 e 26] e incorpora aspectos necessários ao desenvolvimento de uma *WebApp*. Dessa forma, é uma metodologia orientada a objetos e iterativa, baseada em padrões de processos de desenvolvimento unificado de produtos de software.

As fases de modelagem são: i) Modelo Conceitual; ii) Modelo Navegacional; e iii) Modelo de Apresentação. Para o Modelo Conceitual e o Modelo Navegacional, são definidos *profiles* da UML que permitem estender a notação para incorporar novos aspectos por meio de estereótipos. Um estereótipo possibilita estender a semântica para adicionar restrições. Para o Modelo de Apresentação, é utilizado um tipo particular de Diagrama de Classes [01].

A Figura 4.19 exibe a dependência entre os modelo utilizado em UWE; nesse caso, percebesse que o Modelo Conceitual precede o Modelo Navegacional que por sua vez precede o Modelo de Apresentação, sendo que os três necessitam da análise dos casos de uso independente do modelo que esteja sendo criado.

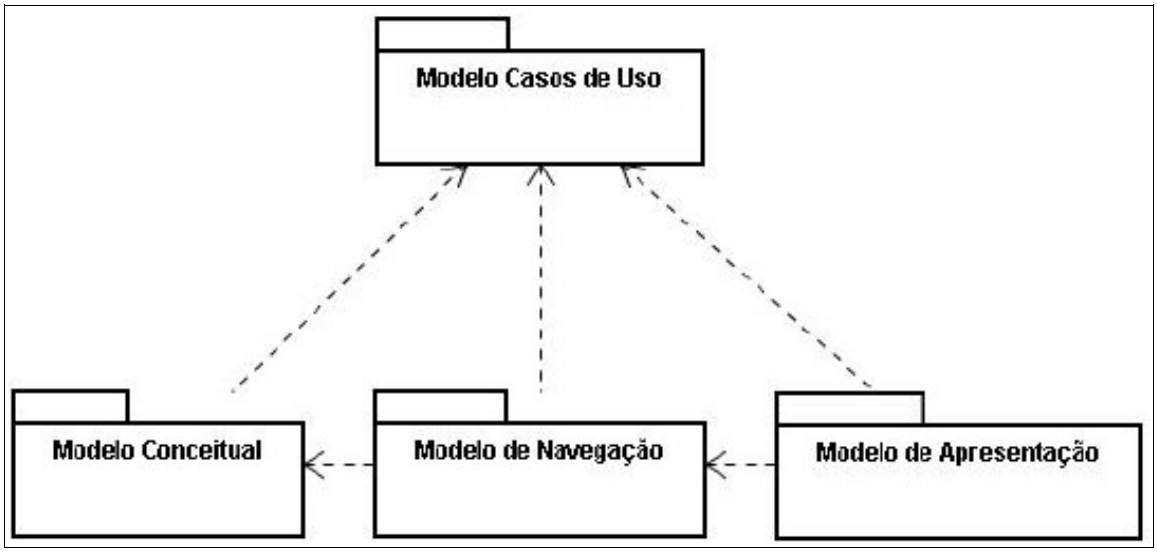


Figura 4.19: Modelos da UWE (Fonte: [01])

4.4.1 Modelo Conceitual

No Modelo Conceitual, as classes e os objetos participantes do produto de software e as relações entre eles são modelados. Nesse caso, são os objetos que estão envolvidos nas atividades que os usuários executarão com a aplicação. Assim, considera-se os objetos que são entrada ou resultado relevante para a atividade, tendo atenção para não incluir as atividades de navegação, apresentação e de interações entre elas [27 e 28].

Exatamente como nos produtos de software tradicionais orientados a objeto, a modelagem é realizada da mesma forma, ou seja, encontrar as classes, definir estrutura de herança, especificar restrições, etc. Elementos de modelagem usados no Modelo Conceitual são: classes conceituais, pacotes e associações. Pacotes e associações são as utilizadas na UML não estendida [29].

A Figura 4.20 mostra um exemplo de Modelo Conceitual de uma Biblioteca *Online*, que é limitado aos dados de funcionamento, embora muitos outros aspectos possam ser anexados devido ao processo incremental e iterativo. Tais aspectos poderiam ser "*search engines*[2]" [27].

[2] São consideradas estrutura de busca robusta que facilitam navegação.

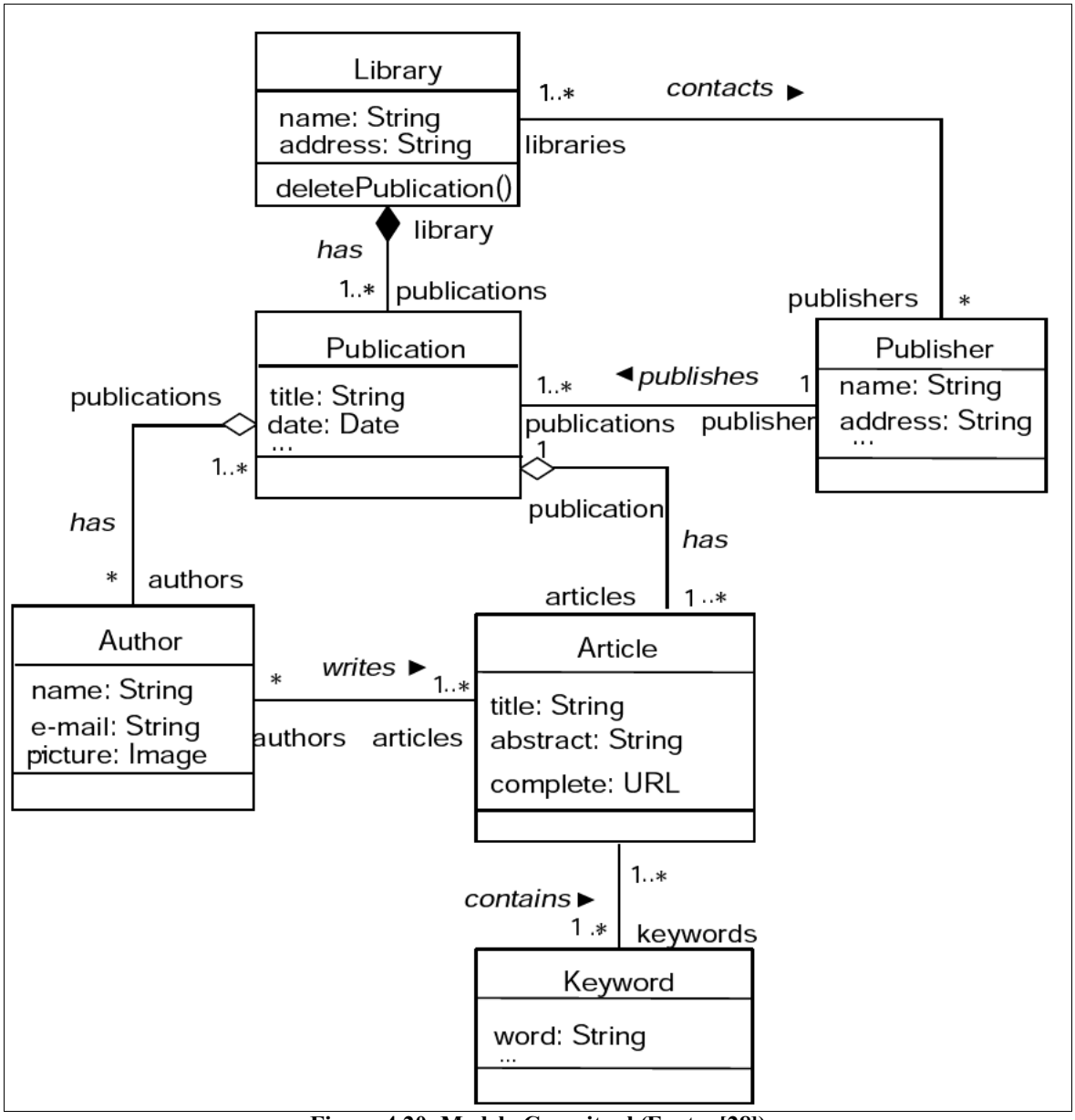


Figura 4.20: Modelo Conceitual (Fonte: [28])

4.4.2 Modelo Navegacional

O Modelo Navegacional é constituído de dois sub-modelos: i) Modelo Navegacional de Espaço; e ii) Modelo Navegacional de Estrutura.

O Modelo Navegacional de Espaço especifica quais objetos ou classe de objetos podem ser alcançados por navegação através da aplicação. Tais objetos são aqueles apresentados no modelo conceitual, mas nem todos farão parte do Modelo Navegacional de Espaço. As classes recebem o estereótipo de Classes de Navegação ou *navigation class* [26 e 29].

A Figura 4.21 apresenta o Modelo Navegacional de Espaço de uma Biblioteca *Online,* o qual mostra o comportamento de navegação das classes *Library, Publication, Author* e *Article*. Percebe-se que as classes *Publication* e *Article* sofreram alterações em relação ao Modelo Conceitual, a classe *Publisher* foi anexada a classe *Publication* e a classe *Keyword* foi anexada a classe *Article.* Outro aspecto é a presença de acesso navegacional, por exemplo a classe *Author* é alcançada por navegação pela classe *Library.*

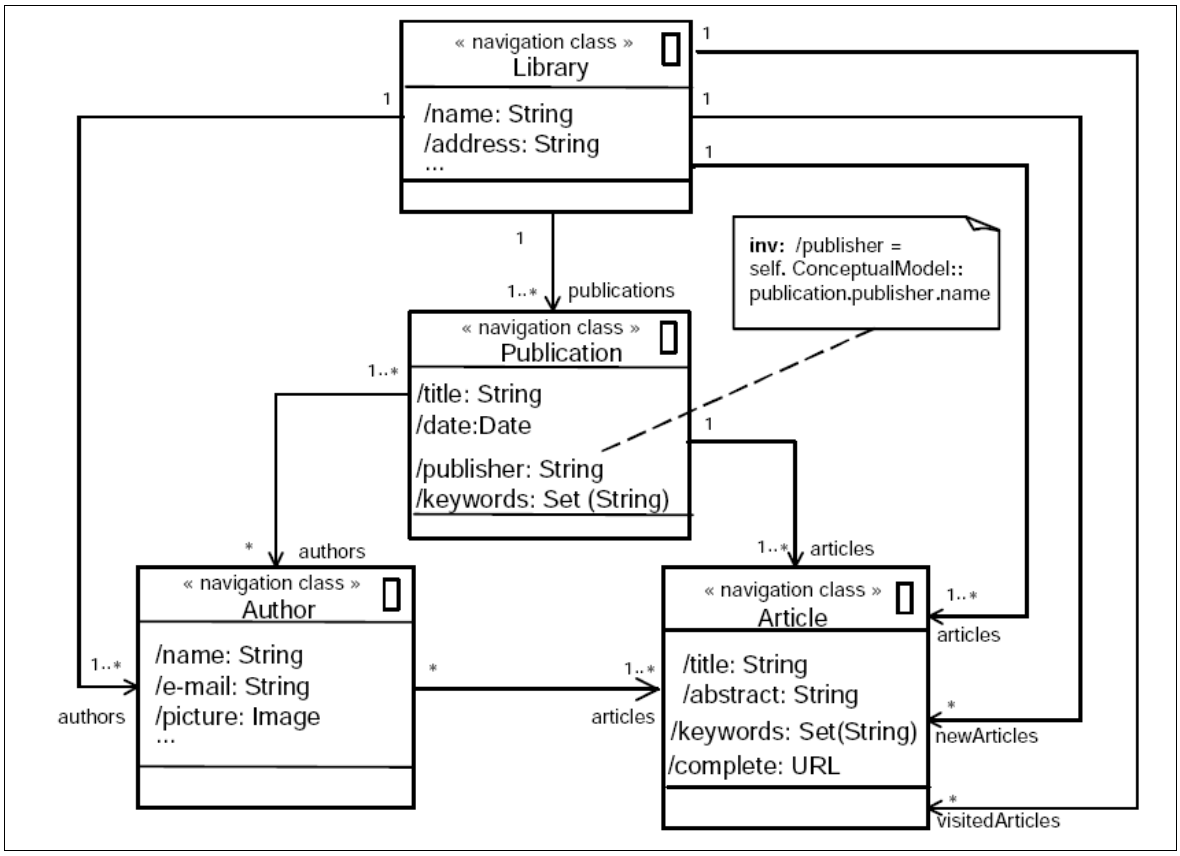


Figura 4.21: Modelo Navegacional de Espaço (Fonte: [28])

O Modelo Navegacional de Estrutura define como estes objetos navegacionais são alcançados pela navegação. Ele é construído com base no Modelo Navegacional de Espaço e consiste em aumentar o Modelo Navegacional de Espaço com índices, guias de navegação, consultas e *menus* [29].

As estruturas para a representação visual do Modelo Navegacional são: as classes de navegação, *index, guided tour*, *query* e *menu* [30].

- Classe de navegação, modela uma classe cujas as instâncias sejam visitadas pelo usuário durante a navegação. As classes da navegação recebem o mesmo nome das classes conceituais correspondentes. Para sua representação, é usado o estereótipo

navigation class conforme apresentado na Figura 4.22

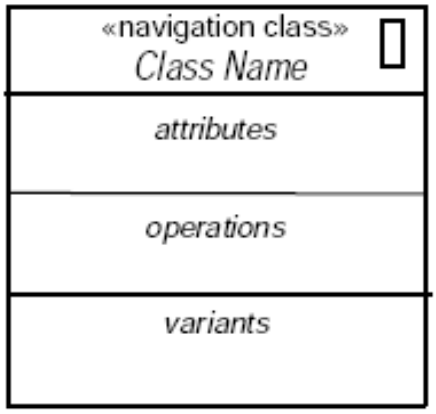


Figura 4.22: Exemplo de Classe de Navegação

- *Index*, permite o acesso direto a instância de uma classe da navegação. Ou seja, modela um objeto que possui inúmeros itens de navegação (*link),* que por sua vez representam o nome de uma instância ou de um *link* para uma classe de navegação. Para classes desse tipo, usa-se o estereótipo <<*index*>> e seu ícone correspondente. A Figura 4.23 ilustra a versão simplifica do uso de *Index*;

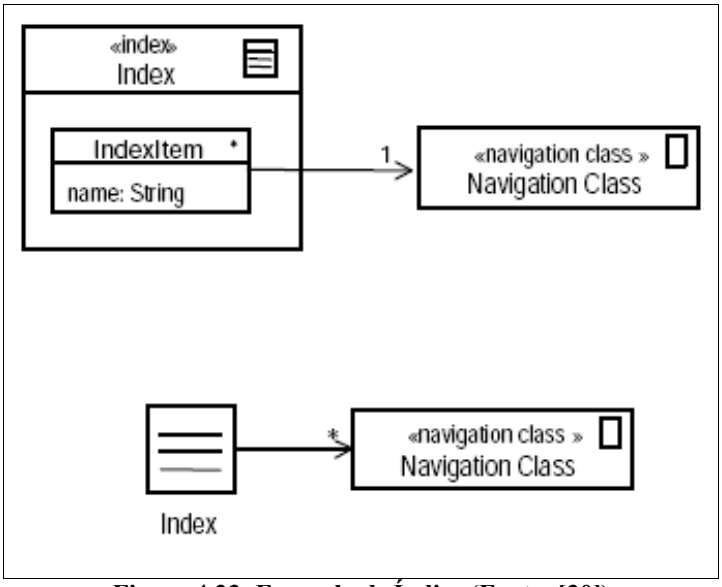


Figura 4.23: Exemplo de Índice (Fonte: [30])

- G*uided Tour*, fornece o acesso seqüencial às instâncias de uma classe de navegação. Para as classes, que contêm objetos G*uided Tour*, usa-se o estereótipo <<*guidedTour*>> e seu ícone correspondente. Qualquer classe G*uided Tour* ou item deve ser conectado a uma classe da navegação. As classes G*uided Tour* podem ser controladas pelo usuário ou pelo sistema. A Figura 4.24 apresenta a versão simplifica

do uso da G*uided Tour;*

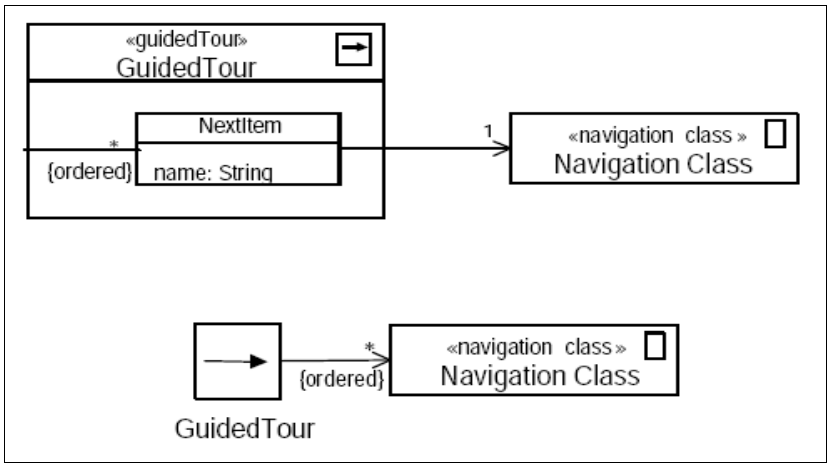


Figura 4.24: Exemplo de Visita Guiada (Fonte: [30])

- *Query*, é a representação de um classe que tem como atributo uma aplicação de busca. Para classes desse tipo, usa-se o estereótipo <<*query*>> e seu ícone correspondente. Como mostrado na Figura 4.25A, qualquer classe *Query* é a fonte de duas associações dirigidas relacionadas pela expressão {*xor*}. Os resultados de uma classe Q*uery* podem alternativamente ser usados como entrada para uma G*uided Tour* conforme mostrado na Figura 4.25C. A Figura 4.25B é a representação simplificada para *Query*;

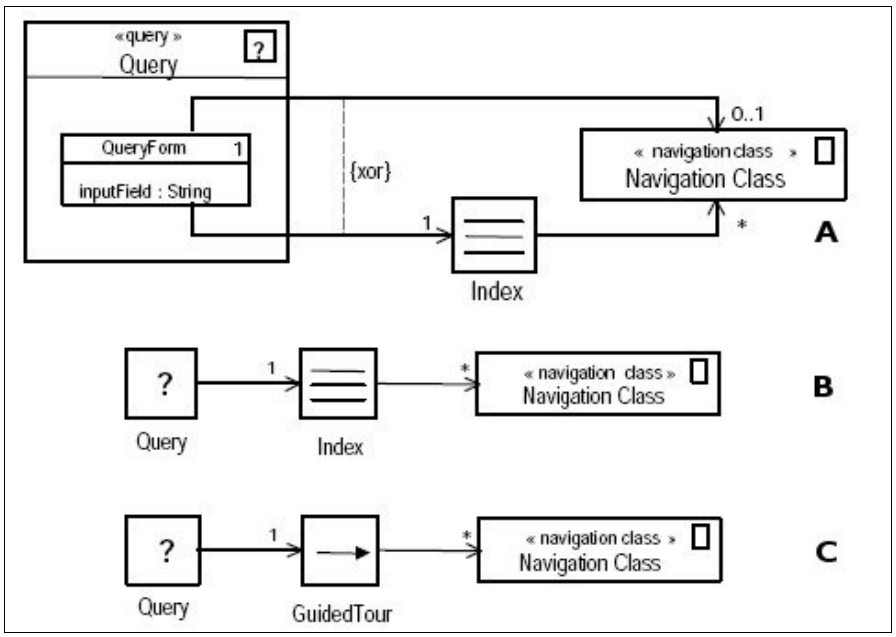


Figura 4.25: Exemplo de Interrogação (Fonte: [30])

- Um *Menu* é um conjunto de elementos heterogêneos em uma quantidade fixa, tais como um *Index,* uma G*uided Tour*, uma *Query*, uma instância de uma classe da navegação ou de um outro *Menu.* Cada elemento de *Menu* tem um nome constante e

possui uma ligação a uma instância de uma classe da navegação ou a um elemento do acesso. A classe de *Menu* é estereotipada por <<*menu*>> com um ícone correspondente conforme apresentado nas Figura 4.26.

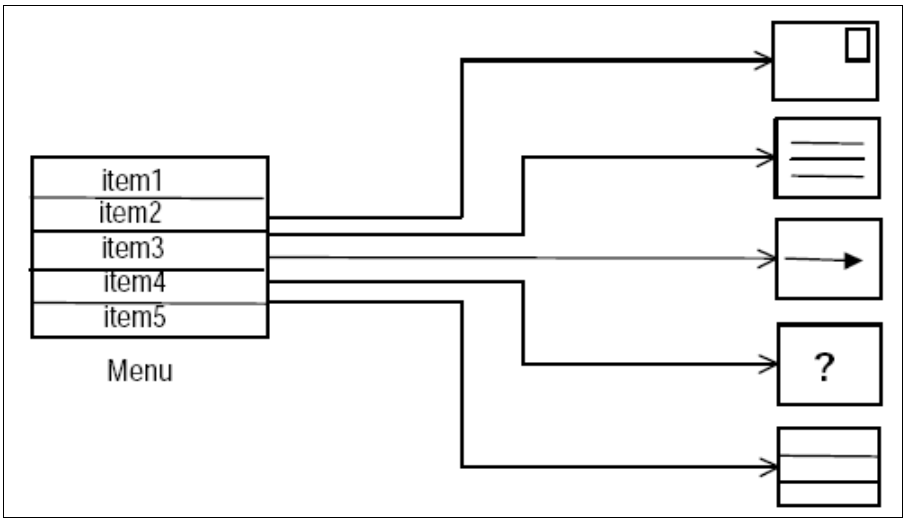


Figura 4.26: Exemplo de Menu (Fonte: [30])

No Modelo Navegacional, a estrutura de navegação de uma aplicação *WebApp* é modelada. Para cada classe conceitual relevante para a navegação no Modelo Conceitual, existe uma classe de navegação. Associações entre classes de navegação são adicionadas se suas classes conceituais são conectadas umas às outras por associações no Modelo Conceitual. Uma associação deve ser adicionada para representar os caminhos de navegação de uma classe de navegação para outra. Além disso, acessos primitivos são adicionados para modelar as possibilidades para o usuário poder navegar na aplicação. Elemento de navegação é a forma genérica da classe de navegação. Cada elemento de navegação representa um nó da navegação *WebApp* [29].

A Figura 4.27 é a representação gráfica do Modelo de Navegação para a aplicação da Biblioteca *Online*; esse exemplo demonstra a navegação entre as classes e os recursos utilizados para essa navegação. Por exemplo a classe *Libary* por meio do elemento *Menu* "*news*" acessa a estrutura *Index "NewArticle"* que por navegação acessa a classe *Article.*

4.4.3 Modelo de Apresentação

No Modelo de Apresentação, a estrutura de uma apresentação da aplicação *WebApp* é modelada. Para cada elemento apresentável no Modelo de Apresentação, existe uma classe de apresentação. Classes de apresentação podem ser colocadas em frames e itens de apresentação, como Textos, Imagens e *Links*, e devem ser adicionados às classes de

apresentação [29].

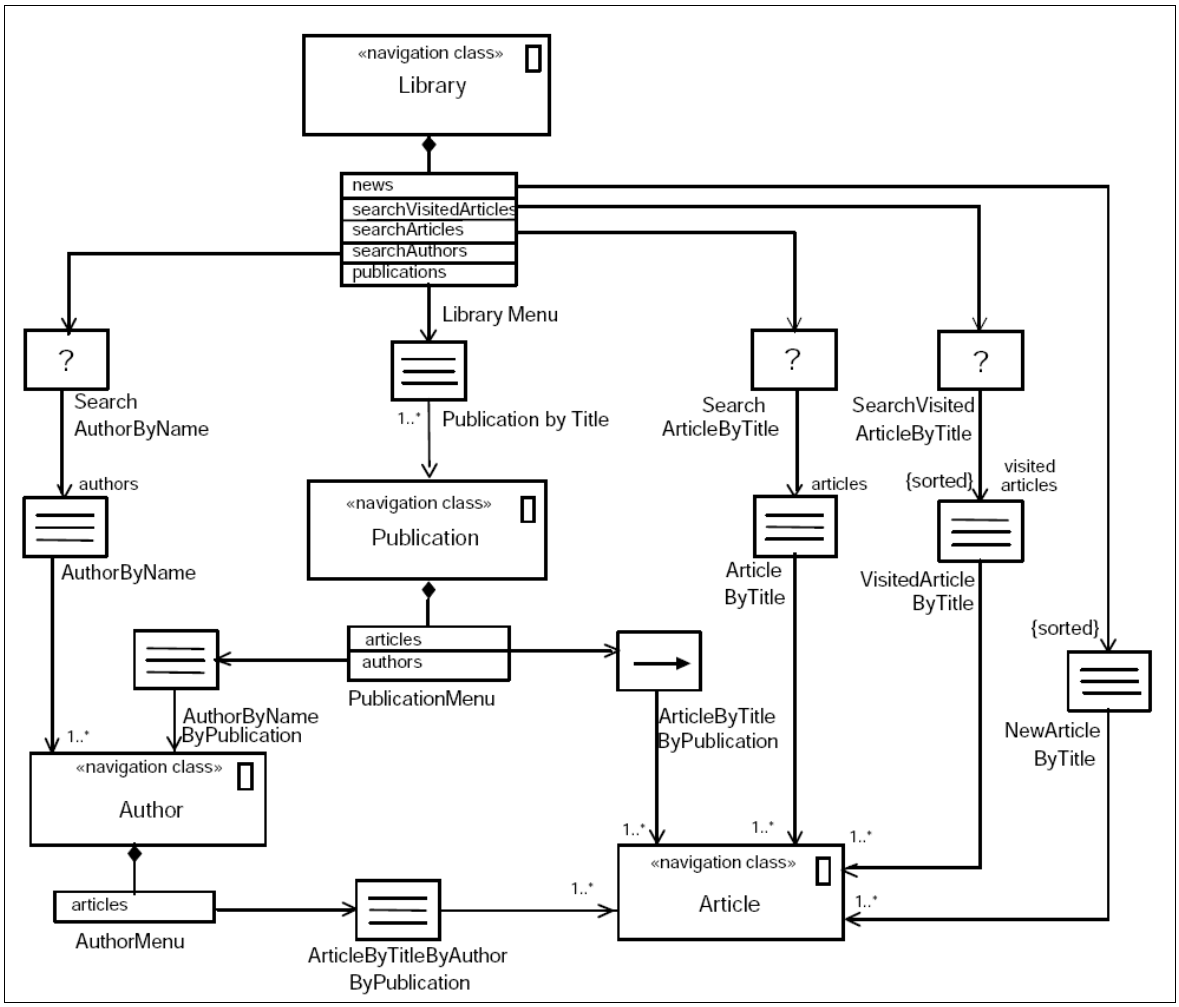


Figura 4.27: Modelo Navegacional (Fonte: [28])

O Modelo de Apresentação é a representação de "onde" e de "como" os objetos da navegação e os acessos primitivos são apresentados para o usuário. O projeto da apresentação suporta a transformação do Modelo Navegacional de Estrutura em um conjunto de modelos que mostram o local estático dos objetos visíveis para o usuário, o esquema de representação destes objetos (páginas no projeto da aplicação *WebApp*) e dos seus comportamentos dinâmicos. O esquema da representação é similar à técnica de desenvolvimento usada por alguns projetistas de interface. O projeto de apresentação focaliza a organização estrutural da apresentação, por exemplo, textos, imagens, formulários e *menus*, e não a aparência física em termos de formatos especiais, cores, etc. Deste modo, decisões são tomadas durante o desenvolvimento de um protótipo da interface ou em uma fase de implementação [29].

A Figura 4.28 mostra a classe de apresentação "*Publication"* relativa a Biblioteca *Online*. Os elementos de interface utilizados na sua implementação são apresentados, sendo do tipo texto (*Title* e *Date)*, botão (*Complete Publication),* coleção de âncoras *(Authors* e *Articles)* e índices (*keyword)*.

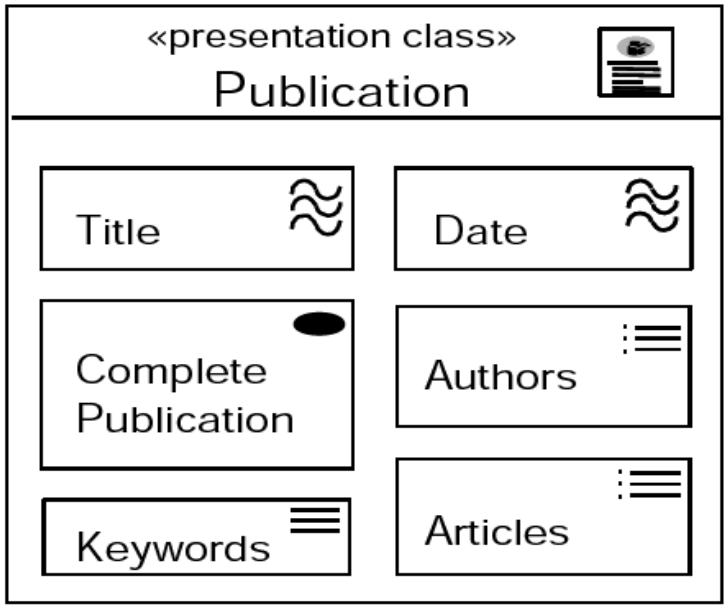


Figura 4.28: Exemplo de Classe de Apresentação (Fonte: [28])

As estruturas para o modelo de apresentação são: *user interface view, presentation class* e *user interface element*, conforme mostrado na Figura 4.29, [30]:

- Uma *user interface view* é uma classe tipo *container* que agupa os elementos abstratos de interface apresentados simultaneamente ao usuário. Para este tipo de classe, é usado o estereótipo *<<UI view>>*;
- Uma *presentation class* é uma unidade estrutural que serve para agrupar em uma *interface view* os elementos de interface utilizados pelo usuário. Para esse tipo de classe, usa-se o estereótipo *<<presentation class>>*;
- Uma *user interface element* é uma classe abstrata que possui diversos elementos que descrevem elemento particulares de uma interface. Por exemplo, as classes com estereótipo *<<text>>*, *<<image>>*, *<<video>>*, *<<audio>>*, *<<anchor>>* e *<<form>>* são subclasses de elementos de *UI element* para modelar os textos, as imagens entre outros.

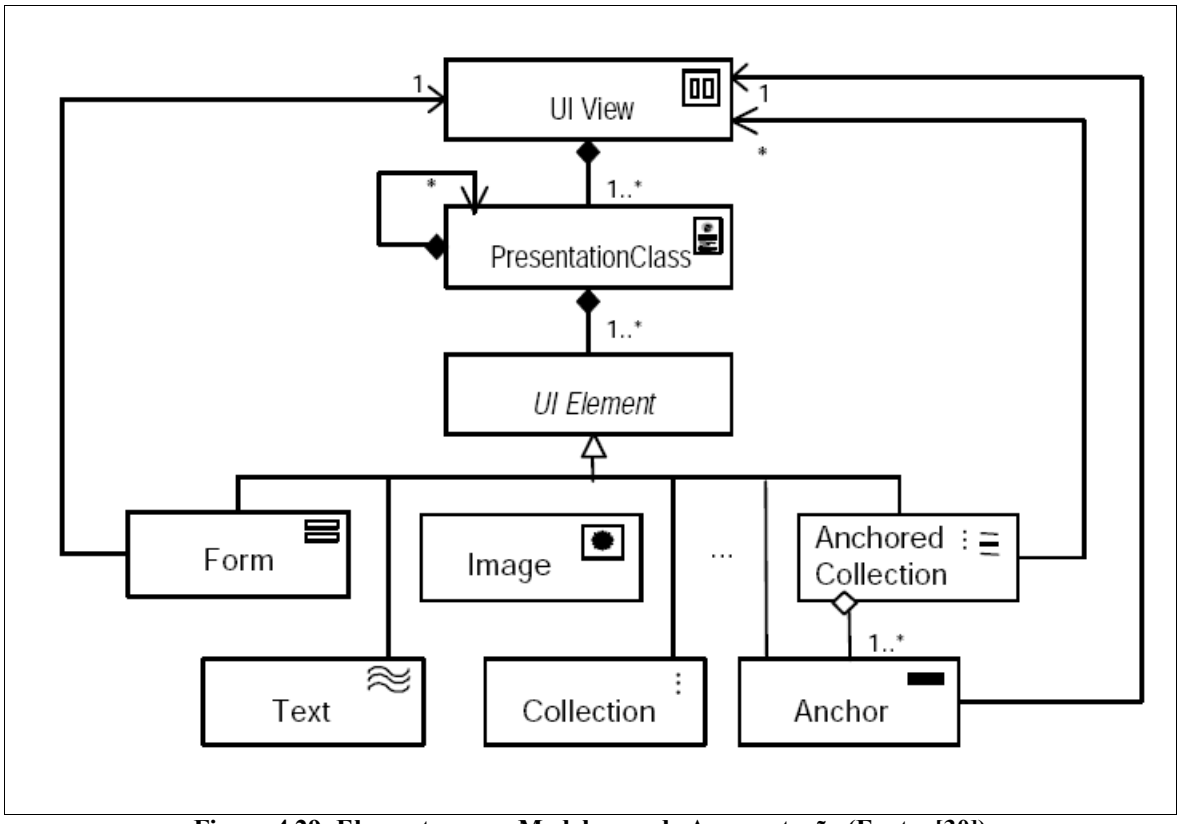


Figura 4.29: Elementos para Modelagem de Apresentação (Fonte: [30])

A Figura 4.30 exibe o Modelo de Apresentação para a Biblioteca *Online*. O modelo mostra a navegação entre as classes de apresentação e qual elemento de interface gerou essa navegação.

4.5 Comparação entre as Abordagens de Modelagem

Os modelos conceituais de cada abordagem de modelagem não apresentam grandes diferenças pois seus focos estão em modelar os dados relativos a construção da implicação, ou seja, as entidades que farão parte do produto. Nesse ponto, as três abordagens de modelagem seguem o mesmo conceito do uso do Diagrama de Classes da UML para modelagem de software tradicional.

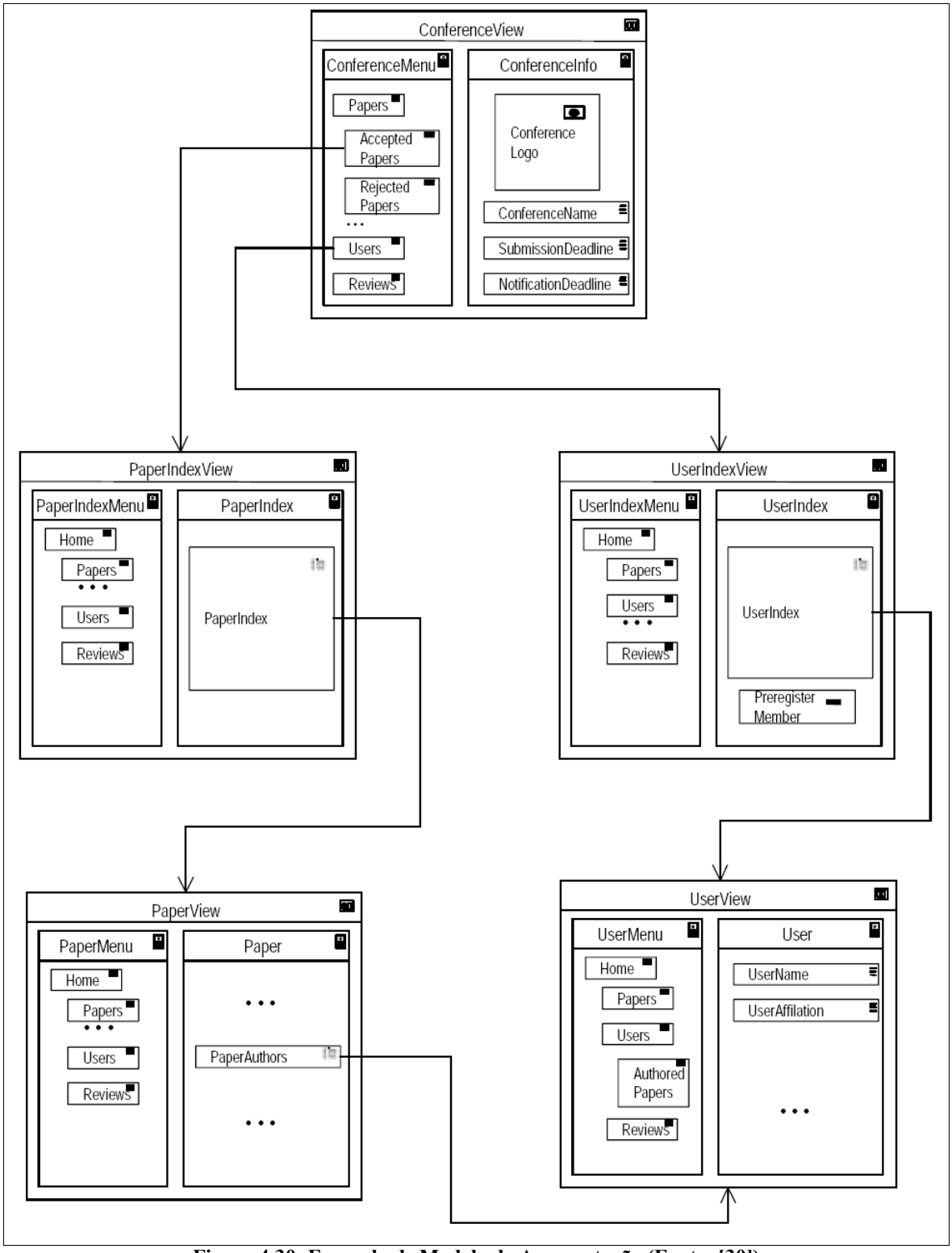


Figura 4.30: Exemplo de Modelo de Apresentação (Fonte: [30])

A Tabela 4.3 mostra que OOHDM e UWE têm foco de apresentação em classes enquanto WebML basea-se no uso de terminações (*units)* para elementos de página *Web.* O

foco de OOHDM é no contexto que os elementos estão inseridos o que leva a dependência da visão do projetista para com a implementação e UWE trabalha com classes que possuem estereótipos definidos.

Tabela 4.3: Resumo dos Modelos de Navegação

	Modelo Navegacional	
	Objeto Navegação (quais?)	**Estrutura Navegação (como?)**
OOHDM	Classes Navegacionais Consiste: Classes de Navegação Classes Básicas • Nós • Elos • Âncora	Contextos Navegacionais Consiste: • Classes Contexto • Classes em Contexto • Elos de Contexto
WebML	*Units*	Links Consiste: *Links* Contextuais *Links* Não-Contextuais *Links* Automáticos *Links* de Transporte
UWE	Modelo Navegacional de Espaço Consiste: Classe de objeto e associações + estereótipo *navigation class*.	Modelo Navegacional de Estrutura Consiste: Expansão do modelo navegacional de espaço com: índices, guias de navegação, consultas e *menus*.

Na Tabela 4.4, pode-se verificar que o Modelo de Interface de OOHDM possui maior formalismo de regras para sua construção, enquanto UWE utiliza classes que representam estrutura de interface. WebML não apresenta uma definição concreta e utiliza a linguagem XML como uma linguagem de notação.

Tabela 4.4: Resumo dos Modelos de Interface

	Modelo de Interface	
	Descrição	**Foco**
OOHDM	Utiliza o formalismo da ADV.	Na atividade ou ação de cada elemento visual para com a implementação.
WebML	Não possui uma definição própria. Utiliza XML para documentação.	Na documentação de interface.
UWE	Utiliza técnica de desenvolvimento usada por projetistas de interface através de classes de apresentação.	Navegação visual da implantação.

4.6 Considerações Finais

Utilizando-se OOHDM, é possível desenvolver projetos modulares e de fácil manutenção, pois o Modelo Conceitual, o Modelo Navegacional e o projeto de interface são tratados como atividades separadas, permitindo concentrar-se em diferentes interesses, resolvendo um a um e retornando para cada etapa construída, quando for necessário. Com isso, obtêm-se estrutura para o raciocínio sobre o processo do projeto e experiência específica para cada atividade. OOHDM oferece independência na escolha de linguagens e ambientes de programação fazendo com que a escolha do projeto seja mais ampla, pois pode-se trabalhar com o paradigma de orientação a objetos e com o paradigma de programação estruturada ou as duas ao mesmo tempo.

As especificações de WebML são independentes, seja pelo lado do cliente que usa linguagens para desenvolver aplicações para usuário, quanto pelo lado do servidor. Desta forma, após serem realizadas as etapas de um projeto em WebML, a facilidade de geração automática da aplicação a partir dos diagramas construídos é concisa e rápida, podendo utilizar uma gama de ferramentas CASE.

UWE tem uma maior referência ao desenvolvimento de aplicações para a *Web* nas suas etapas e tem como foco principal a definição de um projeto sistemático, a personalização e a geração semi-automática.

Estes métodos evidenciam que, para desenvolver aplicações *Web*, é possível e desejável trabalhar com os três modelos principais: Modelo Conceitual, Modelo Navegacional e Modelo de Interface. Cada método possui formalismos específicos para cada um desses modelos, tratando-os separadamente.

Embora a abordagem para o Modelo de Interface de cada um desses métodos seja diferente, eles reconhecem a importância de separá-lo dos outros modelos, visto que, nesta etapa, deve-se representar as trocas de informações que irão ocorrer entre o usuário e a aplicação. Essas trocas de informações são obtidas durante o desenvolvimento do Modelo Conceitual e do Modelo Navegacional.

5 INTEGRAÇÃO DAS ABORDAGENS DE MODELAGEM ESTUDADAS

5.1 Considerações Iniciais

A MAW (Modelagem para Aplicações *WebApps*) é o resultado do estudo realizado nesta monografia, cujo foi o desenvolvimento de uma abordagem de modelagem híbrida para a construção de produtos de software para *Web*. Essa abordagem de modelagem foi proposta mediante a análise de três abordagens de modelagens amplamente conhecidas na literatura: OOHDM, UWE e WebML.

O desenvolvimento de MAW teve como foco abstrair das abordagens de modelagens partes relevantes e consistentes, que em conjunto formassem um todo com características mais interessantes ao desenvolvimento de *WebApps*.

5.2 Desenvolvimento da Proposta da MAW

A MAW baseada nas abordagens de modelagens descritas no capítulo anterior, foi realizada no seguinte contexto:

- Para cada abordagem de modelagem, foram estabelecidos três submodelos com características próprias ao segmento:

  ➔ Foco no Conteúdo. O objetivo é modelar as classes e os objetos participantes do produto de software, ou seja, aqueles que são entradas ou resultados relevantes para a atividade (Modelo Conceitual);

  ➔ Foco na Navegação. A finalidade é modelar quais e como os objetos podem ser alcançados por navegação através da aplicação, ou seja, apresentar uma visão navegacional dos objetos com foco na navegação (Modelo Navegacional);

  ➔ Apresentação de Interface. O objetivo é modelar como os objetos navegacionais são percebidos pelo usuário, ou seja, quais e de que modo os objetos de navegação serão apresentados no *site view* (Modelo de Interface).
- Em seguida, foram verificadas as necessidades de adequação de cada abordagem de modelagem para a segmentação proposta no item anterior, ou seja, verificar a segmentação que cada abordagem de modelagem possui e agrupá-las conforme os focos propostos;

- Finalmente, cada submodelo foi comparado para definir qual é o mais adequado para ser utilizado na construção do MAW.

5.3 Adequação das Abordagens de Modelagem

A adequação das abordagens de modelagem se fez necessária, pois houve a necessidade de padronização dos seus submodelos usados para formulação do MAW.

A UWE não precisou de adequação, pois seus submodelos estão segmentados conforme proposto na seção 5.2.

A OOHDM também apresenta segmentada conforme proposto pela seção 5.2, com a exceção do submodelo Implementação, que foi desconsiderado, uma vez que seu foco trata apenas da tradução dos modelos para alguma linguagem de aplicação.

WebML, no entanto, sofreu as seguintes adaptações:

- Os modelos hipertexto-composição e hipertexto-navegacional foram considerados como Modelo Navegacional, pois possuem características para modelar como e quais informações farão parte da aplicação, o primeiro modela o tipo de objeto navegacional e o segundo modela como eles se relacionam;
- O Modelo de Apresentação e o Modelo de Personalização foram considerados como Modelo de Interface, pois tratam elementos visualizados pelo usuário.

A Tabela 5.1 mostra um resumo de como ficou a adequação das abordagens de modelagem que configurarão o MAW.

Tabela 5.1: Adequação dos Modelos

MAW	OOHDM	WebML	UWE
Modelo Conceitual	Modelo Conceitual	Modelo de Dados	Modelo Conceitual
Modelo Navegacional	Projeto de Navegação	Modelo Hipertexto-Composição + Modelo Hipertexto-Navegação	Modelo Navegacional de Espaço e Modelo Navegacional de Estrutura
Modelo de Interface	Projeto de Interface Abstrata	Modelo de Personalização + Modelo de Interface	Modelo de Apresentação

5.4 Modelagem de Aplicações *WebApps* – MAW

5.4.1 Comparação e Escolha dos Submodelos das Abordagens de Modelagem

O objetivo desta seção é apresentar como e quais submodelos das abordagens estudadas foram escolhidas para formar o MAW. Além disso, serão referenciados os artefatos presentes em cada um desses submodelos.

5.4.1.1 Modelo Conceitual

O Modelo Conceitual funciona como um repositório a partir do qual serão construídas diferentes visões navegacionais do domínio do problema, para qual opta-se por um modelo com foco nos dados que utilize a UML para notação dos objetos conceituais.

O Modelo Conceitual independe das modelagens visualizadas nesse trabalho, pois UWE, OOHDM e WebML tratam esse modelo de forma parecida, utilizando o Diagrama de Classe da UML com algumas extensões pré-definidas; sendo assim, o único artefato gerado por esse modelo.

A importância desse artefato está no sentido de que, a partir dele, será possível gerar as diferentes visões de navegação para o Modelo Navegacional e para decisões relacionadas ao tipo e à estrutura de acesso a dados que serão utilizados para aplicação *WebApp*.

5.4.1.2 Modelo Navegacional

Para o Modelo Navegacional, foi utilizado o Modelo de Navegação de UWE, pois, nesse caso, ele se apresenta mais estruturado semanticamente na forma que os objetos de navegação são apresentados em relação ao outros modelos:

- UWE apresenta estereótipos que definem os objetos navegacionais na sua forma extendida e na abreviada, o que impede ambigüidade de conceitos, além de ter um padrão facilmente reconhecível pela equipe;
- UWE trata os objetos navegacionais com uma estrutura atômica, enquanto OOHDM e WebML inserem o conceito de contextos, que podem variar de acordo com o projetista;
- Dividido em duas etapas (Modelo Navegacional de Espaço e Modelo Navegacional de Estrutura), o Modelo de Navegação da UWE possibilita níveis de detalhamento, simplificando os modelos para compreensão da navegação seja ela na relação

navegacional de espaço para navegacional de estrutura ou em várias visões do navegacional de estrutura.

O Modelo Navegacional adotado apresenta como artefatos o Modelo de Espaço e o Modelo de Estrutura, sendo este último será reavaliado para construção do Modelo de Interface. A importância destes artefatos são em relação a decisões de lógica de negócio, por exemplo, em um sistema *Web* poderia possui restrições de acesso a certas páginas do *site*, nas quais seria necessária autenticação por senha e *login*.

5.4.1.3 Modelo de Interface

Para o Modelo de Interface, é sugerido o uso do Modelo de Interface Abstrata de OOHDM, pois utiliza o formalismo da *Abstract Data Views* (ADV) que possibilita independência do projeto para com a implementação e oferece reusabilidade de componentes ligados ao projeto e a interface *Web*. ADV possui regras e formalismo para o desenvolvimento de interfaces para o usuário bem definidos desde a apresentação do objetos gráficos até os eventos que esses promovem. Devido às duas características semânticas, a ADVs permite visualização de recursos dinâmico (vídeo, áudio, etc) promovendo interpretação do que será apresentado em uma aplicação *WebApps*.

Em relação aos demais modelos, foi visto que a WebML não possui formalismo próprio para especificação da interface, existindo apenas uma classificação de entidades que podem ser usadas para melhorar os recursos gráficos e o uso de XML para documentação. UWE apresenta apenas o recurso de modelagem por classes de apresentação, na qual outros itens são agregados para especificação dos recursos gráficos e a navegação entre eles, mas não deslumbra a ação dinâmica das estruturas.

O artefato gerado é a ADV*charts*, cujo objetivo é mostrar graficamente os objetos e as ações que integraram a interface da aplicação *WebApps*. Sua importância está na fase de implementação do produto *WebApps,* pois as tecnologias de desenvolvimento para interface *Web* levam em consideração o tipo de dados e a estrutura a serem apresentados ao usuário.

5.4.2 Transição entre os Submodelos do MAW

A transição entre os submodelos (Moldeo Conceitual – Modelo Navegacional – Modelo de Interface) consiste na transposição dos objetos conceituais para navegacionais e destes para representações de estruturas de interface.

As informações contidas no Modelo Conceitual são análogas aos conteúdos de um Diagrama de Classe da UML, ou seja, atributos que caracterizam o objeto conceitual, a cardinalidade dos relacionamentos e os estereótipos.

A transição do Modelo Conceitual para o Modelo Navegacional se faz com o desenvolvimento de visões de como se dará a navegação de um objeto ou grupo de objetos conceituais apresentados pelo Diagrama de Classe do Modelo Conceitual.

Nessa fase, o objeto conceitual recebe o estereótipo navegacional, entretanto isso não significa que eles são iguais, pois em certo casos dois ou mais objetos conceituais concebem um objeto navegacional. A conexão entre os objetos navegacionais são feitas agora de forma direcionada e/ou passando por estruturas de navegação (*index*, *guided tour*, *query* e *menu*). Também, nessa fase, são gerados o Modelo Navegacional de Espaço e o Modelo Navegacional de Estrutura.

Pode-se verificar que em um sistema de *Login*, no seu Modelo Conceitual existem dois objetos conceituais que fazem parte do Diagrama de Classe: usuário e tipo de usuário. No Modelo Navegacional, apenas objeto navegacional "usuário do sistema" é apresentado, o qual fará parte do Modelo Navegacional de Espaço e do Modelo Navegacional de Estrutura. Porém, no Modelo Navegacional de Estrutura, podem existir diversas visões de acesso (administrador, usuário 1 e usuário 2).

Com relação a passagem dos elementos do Modelo Navegacional (pertence a UWE) para o Modelo de Interface (pertence a OOHDM), não houve problemas com conexão, pois o Modelo de Interface é baseado em ADV que possui métodos independente da implementação. No Modelo de Interface, as informações são relacionados de acordo com o modo que o usuário irá interagir com o aplicação *WebApp*. Nesse caso, o foco está em determinar a aparência interfacial de cada objeto que será navegado, os eventos externos que afetam a navegação e as diversas funções da aplicação. Os *ADVcharts* gerados nesse processo têm como característica expor de forma gráfica a navegação (ativa ao usuário) apresentado pelo Modelo Navegacional de Estrutura.

5.4.3 Visão Geral de MAW

A conexão entre os submodelos de MAW pode ser vista no diagrama apresentado pela Figura 5.1. As entradas para o Modelo Conceitual são o dados manipulados e apresentados pela aplicação *WebApps*. As saídas do Modelo de Interface serão as interfaces

que, por intermédio do usuário, interagirão com os dados e retornarão o resultado dessa interação.

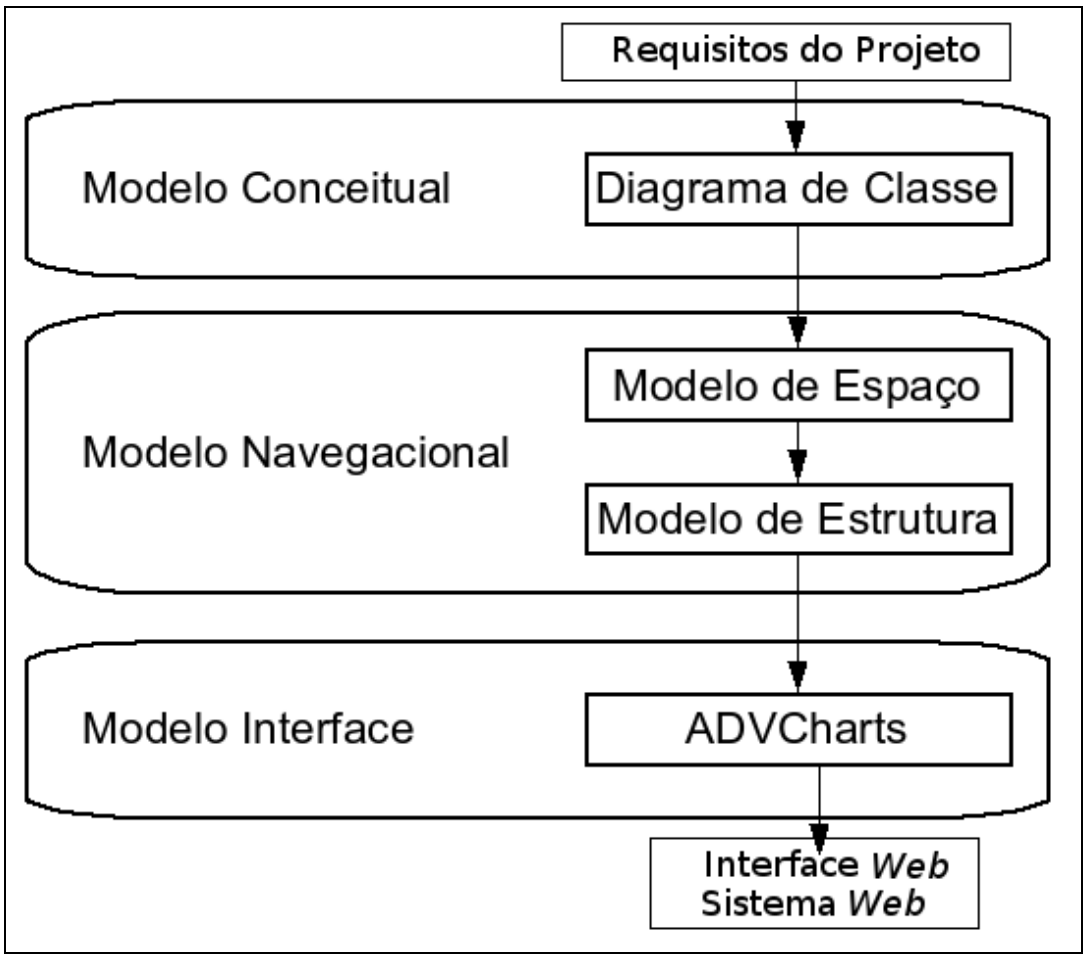


Figura 5.1: Diagrama de Conexão dos Submodelos

A Tabela 5.2 apresenta a estrutura do MAW, na qual as áreas sombreadas mostram os submodelos utilizados para sua formulação, onde se pode notar que ele foi constituído pelas abordagens OOHDM e UWE. Outro ponto é em relação ao Modelo Conceitual, pois é indiferente a adoção do submodelo de uma das abordagens de modelagem.

Tabela 5.2: Modelo MAW

MAW	OOHDM	WebML	UWE
Modelo Conceitual	Modelo Conceitual	Modelo de dados	Modelo Conceitual
Modelo Navegacional	Projeto de Navegação	Modelo Hipertexto-Composição+ Modelo Hipertexto-Navegação	Modelo de Navegação de Espaço e Estrutura
Modelo de Interface	Projeto de Interface Abstrata	Modelo de Personalização + Modelo de Interface	Modelo de Apresentação

5.5 Considerações Finais

O desenvolvimento de MAW teve o desafio de identificar as diferenças entre as abordagens de modelagem estudadas, pois elas são, em sua maioria, extensões da UML ou agregação da UML com modelagens de *designer*, existindo mais semelhanças do que diferenças.

Conforme apresentado, as abordagens foram divididas em três módulos para efeito de facilitar o estudo e de promover a construção do modelo híbrido, porém existiram dificuldades na comparação do Modelo de Navegação, pois este apresentou-se semelhante divergindo somente no modo como os objetos de dados de navegação são vistos. Os Modelos de Interface apresentaram características que possibilitaram maior diferenciação entre eles e permitiram uma escolha mais objetiva para MAW.

6 ESTUDO DE CASO – PROJETO RAD

6.1 Considerações Iniciais

Este capítulo apresenta o uso da modelagem MAW proposta nesse trabalho. Assim, foi realizado um estudo de caso, usando MAW para a modelagem de uma parte do projeto RAD (Registro de Atividades de Docentes), que consiste em um sistema de controle de atividades acadêmicas dos docentes da Universidades Federal de Lavras – UFLA.

6.2 Estudo de Caso

O estudo de caso realizado no projeto RAD teve como foco principal a fase de análise e o sub-sistema de *login* de usuário (*login* com a visão de administrador). O objetivo do estudo de caso é apresentar o uso de MAW.

6.2.1 Modelo Conceitual – Subsistema de *Login*

Para modelagem conceitual do subsistema de *login* do projeto RAD, foi considerado que este subsistema teria quatro objetos conceituais: "*Login*", "Cadastro_Usuário", "Grupo_Usuário" e "*Logs*". O objeto conceitual "*Login*" se faz necessário, pois ele é responsável por criar a sessão que permite o acesso ao sistema RAD. Entretanto, o objeto "*Login*" necessita que o usuário seja identificado pelo sistema, sendo essa ação realizada pelo objeto "Cadastro_Usuário". Outra necessidade é cada usuário ter seu nível de acesso dentro do sistema RAD, definido pelo objeto "Grupo-Usuário". Devido a questões de segurança, cada ação realizada no sistema RAD necessita ser identificado, o que remete ao objeto "Logs".

A Figura 6.1 exibe o Modelo Conceitual para o subsistema de *login* no qual, para cada objeto "*Login*", existe apenas um objeto "Cadastro_Usuário" e esse possui vários objetos "Grupo_Usuário". Cada ação realizada pelos objetos "Cadastro_Usuário" e "Grupo_Usuário" geram objetos "*Logs*".

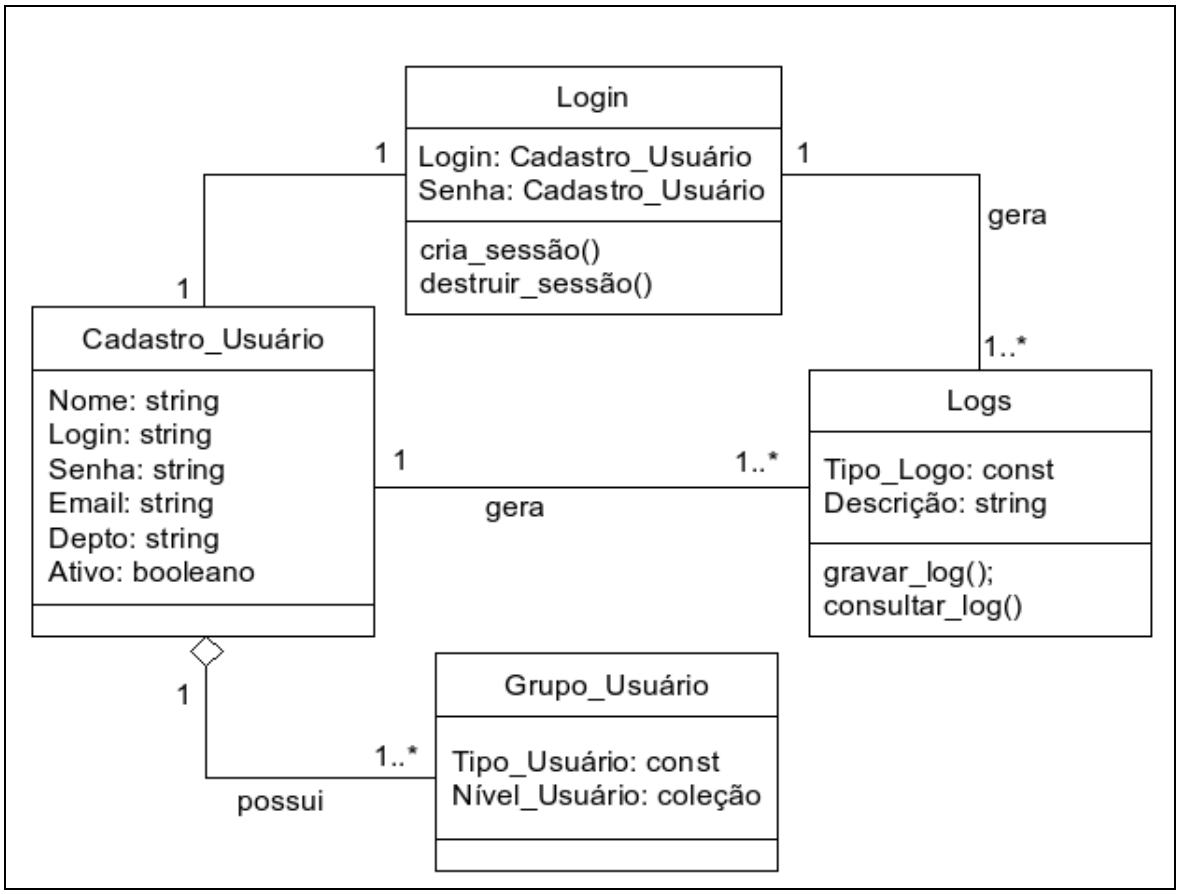


Figura 6.1: Modelo Conceitual *Login* – RAD

6.2.2 Modelo Navegacional – Subsistema *Login*

O Modelo Navegacional do subsistema de *login* apresenta apenas três objetos navegacionais: "*Login*", "Usuário_Sistema" e "*Logs*". Com relação ao objetos navegacionais "*Login*" e "*Logs*", esses permanecem idênticos aos do Modelo Conceitual, entretanto o objeto "Usuário_Sistema" é a fusão do objetos conceituais "Cadastro_Usuário" com "Grupo_Usuário".

Outra mudança está relacionada às associações entre os objetos; no Modelo Navegacional, a associação entre os objetos "*Login*" e "*Log*" desaparece, pois é uma relação de ação interna e não navegacional. A associação entre "Usuário_Sistema" e "*Logs*" também é modificada; no caso do Modelo Navegacional, apresenta-se a ação de acesso, enquanto, no Modelo Conceitual, a ação é de gerar o *log*. Uma nova associação aparece, correspondendo a uma auto-associação do objeto "Usuário_Sistema"

A Figura 6.2 mostra o Modelo Navegacional de Espaço que apresenta quais objetos navegacionais serão utilizados dentro do sistema RAD. A Figura 6.3 ilustra como os objetos navegacionais são navegados, na qual pode-se verificar que o objeto "Usuário_Sistema"

acessa o objeto "*Logs*" por meio de estruturas *menu* e busca e o mesmo se referência através das estruturas *menu*, busca e *index*.

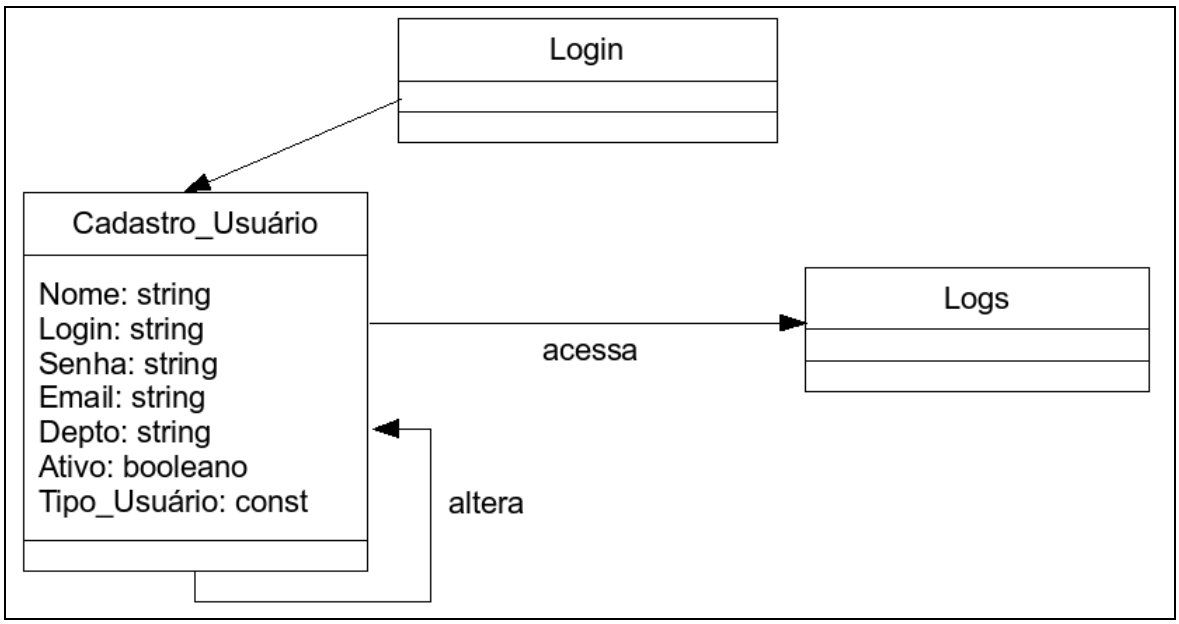


Figura 6.2: Modelo Navegacional de Espaço *Login* – RAD

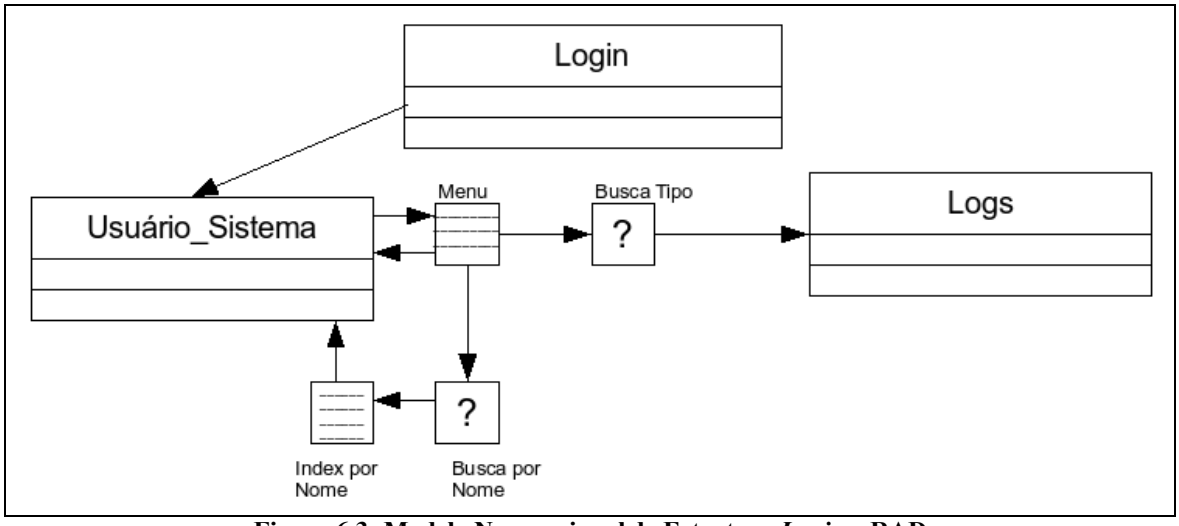


Figura 6.3: Modelo Navegacional de Estrutura *Login* – RAD

6.2.3 Modelo de Interface – Subsistema *Login*

O Modelo de Interface do subsistema de *login* apresenta as estruturas de interface bem como seus eventos. As Figura 6.4 e Figura 6.5 mostram as ADVCharts que apresentam esses elementos:

- A Figura 6.4A demonstrar a interface para conexão ao sistema Rad;
- A Figura 6.4B e a Figura 6.4C mostram a interface do usuário administrador e as funções que estão acessáveis paraesse tipo de usuário;

- A Figura 6.5D exibe a interface para verificar os *logs* existentes e que podem ser filtrados por meio de entradas por formulários;
- A Figura 6.5E ilustra a interface para alteração de dados pessoais;
- A Figura 6.5F mostra a interface de alteração do dados de usuário por intervenção do administrador e que os usuários podem ser localizados por meio de filtros (formulários).

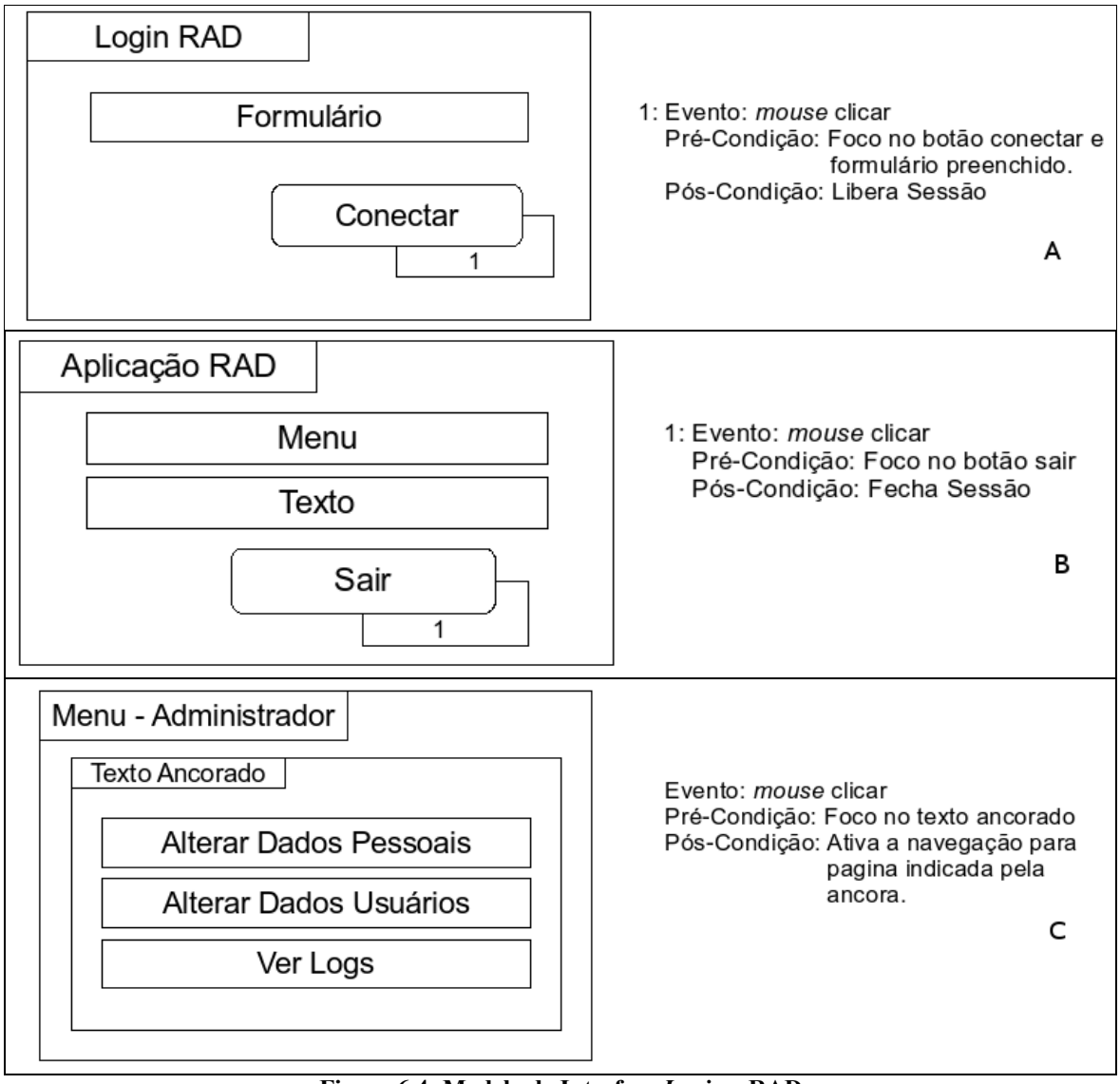


Figura 6.4: Modelo de Interface *Login* – RAD

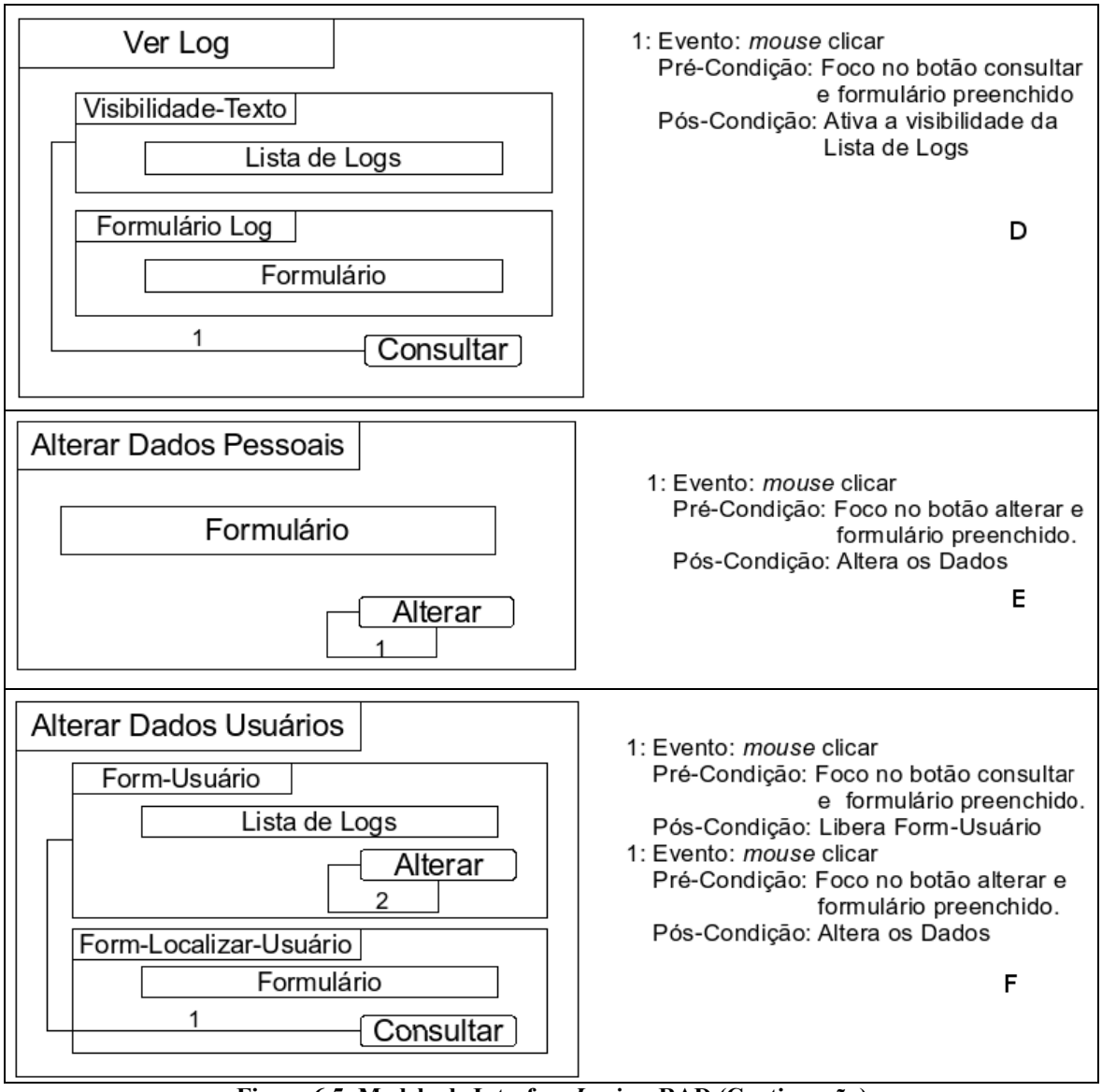


Figura 6.5: Modelo de Interface *Login* – RAD (Continuação)

6.3 Considerações Finais

Considerando que a intenção desse capitulo foi apresentar o uso de MAW em uma aplicação real, foi visto que em um sistema pequeno a complexidade em definir o que é navegação de ação interna do sistema pode ser uma tarefa árdua. Entretanto, após a primeira modelagem, a possibilidade de refinamento dessas decisões tendem a ser facilitadas, pois, ao se ter uma visão melhor do domínio do problema, ele pode ser melhorado em etapas futuras.

7 CONSIDERAÇÕES FINAIS

7.1 Conclusões

O desenvolvimento deste trabalho permitiu a percepção que, apesar do processo de construção de um produto de software comum e uma aplicação para *Web* sejam semelhantes em vários paradigmas, existem complicações estabelecidas pelo dinamismo do contexto em que a *Web* está inserida.

A implicação desse dinamismo fez com que diversas abordagens de modelagem fossem concebidas para esse tipo de desenvolvimento, porém, ao contrário que aconteceu com a UML que se tornou uma convergência de modelos, ainda não há esse movimento para a *Web*.

Foi visto que as modelagens para Engenharia Software *Web* possuem características semelhantes entre elas, mas, ao mesmo tempo, apresentam visões diferentes sobre determinados pontos. Isso se assemelha ao que acontece com a UML e seus diagramas, por exemplo o Diagrama de Comunicação e o Diagrama de Seqüência.

Entretanto, as diferentes visões que as modelagens possuem acabam por promover o desafio de conferir qual abordagem é mais apropriada a um determinado tipo de software ou aplicação para *Web*.

Nesse contexto, percebeu-se que o desenvolvimento para qualquer aplicação *Web* de qualidade depende da escolha de uma abordagem de modelagem adequada, pois a facilidade de transcrição dos modelos para a construção da aplicação depende de como foi realizada a abstração dos dados.

7.2 Contribuição

Como contribuição, pode-se citar o empenho de mostrar um nova área de estudos e de pesquisa para futuros trabalhos, pois este trabalho teve o objetivo de deixar uma pequena base para novos estudos.

Possibilitar ou tentar colocar as idéias de convergências das abordagens de modelagem para Engenharia de Software para *Web* como aconteceu com a UML, pois, ao apresentar uma nova abordagem de modelagem híbrida e baseada nas outras abordagens de

modelagem estudadas, teve-se o objetivo de extrair o que há de relevante de cada uma, promovendo uma abordagem mais abrangente.

7.3 Trabalhos Futuros

A sugestão de trabalhos futuros é a continuação dos estudos sobre a área de Engenharia de Software para *Web*, focalizando o desenvolvimento de abordagens de modelagem para Software Livre (SL) em *Web*. Outra sugestão é a ampliação e o melhoramento de MAW no sentido de compará-lo com outras modelagens não focadas nesse estudo e, também, a sua efetiva aplicação no desenvolvimento de alguma *WebApp* de grande porte.

Uma questão que seria de grande valia é iniciar uma discussão de convergência para modelagens de desenvolvimento *WebApps*, no mesmo sentido que houve para com a UML na Engenharia Software Tradicional. A convergência em si permitiria melhor esforço para conferir a um modelo padrão a capacidade de comporta o dinamismo da *Web*.

REFERÊNCIAS BIBLIOGRÁFIAS

[01] BRITO, L. S. F.,"WEBSCHARTS: Uma Ferramenta de desenvolvimento de aplicações Web baseada no HMBS/M", Dissertação de Mestrado, UFMS, 2003.

[02] CONTE T., M., TRAVASSOS G. H., "Processos de Desenvolvimento para Aplicações Web: Uma Revisão Sistemática" Artigo, 2005.

[03] PRESSMAN, R. S., “Engenharia de Software”, Livro, 5° ed., Editora MC Graw Hill, 2002.

[04] COSTA, C. G. A. da, “Desenvolvimento e Avaliação Tecnológica de um Sistema de Prontuário Eletrônico do Paciente, Baseado nos Paradigmas da World Wide Web e da Engenharia de Software” , Dissertação de Mestrado, Unicamp, 2001.

[05] SITE, http://phpmetar.incubadora.fapesp.br texto-site consultado em 14/08/2007.

[06] SITE, http://www.mundooo.com texto-site consultados em 14/08/2007.

[07] Site, http://phpmetar.incubadora.fapesp.br consultados em 14/08/2007.

[08] LOCATELLI M. H., "Engenharia Software para o desenvolvimento de WebApps e as Metodologias OOHDM e WEBML", Monografia, UFSC, 2003.

[09] ALMEIDA, A. de, DAROLT R., "Pesquisa e Desenvolvimento em UML", Projeto, UNISUL,2001.

[10] BOOCH, G., RUMBAUCH, J., e JACOBSON, I., “UML Guia do Usuário”, Livro, 2° ed., Editora Campus, 2006.

[11] ESMIN, A. A. A.; "Modelando com UML – Unified Modeling Language", artigo, Ulbra, Ji-Paraná, 2003.

[12] BOOCH, G., RUMBAUCH, J., e JACOBSON, I., “The Unified Modeling Language Reference Manual”, Livro, 2° ed., Editora Campus, 1998.

[13] ROQUE, B., “UML – que Raios e Isso”, Texto-Site, PlugMasters, http://www.plugmasters.com.br/sys/materias/476/1/UML---Que-raios-%E9-isso%3F consultado em 29/11/2006.

[14] WIEDENHOFT, G. R., “Modelagem Sistemas em Tempo Real em UML” Texto-Site, http://www.lisha.ufsc.br/~grw/eso/trabalho.html consultado em 29/11/2006.

[15] JUNIOR, J. F. S., “Extensões da UML pra Descrever Processos de Negócio”,

Dissertação de Mestrado, UFSC, 2003.

[16] SITE, http://wpslive.pearsoncmg.com/br_medeiros_uml_1/0,9044,1285491-, 00.html, texto-site, consultado em 10/01/2007.

[17] SITE, http://www.visual-paradigm.com/VPGallery/diagrams/index.html, texto-site, consultado em 10/01/2007.

[18] ROSSI, G., "Um método orientado a objetos para o projeto de aplicações hipermídia", Tese Doutorado, PUC-Rio, 1996.

[19] SANDRÍ, M. L., "Análise Comparativa entre o Modelo OOHDM e Ontologias a partir de uma Aplicação Baseada em Material Didático", Monografia, CULP, 2005.

[20]ÁLVARES, P. M. R. S., "*WebPraxis*: Um Processo Personalizado para Projetos de Desenvolvimento para *Web* ", Dissertação de Mestrado, UFMG, 2001.

[21] SITE http://www-lifia.info.unlp.edu.ar/~fer/oohdm/, texto-site, consultado em 10/01/2007.

[22] CUNHA, M. C. R. M. Da, "Autoria em Hipermídia: O modelo OOHDM Aplicado ao Ensino de Linguagens de Programação", Monografia, UFLA, 2002.

[23] DZENDZIK, I. T., "Processo de Desenvolvimento de Web Sites com Recursos UML", Dissertação de Mestrado , IMPE, 2004.

[24] SITE http://www.webml.org/, texto -site, consultado em 14/08/2007.

[25] LINHALIS F., "Uma Visão Geral de WebML e sua Utilização em uma Ferramenta CASE", Artigo, 2004.

[26] BLANCO, J., "Sistema Web para Controle de Processos Acadêmicos", Monografia, Unesp-Guaratinguetá, 2006.

[27] KRAUS, A., KOCH, N., "The Expressive Power of UML-based Web Engineering", Artigo, 2002.

[28] KRAUS, A., KOCH, N., "Generation of Web Applications from UML Models Using an XML Publishing FrameWork", Artigo, 2002.

[29] MACHADO, D de S, "Manutenção e Documentação do Portal Corporativo da 6ª Região da PMMG", Monografia, Ufla, Lavras, 2004.

[30] KOCH, N., KRAUS, A., HENNICKER, R., "The Authoring Process of the UML-based Web Engineering Approach", Artigo, 2002.

Ficha Catalográfica preparada pela Divisão de Processos Técnico da Biblioteca Central da UFLA

Moreira, Rodrigo Pereira

MAW – Uma Abordagem Híbrida de Modelagem para Aplicações WebApps / Rodrigo Pereira Moreira. Lavras – Minas Gerais, 2007. XX : il.

Monografia de Graduação – Universidade Federal de Lavras. Departamento de Ciência da Computação.

1. Introdução. 2. Engenharia de Software. 3. UML. 4. Abordagens de Modelagem para o Desenvolvimento de Software para Web. I. MOREIRA, R. P. II. Universidade Federal de Lavras. III. Título.