2
Conceitos

Neste capítulo serão apresentados os conceitos que serviram de base para a elaboração desta dissertação. Também serão referenciados os trabalhos cujas idéias e fundamentos inspiraram e nortearam o desenvolvimento do trabalho apresentado.

2.1.
Design Rationale

Design rationale consiste nas decisões tomadas durante o processo de *design* de um artefato e nas razões que motivaram tais decisões. O *design rationale*, conforme definido em MacLean (1989), inclui não somente a descrição de um artefato em potencial, mas também opções alternativas e as razões que levaram à escolha de soluções específicas para produzir o artefato.

O registro do raciocínio utilizado durante o *design* possui o objetivo de guiar a tomada de decisões, manter a rastreabilidade dos *designs* e acumular conhecimento sobre as aplicações. Ainda segundo MacLean (1989), um objetivo importante do *design rationale* é garantir que futuramente as questões essenciais ao entendimento do *design* serão óbvias para outras pessoas (ou mesmo para o próprio autor do *design*).

Explicitar o raciocínio por trás de um *design* auxilia no entendimento das razões pelas quais as escolhas feitas foram consideradas apropriadas. O uso do *design rationale* pode representar uma solução para auxiliar os projetistas a identificar questões que de outra maneira poderiam passar despercebidas (Horner, 2006). Além disso, sistemas desenvolvidos para trabalhar com *design rationales* podem permitir aos projetistas procurar por projetos ou questões similares, de maneira a combinar soluções e a identificar idéias não consideradas.

O registro e uso do *design rationale* traz uma série de benefícios, sendo os mais significativos descritos a seguir:

- facilitar a comparação entre diferentes *designs* e permitir a combinação de soluções não consideradas pelo projetista (reuso de soluções), devido à maneira estruturada de captura do *design rationale* (MacLean, 1989; Gruber & Russell, 1996; Lee, 1997; Agg, 2005);
- auxiliar na resolução de problemas, ao prover informações sobre o raciocínio por trás do *design* (MacLean, 1989) ;
- auxiliar na comunicação entre os projetistas, ao tornar explícito o *design* desenvolvido e ao comunicar decisões críticas realizadas no passado, quais alternativas foram investigadas e as razões para a alternativa escolhida (MacLean, 1989; MacLean et al., 1996; Fischer et al., 1996; Gruber & Russell, 1996; Dix et al., 1998);
- auxiliar o projetista a recuperar decisões previamente tomadas, suas justificativas e argumentos. “Talvez, devido à fragilidade da memória humana, os projetistas necessitem de seus próprios registros de *design rationale* tanto quanto os responsáveis pela manutenção que trabalharão no sistema em um momento posterior.” (Conklin & Yakemovic, 1996);
- facilitar a manutenção do artefato, ao permitir que as pessoas responsáveis pela manutenção e evolução do artefato entendam de maneira mais clara e completa o *design* (MacLean et al., 1996; Gruber & Russell, 1996; Conklin & Yakemovic, 1996; Lee, 1997; Burge & Brown, 2000; Shum, 2006);
- evitar re-trabalho, ao disponibilizar as decisões aceitas e rejeitadas, permitindo o entendimento dos motivos que levaram a elas (Burge & Brown, 2000);
- registrar o raciocínio utilizado dentro de uma organização, ou “memória da empresa”, que poderia ser perdida caso os projetistas fossem desligados da empresa (Conklin & Yakemovic, 1996; Burge & Brown, 2000);
- auxiliar a prever as conseqüências de alterações propostas ao *design* (MacLean, 1989; MacLean et al., 1996; Fischer et al., 1996; Gruber & Russell, 1996);
- facilitar o aprendizado do *design* (Lee, 1997; Burge & Brown, 2000);

- facilitar a transferência de conhecimento sobre *design* entre projetos com *rationales* similares (Dix et al., 1998);
- encorajar a deliberação e considerações explícitas de alternativas, revelando soluções que de outra forma poderiam não ser percebidas (Dix et al., 1998);
- auxiliar na detecção de erros de *design*, ao prover uma descrição explícita do raciocínio utilizado durante o processo de *design* (Conklin & Yakemovic, 1996).

Especificamente sobre o auxílio à comunicação, deve-se ressaltar que freqüentemente os projetistas possuem sua própria maneira de analisar as necessidades e capacidades dos usuários. Sendo assim, a existência de um *design rationale* explícito pode ajudar a identificar áreas onde premissas inadequadas foram feitas, ou onde premissas chaves não foram explicitadas, e mitigar a tendência dos projetistas de não perceber alternativas possíveis ao tomar decisões importantes (MacLean, 1989).

Sobre a estruturação do *design*, especialmente das razões do *design*, deve-se evidenciar o benefício de prover mecanismos pelos quais as pessoas com diferentes metas possam comunicar suas posições sobre as questões de *design*. Também permite que pessoas não diretamente envolvidas com o processo de *design* utilizem-se da documentação produzida para auxiliar na análise das decisões que foram previamente consideradas (Horner, 2006).

Outro benefício da explicitação do *design rationale* é prover um vocabulário comum e registrar a “memória” dos projetos, além de facilitar o entendimento e a negociação, permitindo que os participantes cheguem a um consenso sobre as questões de *design*. Além disso, auxilia os projetistas a rastrearem as questões não resolvidas e suas dependências, a darem prosseguimento a processos de *design* interrompidos e a oferecerem suporte à alocação distribuída de tarefas (Lee, 1997).

Em suma, o objetivo maior do registro e uso do *design rationale* é produzir artefatos mais completos, em menor tempo, com menos esforço, mais fáceis de serem mantidos e evoluídos, e com melhor qualidade de *design*.

2.2. Ontologias

Ontologias são especificações formais e explícitas de uma conceitualização compartilhada (Gruber, 1993). Ainda segundo Gruber, uma conceitualização é uma visão simplificada e abstrata do mundo que se quer representar. A conceitualização representa um modelo abstrato de algum aspecto do mundo que se deseja representar, de modo a identificar os conceitos mais significativos desse aspecto e os relacionamentos entre eles. Explícita significa que os conceitos estão claramente definidos e formal significa que a ontologia deve ser passível de processamento automático (Fensel, 2001). Compartilhada significa que a ontologia descreve um domínio conhecido por um grupo de pessoas, e não por apenas um indivíduo.

Em outras palavras, uma ontologia pode ser definida como uma descrição formal e explícita de conceitos em um domínio. Os conceitos são representados como classes, também sendo representadas as propriedades e relacionamentos entre as classes e suas restrições (Noy & McGuinness, 2001). Desse modo, uma ontologia descreve um domínio através de proposições lógicas que definem quais são os conceitos do domínio, como eles estão relacionados e como o relacionamento entre eles está restringido.

As ontologias podem ser utilizadas para descrever semanticamente os dados, permitindo a estruturação e amplo entendimento do significado do conteúdo descrito. Uma ontologia descreve o conhecimento através de um vocabulário definido, incluindo regras formais que são computacionalmente processáveis. Devido a isso, o conhecimento pode ser compartilhado, estendido e reusado por diferentes grupos de pessoas, de diferentes domínios.

2.3. Kuaba

Kuaba é uma abordagem para a representação de *design rationale* que faz uso de uma ontologia de mesmo nome desenvolvida por Medeiros (2006). Esta ontologia descreve um modelo de representação de conhecimento para *design rationale*. A ontologia Kuaba permite que seja adicionada semântica ao conteúdo do raciocínio capturado, possibilitando a realização de inferências e outras operações computáveis no *design rationale*.

Para que os benefícios mencionados anteriormente de se registrar e utilizar o *design rationale* sejam alcançados é necessária a existência de uma linguagem de representação adequada. Segundo Lee & Lai (1996), se o *design rationale* fosse representado através de uma linguagem informal, os benefícios obtidos não seriam significativamente maiores do que os de um processo tradicional de *design*, com anotações em texto livre.

O poder expressivo da linguagem de representação está relacionado à sua capacidade de ser processada computacionalmente, que por sua vez está relacionada ao formalismo da linguagem escolhida. Como a abordagem Kuaba utiliza uma ontologia descrita em uma linguagem formal para estruturar os elementos do *design rationale*, seu poder expressivo é elevado.

A ontologia Kuaba define um vocabulário para registrar o *design rationale* durante o processo de *design*, juntamente com as decisões e as justificativas para estas decisões, além dos relacionamentos entre os argumentos e os artefatos gerados (Medeiros et al., 2005b).

Segundo Potts (1996), o *design rationale* pode se tornar mais poderoso através da incorporação do conhecimento do domínio, ou seja, do conhecimento sobre como realizar o *design* de software. Esse conhecimento pode ser representado pelos metamodelos existentes para *design* de software, como por exemplo, a linguagem de especificação UML[3] para a descrição de diagramas de classes. A abordagem Kuaba utiliza a semântica formal dos artefatos definida pelos métodos de *design* ou linguagens de modelagem utilizados, o que permite a um ambiente de *design* integrado oferecer sugestões de opções de *design* a cada passo do *design*. Na abordagem Kuaba, o metamodelo do método de *design* é utilizado na criação de instâncias dos elementos de raciocínio (questão e idéia), o que permite que seja automatizada a geração de partes dos valores que seriam informados pelos usuários durante o processo de *design* (Medeiros et al., 2005a).

A abordagem Kuaba possui um vocabulário definido que oferece um conjunto de elementos que permitem a captura de *design rationale* de forma precisa e estruturada. Os elementos deste vocabulário são apresentados de uma forma simplificada na figura a seguir, usando a notação gráfica da UML para facilitar a visualização:

[3] http://www.uml.org/

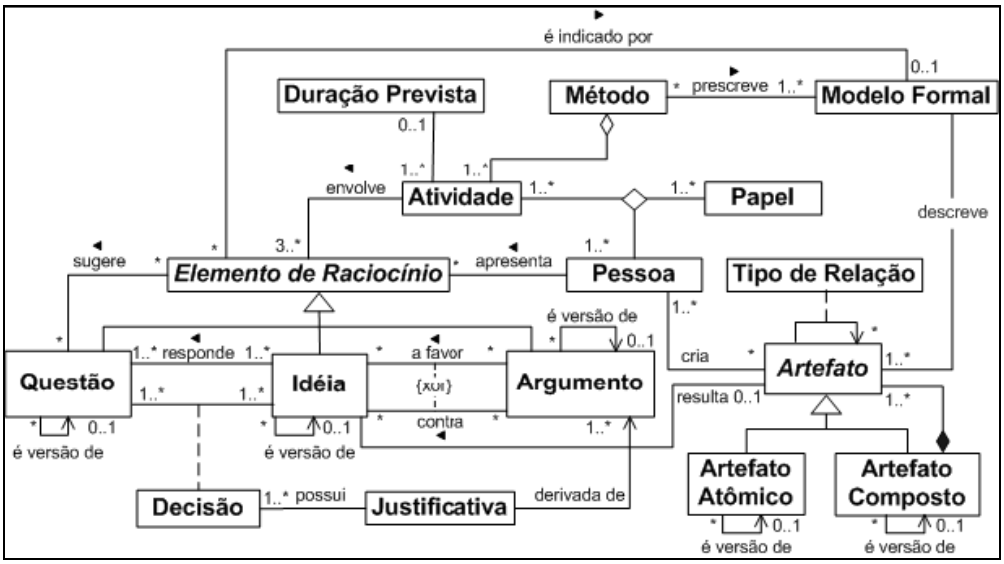

Figura 1 – Elementos do vocabulário Kuaba, copiado de Medeiros (2006)

O vocabulário da ontologia Kuaba permite a representação de todo o raciocínio utilizado no processo de *design* de um artefato, e é constituído basicamente pelos seguintes elementos:

- Elementos de raciocínio: são os itens principais, divididos em 3 categorias:
  - Questões: representam os questionamentos que surgem na elaboração do artefato, e que podem possuir várias alternativas de solução.
  - Idéias: são as soluções propostas para as questões, e são embasadas em argumentos.
  - Argumentos: representam as razões contra ou a favor de uma idéia que responde a uma questão.
- Decisão: entre uma questão e uma idéia há uma decisão associada, que indica se a idéia foi aceita ou rejeitada como solução de uma questão.
- Justificativa: para cada decisão deverá haver uma justificativa, sempre baseada nos argumentos associados à idéia cuja decisão está sendo justificada.
- Artefato: é a implementação de uma idéia, e pode ser um artefato atômico ou composto por outros artefatos.
- Método: representa o método de *design* utilizado na modelagem da aplicação, como por exemplo, o método OOHDM (Schwabe & Rossi, 1998).

- Modelo formal: é definido pelo método de *design* adotado, e dirige o processo de *design rationale* da aplicação, como por exemplo, o modelo navegacional do método OOHDM.

O vocabulário da ontologia Kuaba compreende, além dos itens citados, outros que não serão melhor detalhados por fugirem ao escopo do trabalho. Para maiores informações, ver Medeiros (2006).

A partir do conhecimento do modelo formal utilizado na modelagem da aplicação, pode-se instanciar o Kuaba de modo a produzir um grafo de *design rationale*, contendo todas as idéias consideradas pelo projetista. O conhecimento do modelo formal adotado permite que as questões sejam geradas automaticamente.

Um exemplo de um grafo Kuaba gerado para parte do projeto de navegação de uma aplicação de uma loja de CDs, baseado no método OOHDM, é ilustrado a seguir:

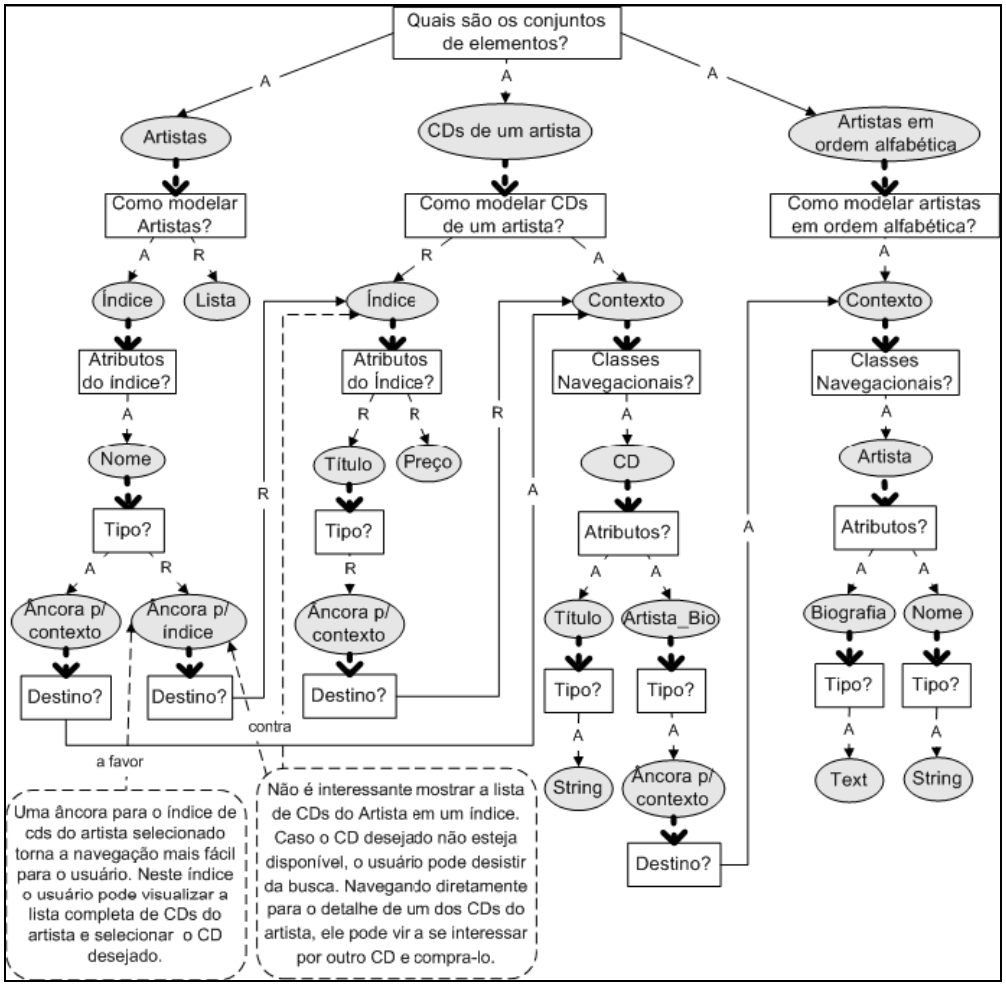

Figura 2 – Grafo Kuaba, copiado de Medeiros (2006)

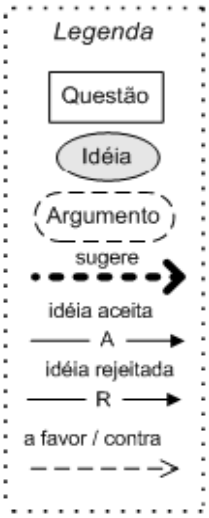

Figura 3 – Legenda da representação do vocabulário Kuaba, copiado de Medeiros (2006)

A ontologia definida pela abordagem Kuaba contribui para o processo de captura e uso de *design rationale*, ao permitir o reuso de software em um nível mais alto de abstração, através da integração de *rationales* para o desenvolvimento de novos artefatos. Esta integração, permitida através da realização de operações computáveis sobre o *design rationale* capturado, auxilia o projetista a obter soluções de *design* mais completas, além de facilitar a manutenção e evolução, que podem ser consideradas como uma continuação de um *design* anterior.

2.4. Aplicações Hipermídia

Os termos hipermídia e hipertexto foram concebidos por Ted Nelson (Nelson, 1965) para definir conceitos até então não formalizados. O termo hipertexto foi cunhado para designar uma forma de representação da informação que refletisse a maneira não-linear, associativa e não-hierarquizada do funcionamento do pensamento humano. O termo hipermídia consiste na união dos conceitos de hipertexto e multimídia, sendo multimídia considerada o conjunto de meios de informação que incluem, além dos elementos textuais, elementos não puramente textuais, como por exemplo, vídeos, sons, gráficos e imagens.

Hipertexto atualmente significa uma abordagem para apresentação de informação cujos documentos possuam vínculos (hiperlinks ou links) para outros documentos ou para outras partes do mesmo documento. O sistema de hipertexto mais conhecido é a WWW. Esses vínculos ou referências permitem ao usuário realizar uma navegação não-linear nas informações disponibilizadas.

Aplicações hipermídia, por sua vez, consistem em sistemas que permitem ao usuário percorrer informações inter-relacionadas de maneira não-linear, por meio de referências entre elementos como textos, animações, gráficos, ilustrações e outros.

Segundo Halasz (1991), hipermídia é um estilo de construção de sistemas para representação e gerenciamento de informação por meio de um conjunto de nós multimídia conectados entre si por links. A navegação pode ser realizada através desses links ou por meio de pesquisa de conteúdo relacionado (mecanismos de busca).

Devido à natureza peculiar das aplicações hipermídia, cujas propriedades mais expressivas são o aspecto navegacional e a heterogeneidade da informação manipulada, fez-se necessário o desenvolvimento de métodos de *design* apropriados para tais características.

"As metodologias tradicionais de engenharia de software ou as de desenvolvimento de sistemas de informação utilizando bases de dados não contêm abstrações úteis capazes de facilitar a tarefa de especificar aplicações que incorporem a metáfora do hipertexto." - tradução livre de Schwabe & Rossi (1998).

Mais especificamente, as metodologias tradicionais não possuem o conceito de elos entre os itens de informação, além de não proverem o suporte necessário para a integração de hipertexto na interface.

No intuito de oferecer suporte ao desenvolvimento de aplicações hipermídia foram criados métodos de *design* específicos, como por exemplo, OOHDM e SHDM, que serão melhor detalhados na próxima seção.

2.5. Métodos de Design para Aplicações Hipermídia

Devido à necessidade de um processo estruturado e sistemático para o desenvolvimento de aplicações hipermídia foi desenvolvido o HDM – *Hypermedia Design Method* (Garzotto et al., 1993), um método de *design* que permite a descrição concisa das aplicações independente da plataforma de

implementação. Após o HDM, vários outros métodos com ênfase no aspecto navegacional das aplicações hipermídia surgiram, como por exemplo, WebML (Ceri et al., 2000), OO-H Method (Gómez et al., 2000), OOHDM (Schwabe & Rossi, 1998) e SHDM (Lima, 2003).

O projeto de aplicações hipermídia requer uma atenção especial ao modo como o usuário irá navegar através dos nós de informação. É parte fundamental da modelagem da aplicação a maneira como as associações entre os itens de informação serão estruturadas, pois da qualidade destas ligações dependerá em grande parte o sucesso da aplicação hipermídia. O usuário deve sentir-se confiante ao navegar através dos nós de informação para realizar suas tarefas, explorando sem dificuldade o conteúdo desejado.

Nas próximas seções serão apresentados os métodos de *design* OOHDM e SHDM, cujos modelos servem de base para a criação de aplicações no ambiente HyperDE, que por sua vez foi estendido para a elaboração do ambiente HyperDE+DR, foco desta dissertação.

2.5.1. OOHDM

O método OOHDM *(Object Oriented Hypermedia Design Method)* foi concebido com o intuito de disciplinar o processo de desenvolvimento de aplicações hipermídia. Esse método utiliza técnicas de orientação a objetos e compreende 5 etapas, onde cada etapa aborda uma questão particular de *design*, gerando ou enriquecendo um modelo através de uma combinação de estilos iterativos e incrementais de desenvolvimento.

A abordagem OOHDM possui as seguintes abstrações centrais (Schwabe & Rossi, 1998):

- *"a noção de que objetos navegacionais são visões, do ponto de vista de banco de dados, de objetos conceituais;*
- *o uso de abstrações apropriadas para organizar o espaço de navegação, com a introdução de contextos de navegação;*
- *a separação de questões de interface das questões navegacionais;*
- *uma identificação explícita de que existem decisões de design que devem ser realizadas somente no momento da implementação."*

As 5 etapas do OOHDM compreendem: levantamento de requisitos, projeto conceitual, projeto navegacional, projeto de interface abstrata e

implementação. A fase de levantamento de requisitos consiste na elicitação dos tipos de informação que a aplicação deverá manipular, e das tarefas que deverão ser apoiadas. O levantamento de requisitos inclui as seguintes etapas:

- identificação dos atores e tarefas: nesta etapa o projetista interage com o domínio da aplicação para identificar os papéis que serão desempenhados pelos usuários e os objetivos que os usuários desejam alcançar ao utilizar a aplicação.
- especificação dos cenários: nesta etapa são especificados os cenários[4] que descrevem as tarefas a serem realizadas pelos usuários, identificando-se as informações que serão trocadas entres eles e a aplicação.
- especificação dos casos de uso: a partir dos cenários, deve-se especificar os casos de uso[5], incluindo as informações descritas nos cenários, identificando as seqüências de ações e as operações realizadas sobre os itens de dados.
- especificação dos UIDs: para cada caso de uso define-se um UID (Diagrama de Interação do Usuário), que consiste em uma representação gráfica das interações entre o usuário e a aplicação.
- validação dos casos de uso e dos UIDs: nesta etapa os casos de uso e UIDs são validados com os usuários, de modo a ser obtida uma aprovação sobre as tarefas e informações descritas.

O projeto conceitual consiste na análise do domínio da aplicação, descrevendo-o através de classes e relacionamentos entre essas classes. O projeto conceitual é independente de plataforma de hardware e software, e gera como produto um esquema de classes conceituais representadas como em modelos orientados a objetos da UML. A diferença em relação ao modelo da UML é a presença de atributos multi-tipados, ou seja, atributos que podem ter diferentes tipos que representam perspectivas. O esquema conceitual é gerado a partir dos UIDs descritos na fase de levantamento de requisitos.

O projeto navegacional consiste na criação de um esquema navegacional a partir do esquema conceitual, agregando a ele aspectos (atributos)

[4] Para maiores informações sobre cenários, consultar (Zorman, 1995) e (Carroll, 1995).

[5] O livro (Jacobson et al., 1992) descreve extensamente o tema. O artigo (Jacobson, 2003) provê um resumo do assunto, com informações como origem, evolução e melhorias futuras.

navegacionais. O esquema navegacional provê uma visão sobre o modelo conceitual, possibilitando a geração de diferentes modelos navegacionais para um mesmo modelo conceitual. Também são gerados nesta etapa o esquema de classes em contexto, o diagrama de contexto de navegação e os cartões de especificação. O esquema de classes em contexto descreve como as classes se comportam dependendo do contexto em que se encontram. O diagrama de contexto de navegação descreve como é a navegação entre os elementos da aplicação. Os cartões de especificação descrevem as características de cada contexto e estrutura de acesso definido. Visando um melhor entendimento, as principais primitivas do projeto navegacional serão descritas a seguir:

- nós: representam instâncias de classes navegacionais através do qual os usuários podem navegar. Um nó pode ser uma visão sobre uma ou mais classes do modelo conceitual, e engloba as informações necessárias e relacionadas para a execução de uma tarefa.
- elos: constituem os relacionamentos entre os nós, geralmente derivados dos relacionamentos do modelo conceitual. Definem a navegação possível entre os nós de informação.
- contextos de navegação: um contexto consiste em um conjunto de objetos que serão explorados pelo usuário durante a realização de uma tarefa, possuindo uma mesma estrutura de navegação.
- estruturas de acesso: representam as maneiras possíveis de se acessar um nó. Uma estrutura de acesso, consiste em um conjunto de itens ordenados que permitem acesso a outros objetos (contextos ou estruturas de acesso), funcionando como um índice.
- âncoras: constituem atributos de estruturas de acesso ou de classes navegacionais que permitem a navegação para outros objetos (índices ou nós).

Um conceito interessante introduzido é o de *landmark*. Um *landmark* funciona como uma âncora que pode ser acessada de qualquer parte da aplicação, em qualquer momento. Um *landmark* pode ser considerado um item de um menu fixo da aplicação, sempre visível para o usuário.

A etapa de projeto de interface abstrata permite criar um modelo abstrato de interface independente da tecnologia de interface utilizada. O objetivo desta etapa é definir como os objetos da aplicação serão visíveis para o usuário e quais deles permitirão a ativação da navegação e outras funcionalidades, além

de descrever as transformações possíveis na interface. O comportamento da interface é definido pela especificação de como os eventos serão tratados e como será a comunicação entre os objetos de interface e os objetos navegacionais.

A etapa de implementação consiste no mapeamento dos objetos conceituais, navegacionais e de interface no ambiente de execução escolhido. O produto desta etapa é a aplicação final, gerada através da transformação dos modelos em códigos executáveis.

2.5.2. SHDM

O método SHDM, proposto por Lima (2003), é uma extensão do OOHDM e compreende as mesmas 5 fases. A diferença está no fato de que o SHDM agrega semântica aos dados manipulados através de ontologias, e permite a construção de navegação facetada. Os esquemas conceitual e navegacional são estendidos através de ontologias, o que agrega um valor expressivo significativo, além de torná-los implementáveis na Web Semântica[6].

Novas primitivas de estruturas de acesso foram modeladas para permitir o uso de múltiplos critérios de classificação dos objetos navegáveis, além da mistura desses critérios. Essas novas primitivas foram denominadas estruturas de acesso facetadas, constituindo uma especificação concisa equivalente a grandes enumerações de caminhos de navegação possíveis (Lima & Schwabe, 2003).

2.6. Ambiente HyperDE

A ferramenta HyperDE *(Hypermedia Development Environment)*, conforme descrita em Nunes (2005), *"é a combinação de um framework MNVC e um ambiente de desenvolvimento rápido que permite a arquitetos de informação construir em um curto espaço de tempo um protótipo de uma aplicação modelada segundo os métodos OOHDM ou SHDM."* O framework MNVC mencionado é o padrão MVC (*model-view-controller*) largamente utilizado, acrescido da letra N para indicar que o modelo contém funcionalidades navegacionais.

[6] http://www.w3.org/2001/sw/

O usuário do ambiente HyperDE desenvolve suas aplicações dinamicamente, de forma interativa, a partir das informações inseridas na ferramenta. De acordo com a arquitetura da ferramenta, o usuário (no caso, o projetista ou desenvolvedor da aplicação) informa as classes navegacionais e os demais itens que constituirão sua aplicação hipermídia, com base na modelagem OOHDM ou SHDM previamente realizada.

Como o HyperDE combina uma arquitetura orientada a modelos com o uso de DSL *(Domain Specific Languages)*, conforme o usuário informa os itens do seu modelo navegacional, a aplicação vai sendo gerada automaticamente, podendo ser visualizada a cada interação com a ferramenta. Adicionalmente, como o modelo é especificado segundo o metamodelo para um método, também é possível manipular dinamicamente o modelo durante a execução, o que permite a geração de aplicações especificadas de forma muito concisa (Nunes & Schwabe, 2006), assim como adaptáveis (Assis, 2005).

Quando o usuário informa os itens de seu modelo navegacional, o metamodelo navegacional do HyperDE é instanciado. Este metamodelo contém as primitivas utilizadas nos métodos OOHDM e SHDM, como contextos, índices, âncoras, classes navegacionais, elos e landmarks. Assim sendo, ele descreve como são as relações entre as diferentes classes que poderão existir ao longo do desenvolvimento de uma aplicação hipermídia, conforme os métodos OOHDM e SHDM. Na versão atual, o HyperDE não oferece suporte ao conceito de navegação facetada.

2.6.1. Primitivas

A seguir serão descritas as principais primitivas do ambiente HyperDE, baseadas nos modelos OOHDM e SHDM. Os nomes estão em inglês para manter a compatibilidade com a ferramenta, que foi toda desenvolvida na língua inglesa. As traduções, quando necessárias, estão entre parênteses.

- NavClasses (Classes navegacionais): representam classes como no diagrama de classes da UML, acrescidas de uma visão navegacional. Tal visão é inserida na forma de atributos que permitem a definição de um destino para uma navegação.

- NavAttributes (Atributos de classe navegacional): representam os tipos possíveis de atributos que uma classe navegacional pode conter. São eles:
  - Simple Attribute: são atributos simples, sem navegação, que podem conter valores tais como texto, e-mail, entre outros.
  - Computed Attribute: são atributos computados, ou seja, derivados de um trecho de código especificado.
  - Index Attribute: são atributos que representam um índice.
  - Index Anchor Attribute: são atributos que representam uma âncora para um índice. Uma âncora é um link que, ao ser clicado, leva a um destino especificado.
  - Context Anchor Attribute: são atributos que representam uma âncora para um contexto.
- NavOperations (Operações de classe navegacional): representam operações que uma classe navegacional pode ter. Também conhecidas como métodos ou funções.
- Links (Elos): representam os relacionamentos entre as classes navegacionais, como em um diagrama de classes da UML.
- Contexts (Contextos): representam uma determinada visão sobre uma classe navegacional. Basicamente consiste em selecionar instâncias/atributos específicos de classes navegacionais segundo algum filtro. Ex.: todas as instâncias da classe Professor que atuem na área de Engenharia de Software.
- Indexes (Índices): representam estruturas de acesso para exibição de informações que, ao serem clicadas, levam a um destino onde informações relacionadas são exibidas. Ex.: um índice de professores onde, ao se clicar no nome de um professor, o destino é a página contendo as informações acadêmicas do mesmo.
- IndexAttributes (Atributos de índices): representam os tipos possíveis de atributos que um índice pode conter. São eles:
  - Simple Attribute: são atributos simples, sem navegação, que podem conter valores tais como texto, e-mail, entre outros.
  - Index Anchor Attribute: são atributos que representam uma âncora para um índice.
  - Context Anchor Attribute: são atributos que representam uma âncora para um contexto.

- Landmarks: representam os pontos iniciais da aplicação. Em termos visuais, representam o menu da aplicação gerada, por onde o usuário poderá iniciar sua navegação.
- Views (Visões): representam formas de exibir as informações na interface.
- Nodes (Nós): representam as instâncias das classes NavClass, NavAttribute, NavOperation e Link do metamodelo.

Essas primitivas fazem parte do metamodelo do HyperDE, cuja estrutura genérica é ilustrada a seguir:

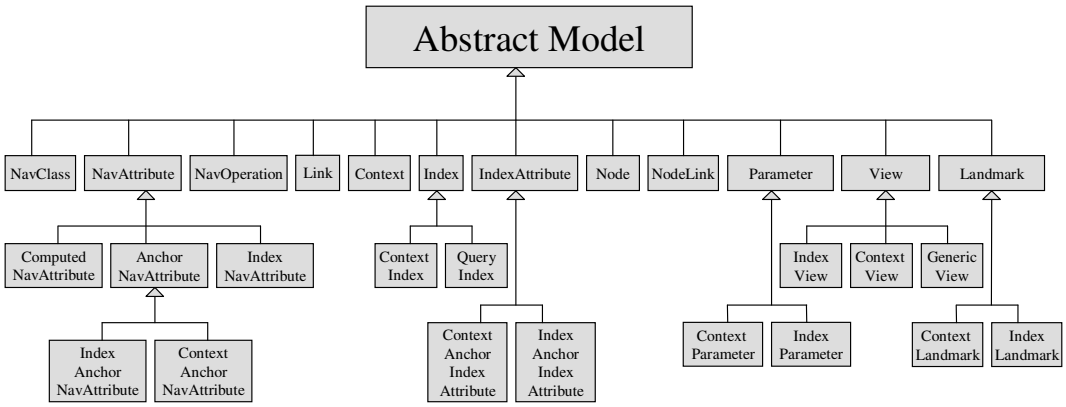

Figura 4 – Diagrama do Metamodelo Navegacional do Ambiente HyperDE

2.6.2. Framework MNVC

Como mencionado anteriormente, o ambiente HyperDE é baseado no framework MNVC, possuindo 3 camadas: modelo navegacional, controle e visão.

A camada de modelo navegacional permite a instanciação do metamodelo baseado nos métodos OOHDM e SHDM. Nessa camada pode-se definir e manipular (Nunes, 2005):

- *contextos navegacionais;*
- *estruturas de acesso (índices) e seus atributos;*
- *classes navegacionais, seus atributos e operações;*
- *elos;*

- *landmarks;*
- *nós, seus atributos e relacionamentos.*

Através da instanciação dessas primitivas o projetista/desenvolvedor define seu próprio modelo navegacional (o modelo que representa a aplicação sendo desenvolvida).

A camada de controle oferece controladores responsáveis pelas seguintes ações:

- controle da navegação;
- controle da manipulação do metamodelo do ambiente;
- controle da manipulação do modelo da aplicação.

O controle de navegação é responsável pela apresentação da visão adequada ao contexto, índice ou classe navegacional sendo exibida, além de controlar a construção das estruturas de acesso.

A camada de visão permite a definição de visões para contextos, classes ou índices. Visões são maneiras personalizadas de exibir o conteúdo na interface. Sendo assim, pode-se associar visões a determinados itens, de modo que quando estes sejam exibidos na interface, apareçam segundo a visão previamente definida.

Adicionalmente, *o framework fornece uma série de funções auxiliares (chamadas de "helpers") para ajudar no desenho de primitivas do sistema* (Nunes, 2005). Estas funções têm o objetivo de facilitar o desenvolvimento de itens de interface, *reduzindo a necessidade de conhecimento da linguagem de programação sob a qual o ambiente foi construído.*

A estrutura genérica do framework MNVC descrito é apresentada a seguir:

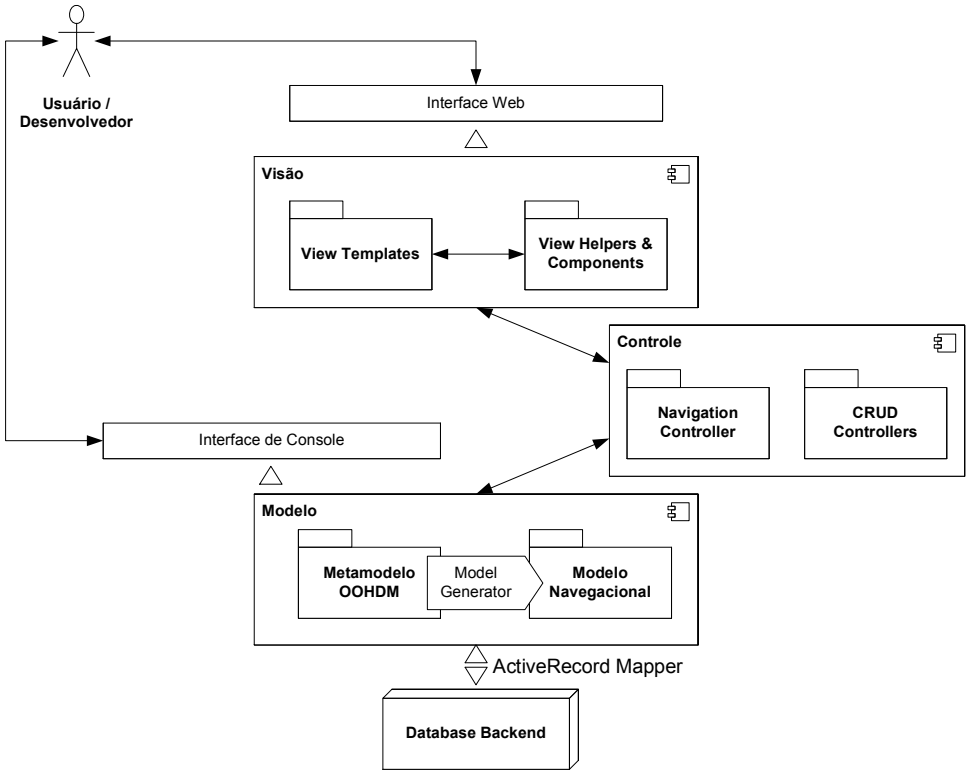

Figura 5 – Componentes da arquitetura do HyperDE, copiada de Nunes (2005)

2.6.3. Arquitetura de Implementação

O HyperDE foi construído como uma modificação do framework *Ruby on Rails*[7], sendo a camada de persistência *(ActiveRecord)* substituída por uma camada semântica *(SemanticRecord)*. As instâncias das classes, as classes criadas e o próprio metamodelo são armazenados em arquivos RDF[8], gerenciados pelo banco de dados semântico Sesame[9].

A seguir encontra-se uma descrição resumida dos principais aspectos de implementação.

[7] http://www.rubyonrails.com

[8] http://www.w3.org/RDF/

[9] http://www.openrdf.org/

2.6.3.1. Linguagem Ruby

A linguagem utilizada para definir a ferramenta foi *ruby*[10], que é uma linguagem de script interpretada, orientada a objetos. A escolha dessa linguagem foi grandemente determinada pela facilidade de aprendizado e pelos recursos que oferece. Sua sintaxe é flexível e dinâmica, permitindo a programação da aplicação enquanto ela é executada. Por ser inteiramente orientada a objetos, o mapeamento dos modelos baseados nos métodos OOHDM e SHDM para os objetos de programação é facilmente realizado. Além disso, a linguagem *ruby* possui o framework *Ruby on Rails*, descrito na próxima seção.

2.6.3.2. Framework Rails

O framework *Ruby on Rails*, ou simplesmente *Rails*, permite o desenvolvimento rápido de aplicações Web utilizando o modelo MVC *(model-view-controller).* É baseado na linguagem *ruby* e oferece uma grande robustez e agilidade no desenvolvimento de aplicações e foi escolhido devido à praticidade, facilidade de uso e produtividade (Nunes, 2005). A camada de modelo, conforme comentado anteriormente, foi reescrita para oferecer suporte ao armazenamento e manipulação de objetos em um banco de dados semântico, Sesame, descrito na próxima seção.

2.6.3.3. Banco de Dados Sesame

"Sesame é um banco de dados semântico de código-aberto, escrito em Java que utiliza RDF para armazenamento de informações e suporta nativamente consultas e inferências de declarações RDF." (Nunes, 2005) O suporte a RDF resultou em uma vantagem de sua adoção, pois o RDF acomoda perfeitamente os metamodelos do OOHDM e SHDM. Outras vantagens da escolha desse banco de dados foram: (i) interface de acesso através de protocolo http, o que o torna independente de plataforma das aplicações clientes; (ii) nível satisfatório de desempenho; (iii) interface visual Web que facilita o acesso aos repositórios de dados. A linguagem de consulta utilizada pelo

[10] http://www.ruby-lang.org/en/

Sesame é SeRQL (pronuncia-se como “circle”, em inglês). Para maiores detalhes sobre a linguagem SeRQL *(Sesame Rdf Query Language)*, ver Broekstra & Kampman (2003).