Uma infra-estrutura de Reutilização baseada em Modelos de Domínio

Regina Maria Maciel Braga
Claudia Maria Lima Werner
Marta Mattoso

Lista de Figuras

Lista de Tabelas

1. Introdução

1.1 Motivação

Atualmente, com a competitividade do mercado cada vez maior, a necessidade das empresas concentrarem-se cada vez mais na capacidade de seus funcionários é um requisito essencial [BER97]. Neste contexto, as empresas de desenvolvimento de software atravessam os mesmos problemas de competitividade das empresas em geral, aliando problemas organizacionais, como a necessidade de redução do número de funcionários, à problemas técnicos, como a necessidade de melhoria no processo de desenvolvimento de software.

Em relação aos problemas técnicos, o desenvolvimento baseado em componentes [CZA97] está cada vez mais ganhando adeptos e é uma das propostas promissoras para a melhoria da produção de software nos próximos anos. A construção de software através de componentes leva a uma maior reutilização e consequente alta na produtividade [BER97], o que enfatiza a crença que esta nova modalidade de desenvolvimento de software venha a crescer.

Hoje, o que temos na maioria dos produtos de software disponíveis no mercado é uma abordagem de desenvolvimento em blocos monolíticos, formados por um grande número de partes interrelacionadas, onde estes relacionamentos estão na maioria das vezes implícitos. O desenvolvimento baseado em componentes tem como objetivo a "quebra" destes blocos monolíticos em componentes interoperáveis, reduzindo desta forma a complexidade, assim como os custos, através da reutilização de componentes exaustivamente testados. Desta forma, observa-se que o desenvolvimento de software baseado em componentes envolve três atividades: i) a reutilização de componentes, ii) a concepção fragmentada e iii) a interoperação entre estes componentes.

Segundo Krueger [KRU92], para uma técnica de reutilização de software, como é o caso do desenvolvimento baseado em componentes, ser efetiva, ela deve reduzir a distância cognitiva entre o conceito inicial de um sistema e sua implementação final. Sendo assim, no contexto do processo de desenvolvimento de software baseado em componentes, a aplicação da reutilização nas fases iniciais do desenvolvimento deve ter como resultado a reutilização de componentes na implementação da aplicação, ou seja, deve haver uma relação estreita entre os

conceitos do domínio considerados reutilizáveis nas fases iniciais do processo de desenvolvimento e os componentes codificados que serão utilizados na implementação da aplicação. Sendo assim, a redução desta distância congnitiva passa por uma integração das três atividades anteriormente identificadas.

A associação entre estas atividades não é uma tarefa simples. As abordagens de desenvolvimento baseado em componentes encontradas na literatura tendem a privilegiar uma atividade em detrimento das demais. Segundo Klaus Berg [BER97], existem três abordagens principais na adoção do desenvolvimento de software baseado em componentes, a saber:

- Desenvolvimento baseado em *Frameworks*: Esta abordagem provê o desenvolvedor com *frameworks* de aplicações e arquiteturas de software específicos para um dado domínio. Os componentes neste contexto não são genéricos, ou seja, geralmente são construídos para domínios específicos, requerendo assim uma arquitetura que represente como estes componentes são agregados. Sem a arquitetura, a interoperabilidade dos componentes fica prejudicada e a produtividade esperada não é alcançada.
- Técnicas formais e baseadas no conhecimento: Esta abordagem, por ser complexa, ainda está no âmbito de pesquisas acadêmicas. O objetivo principal é a descrição de componentes e seus relacionamentos através do uso de métodos formais e/ou informais, levando em consideração o conhecimento do domínio para automatizar o processo de construção de aplicações. Esta abordagem é baseada no uso de componentes desenvolvidos de maneira independente (componentes já preexistentes, que podem ser resultado da reengenharia de sistema legados), que negociam a sua interoperabilidade através do uso do conhecimento do domínio e semântica do componente. Esta abordagem é muito utilizada em programação orientada a agentes [RUS95].
- *Componentware*, que representa uma série de atividades relacionadas com o objetivo de interrelacionar componentes através de modelos de objetos ou similares. Neste contexto, modelos de objetos como o COM da Microsoft e o CORBA da OMG [OMG95] são utilizados, descrevendo como os objetos se comunicam uns com os outros, não se preocupando com a linguagem de programação, espaço de endereçamento, máquina ou sistema operacional utilizados. Modelos de documentos como o OLE 2 da Microsoft e OpenDoc do CILabs também são

usados nesta abordagem. Atualmente, o *componentware* vêm crescendo a passos largos e vem destacando-se com maior ênfase no mercado. Como exemplo de utilização desta abordagem temos o desenvolvimento de componentes utilizando tecnologias como ActiveX, JavaBeans, como é o caso do projeto *San Francisco*, da IBM [BOH97];

Analisando as três abordagens acima em relação ao suporte dado ao desenvolvimento de software baseado em componentes, podemos fazer as seguintes considerações:

- a grande vantagem da abordagem baseada em *frameworks* é que a arquitetura de software do domínio guia o desenvolvedor na escolha e utilização dos componentes, ao passo que no *componentware* o desenvolvedor é quem deve decidir que componentes utilizar e como agregar estes componentes. A abordagem de desenvolvimento baseado em *frameworks* contempla, de maneira adequada, as fases de projeto e implementação dos componentes, provendo construções, como a arquitetura de software do domínio, que facilitam a escolha e utilização dos componentes por parte do desenvolvedor. No entanto, esta abordagem não dá ênfase a necessidade de entendimento inicial dos conceitos do domínio e de como estes conceitos estão relacionados, ou seja, falta, nesta abordagem, um suporte mais adequado às fases *upstream*. Uma forma de solucionar este problema pode ser, como ressaltado no item anterior, a sua integração com técnicas baseadas em conhecimento;
- o emprego de técnicas formais e baseadas no conhecimento tem como principal vantagem permitir um conhecimento detalhado dos conceitos do domínio e seus relacionamentos, em altos níveis de abstração. No entanto, apesar desta abordagem apresentar-se promissora, na prática, o que vemos na maioria dos Sistemas Baseados em Conhecimento, é que o suporte à construção de componentes, se dá, prioritariamente, nas fases iniciais do desenvolvimento, ou seja, nas chamadas fases *upstream*. Alguns exemplos de projetos baseados nesta abordagem são o SHADE [GRU92] e os trabalhos desenvolvidos pela equipe do pesquisador Fischer, da Universidade do Colorado [FIS96]. Para que esta abordagem produza resultados concretos, uma das possibilidades é a sua integração com técnicas que privilegiem o projeto e implementação de componentes, como é o caso do desenvolvimento baseado em *frameworks*;

- a utilização da abordagem de *componentware* esbarra em alguns problemas como a falta de orientação em relação a escolha dos componentes mais adequados a um dado contexto e a melhor maneira destes componentes se relacionarem. Muitas vezes, a forma de composição escolhida pelo desenvolvedor não é a mais adequada, e nem os componentes escolhidos são os que atendem melhor às necessidades da aplicação, levando-se em consideração as características do ambiente e do domínio onde serão utilizados. Um forma de atenuar estes problemas é a utilização de um modelo que "guie" o desenvolvedor na escolha de componentes a serem utilizados em um dado contexto. Por outro lado, a abordagem de *componentware* tem uma característica que a torna primordial no processo de desenvolvimento atual, que é a facilidade de permitir que objetos heterogêneos sejam agregados;

Analisando as vantagens e deficiências das três abordagens descritas por Berg [BER97], vemos que estas não são exclusivas e sim complementares, sugerindo assim uma abordagem híbrida. Como principais contribuições destas três abordagens descritas em [BER97] para a proposta de uma abordagem híbrida, temos a integração das três atividades identificadas anteriormente, ou seja, reutilização, modelagem e interoperação. Especificamente, essa integração proporciona: facilidades para a distribuição, comunicação e interoperação dos componentes, que é possível através do estudo de técnicas baseadas em *componentware*, como CORBA [OMG95]; um melhor entendimento e aquisição de conceitos do domínio, em altos níveis de abstração, que pode ser alcançado através do estudo de técnicas baseadas em conhecimento, como agentes inteligentes; e a concretização deste conhecimento abstrato em componentes reutilizáveis do domínio, através da adoção das técnicas de desenvolvimento baseadas em *frameworks*.

1.2 Objetivos

Considerando-se as dificuldades do uso efetivo de uma técnica de desenvolvimento de software baseado em componentes, este trabalho tem por objetivo apresentar uma abordagem híbrida, contemplando de forma integrada as diversas atividades envolvidas no desenvolvimento baseado em componentes. No contexto deste trabalho, denominaremos esta abordagem híbrida de **modelo para uma infra-estrutura de reutilização**.

Com a especificação de uma abordagem híbrida em relação às descritas por Berg, estamos buscando reduzir esta distância cognitiva entre os conceitos do domínio, provendo modelos que agreguem estes conceitos, em diferentes níveis de abstração, permitindo ao usuário um maior entendimento e utilização dos conceitos do domínio para facilitar a reutilização de componentes na implementação da sua aplicação.

Para atingirmos nossos objetivos, ou seja, para que a reutilização de componentes no desenvolvimento de novas aplicações em um dado domínio seja eficaz, é necessário definir: um modelo para uma infra-estrutura de reutilização, que privilegie o desenvolvimento baseado em componentes em todos os seus aspectos; e um conjunto de serviços básicos que facilitem a implementação deste modelo.

Segundo a definição dada por Tsichritzis [TSI82], um modelo de um sistema de computação genérico deve ser composto de três partes principais: parte estrutural, onde são definidas as entidades componentes do sistema; parte operacional, onde são definidas as operações que irão atuar sobre estas entidades; e regras de integridade, que definem a sistematização das operações constantes do modelo. Fazendo um mapeamento da definição de modelo de Tsichritzis para um modelo de infra-estrutura de reutilização, poderíamos caracterizar os três componentes como:

- a *parte estrutural* da infra-estrutura corresponde aos diversos **modelos de domínio,** já que representam os conceitos do domínio em diversos níveis de abstração. Estes modelos são representados pelos seguintes tipos: **modelos conceituais** que representam os diversos conceitos do domínio em um nível de abstração que facilite o entendimento por parte do usuário; modelos de projeto, ou **arquiteturas de software do domínio**[1], que representam como os conceitos do domínio são compostos, para permitir a construção de aplicações do domínio; e também como requisitos operacionais tais como distribuição, uso de um banco de dados, requisitos de desempenho, entre outros, devem ser tratados no contexto do domínio; e **modelos implementacionais**, tais como *frameworks*, que representam os componentes reutilizáveis do domínio e seus relacionamentos, e como estes componentes reutilizáveis são agregados às aplicações do domínio a serem desenvolvidas. A escolha das representações a serem utilizadas para estes modelos de domínio é igualmente importante. Estas representações têm que utilizar

construções com as quais os desenvolvedores que irão utilizar estes modelos estejam familiarizados.

- a *parte operacional* da infra-estrutura é especificada através da utilização de um processo adequado para a criação e evolução dos modelos de domínio. A este processo damos o nome de **Engenharia de Domínio** (ED). Podemos conceituar a ED como sendo o processo utilizado para a criação de uma infra-estrutura de auxílio à especificação, projeto e implementação de aplicações em um dado domínio [SEI96]. A ED é uma atividade essencial para que os modelos sejam especificados de forma correta, abrangendo todo o universo do domínio o qual se quer modelar.
- as *regras de integridade*, correspondem à adoção de um **método** que sistematize o processo de ED, permitindo a criação e utilização de modelos de domínio de acordo com o paradigma de desenvolvimento a ser seguido (OO, estruturado, baseado em eventos, etc.). Portanto, tão importante quanto a adoção da ED, é a utilização de um método que permita a geração de modelos conceituais, arquiteturas de software de domínio e modelos de implementação (*frameworks* de aplicação) integrados, consistentes, facilitando a transição entre os diferentes níveis de abstração.

Entretanto, para a especificação da infra-estrutura de reutilização, necessitamos, além de um modelo que contemple a estrutura, operações e regras de integridade, de componente operacional que permita a representação, o armazenamento e o gerenciamento das diversas informações do modelo. Dentre os serviços deste componente operacional, destacam-se:

- um modelo de armazenamento genérico e eficiente para o armazenamento da estrutura, operações e regras de integridade;
- controles operacionais tais como: mecanismos de segurança e acesso multiusuário;
- estruturas de acesso à infra-estrutura, que privilegiem questões como o desempenho e facilidade de uso;
- mecanismos que facilitem a evolução da infra-estrutura como um todo.

Analisando estes serviços básicos requeridos, podemos considerar a utilização de um SGBD (Sistema de Gerenciamento de Bases de Dados) como provedor destes serviços. Um SGBD possui, entre outros serviços, um modelo de armazenamento

[1] Termo equivalente ao utilizado na literatura como DSSA [TRA94] (arquitetura de software

eficiente, mecanismos para a especificação de consultas que privilegiam a eficiência, controle operacionais otimizados. Além disso, a disponibilidade de um modelo de representação poderoso, que permita a especificação dos modelos de domínio de forma adequada, é característica básica dos chamados SGBDs não convencionais. Dentre os modelos de representação utilizados pelos SGBDs não convencionais, o modelo OO é o que se apresenta como mais adequado para os nossos propósitos, por ser: i) um modelo expressivo, capturando de forma adequada os conceitos do domínio; ii) semi-formal, permitindo assim a checagem de certas consistências entre os conceitos representados; além de iii) facilitador da evolução dos conceitos. Sendo assim, para a disponibilização dos serviços requeridos, devemos aliar o modelo OO aos serviços básicos providos pelos SGBDs, ou seja, devemos utilizar os serviços de um SGBDOO (Sistema de Gerenciamento de Banco de Dados Orientado a Objetos), que alia a capacidade de representação do modelo OO aos serviços de um SGBD. Este componente é fundamental para a implementação dos serviços de *componentware.*

Desta forma, apresentamos neste documento os principais aspectos envolvidos na especificação do que denominamos de uma **Infra-estrutura de Reutilização Baseada em Modelos de Domínio (IRBMD)**. Conceituamos esta infra-estrutura como sendo um arcabouço onde são especificados modelos conceituais, arquiteturas de software do domínio e modelos implementacionais específicos para domínios de aplicação previamente selecionados, utilizando: A) representações dos modelos adequadas para o contexto onde será utilizada a infra-estrutura; B) um processo de ED consistente; C) um método de ED; e D) um conjunto de serviços de gerência de objetos, que permita a representação, armazenamento e gerenciamento dos demais componentes da IRBMD.

1.3 Resultados Esperados

Acreditamos que as principais contribuições do desenvolvimento deste trabalho serão:

- integração, em uma abordagem híbrida, das três abordagens principais do desenvolvimento de componentes, ou seja, desenvolvimento baseado em *frameworks*, desenvolvimento baseado em conhecimento, *componentware*,

específica do domínio)

principalmente em relação as duas primeiras abordagens. Dentro deste item, podemos destacar:

  - realização de um estudo detalhado de como unir conceitos do desenvolvimento baseado em *frameworks* com desenvolvimento baseado em conhecimento;
- desenvolvimento da IRBMD, direcionada para o apoio ao desenvolvimento de software. Dentro deste item, podemos destacar:
  - redução da distância cognitiva entre os conceitos iniciais do domínio de uma aplicação a ser desenvolvida e a implementação efetiva desta aplicação em um sistema computacional;
- especificação de um método de engenharia de domínio (ED), que contemple todas as fases da ED, de forma sistemática, onde os modelos (produtos) especificados em cada uma das fases sejam consistentes com os demais, facilitando assim a transição entre estes modelos. Dentro deste item, podemos ainda destacar:
  - utilização do conceito de padrões, de forma sistemática em uma abordagem de reutilização orientada a modelos, e sua inserção em um método de ED;
- realização de um estudo detalhado em relação a quais requisitos de armazenamento e gerenciamento um SGBD precisa ter para atender às necessidades da IRBMD. Neste item, podemos destacar:
  - uso de um sistema orientado a objetos e extensão de seus recursos de modo a atender aos requisitos da IRBMD.
- Avaliação da IRBMD, através da realização de estudo de casos que comprovem a real aplicabilidade da IRBMD.

1.4 Trabalhos Relacionados

Existem várias pesquisas relatadas na literatura que possuem algum tipo de similaridade com o nosso trabalho. Entretanto, a maioria se concentra em um ou outro aspecto da nossa proposta. Nenhuma delas abrange, com igual ênfase, as diversas atividades identificadas como fundamentais ao desenvolvimento de software baseado em componentes e as tecnologias consideradas no nosso trabalho. Descrevemos a seguir estas pesquisas, agrupando-as de acordo com os seguintes aspectos: abordagens que especificam infra-estruturas de reutilização como um todo, abordagens que enfocam a construção de modelos conceituais, abordagens que discutem a

especificação de arquiteturas de software do domínio e abordagens que enfocam a especificação de métodos de ED.

Em relação à especificação de **infra-estruturas** voltadas para o apoio ao desenvolvimento de software baseado em componentes, temos algumas propostas interessantes:

- O grupo de pesquisa da Universidade do Colorado, liderado pelo pesquisador Gerhard Fischer [FIS96], enfoca a modelagem do domínio utilizando técnicas de IA. Fischer e sua equipe disponibilizaram, desde o início dos trabalhos, em 1992, uma série de ambientes de projeto, denominados por eles de Ambientes de Desenvolvimento Orientados a Domínio (ADSDs). Estes ambientes cobrem diversos domínios de aplicação, indo do domínio de interfaces gráficas, através do ambiente FRAME [FIS95b], passando pelo domínio de projetos de cozinha, através do JANUS [FIS95b], e menus viva-voz para telefonia, através do VDDE [FIS95b], até ambientes para projetos de redes (NetDE) [SUM96] e projetos multimídia (EMMA) [NAK96]. Analisando os projetos até hoje apresentados pela equipe, podemos notar uma clara tendência para a ênfase no entendimento e especificação dos conceitos do domínio. Mecanismos como o de cooperação e o sistema de crítica, entre outros [VIL97], que tornam a abordagem de Fischer inovadora, são totalmente voltados para auxiliar na especificação e entendimento dos conceitos do domínio. Além disso, os projetos de Fischer privilegiam a especificação de ambientes para domínios específicos, quando comparado com outros projetos, como o de Gomaa a seguir, que privilegia o desenvolvimento de um ADSD genérico.
- O trabalho de Gomaa [GOM96] e sua equipe enfoca a criação de uma estrutura de reutilização baseada na automatização de seu método de engenharia de domínio, o EDLC (*Evolutionary Domain Life Cycle*), criando assim um ADSD genérico, denominado KBSEE (*Knowledge Based Software Engineering Environment*). O ambiente ainda está em fase de protótipo [GOM96] e seu objetivo principal é ser independente de um domínio específico, podendo, portanto, armazenar conhecimento a respeito de diversos domínios de aplicação, auxiliando assim no desenvolvimento de aplicações específicas cujo conhecimento do domínio esteja armazenado no repositório do ambiente.

- Outras pesquisas estão sendo realizadas na Universidade de Nebraska, pelo pesquisador Henninger e sua equipe [HEN95], [HEN97]. Este grupo propõe uma infra-estrutura de reutilização baseada na especificação de um repositório de conhecimento sobre projetos de software em um dado domínio. Os estudos enfocam, principalmente, na criação de mecanismos para a especificação do conhecimento e a utilização deste conhecimento através de ferramentas baseadas em esquemas de classificação similares ao de Prieto [PRI87]. Uma característica interessante desta abordagem é a ênfase na especificação de mecanismos que suportem a evolução do conhecimento armazenado no repositório.

Analisando estes trabalhos em relação à abordagem utilizada pela IRBMD, podemos fazer as seguintes considerações:

- O trabalho de Henninger [HEN97], apesar de estar ainda em fase inicial, possui uma abordagem semelhante à nossa, no que tange a utilização de um repositório centralizado para o armazenamento do conhecimento do domínio. No entanto, Henninger aposta apenas no armazenamento de conhecimento do projeto das aplicações, não se importando com o armazenamento de conceitos mais abstratos sobre um dado domínio. Além disso, não utiliza nenhum tipo de processo sistemático para a especificação do conhecimento do domínio no repositório. Entretanto, especifica métodos de classificação e busca do conhecimento do domínio que podem ser interessantes para a nossa abordagem;
- Comparando o trabalho da equipe de Fischer com a IRBMD, podemos notar que os ambientes de projeto de Fischer, principalmente o EMMA [NAK96], possuem uma preocupação maior com a representação de modelos conceituais. Entretanto, não notamos uma preocupação no detalhamento de um método para estruturação da sua base de conhecimento. Neste sentido, apenas são descritos alguns passos básicos, que são: *Seeding*, que consiste na criação de uma estrutura inicial para o ADSD; *Evolutionary Growth*, que tem como objetivo a evolução do conhecimento do domínio; e *Reseeding*, com o objetivo de agregar ao ADSD o conhecimento do domínio adquirido na fase anterior. Analisando estes passos básicos em relação à utilização de um processo de ED e a sua sistematização, através da adoção de um método, consideramos os passos descritos por Fischer muito abstratos, não definindo, de forma concreta, quais serão os resultados de cada fase. Além disso, todos os ambientes de projeto até então propostos pela equipe são específicos para

domínios pré-definidos, não podendo portanto serem utilizados para armazenar conhecimento a respeito de outros domínios de aplicação. Alguns autores, como Garlan [GAR96] e Ning [NIN94] em [VIL97], discordam desta abordagem, alegando que a produtividade alcançada com a utilização de tais ambientes pode não suplantar os custos de desenvolvê-los.

- Em relação a nossa proposta, o KBSEE apresenta algumas similaridades tais como: a adoção de um método para sistematizar a engenharia de domínio, o que garante a consistência entre os modelos; e a especificação de modelos de domínio em diversos níveis de abstração, utilizando principalmente o paradigma orientado a objetos. Entretanto, em relação a armazenagem dos dados, o KBSEE requer transformações entre diferentes representações e bases de dados. Isto acarreta em armazenamento de informações redundante e queda de desempenho. O KBSEE utiliza, inicialmente, os serviços de um Sistema Baseado em Conhecimento para armazenar os conceitos iniciais do domínio, na forma de regras. Em uma segunda etapa, com o objetivo de checar a consistência entre estes conceitos iniciais, transforma os conceitos armazenados na forma de regras para um modelo relacional, utilizando para o armazenamento deste modelo relacional, um SGBD. Posteriormente, transforma o modelo relacional para um modelo de objetos, utilizando um gerente de objetos, desenvolvido no contexto do projeto, para o armazenamento deste último modelo. Analisando a descrição deste sistema de armazenamento feita em [GOM96], podemos notar que não existe nenhuma preocupação com a integração destas diversas representações e bases de dados. Não existe um modelo integrador, que seria o componente responsável pela integração e consistência entre as diferentes representações. Neste sentido, a nossa proposta difere da de Gomaa, sendo baseada na utilização de uma única base para o armazenamento dos modelos de domínio, utilizando para isso a estrutura de um SGBD, o que leva a um desempenho melhor por utilizar uma estrutura de armazenamento adequada para dados complexos.

Em relação à especificação de modelos de domínio, mais especificamente **modelos conceituais**, principalmente utilizando conceitos de Sistemas Baseados em Conhecimento, temos também alguns trabalhos interessantes:

- A equipe do projeto SHADE [GRU92], da Universidade de Stanford, é um grupo de pesquisa especializado no suporte ao conhecimento do domínio na especificação de projetos, que, apesar de não ser especificamente voltado para a construção de

software, possui uma abordagem interessante em relação a representação do conhecimento do domínio. O SHADE é baseado na utilização de ontologias para a especificação de componentes reutilizáveis em projetos. No entanto, o SHADE é baseado em um formalismo muito grande e permite o suporte apenas às fases iniciais do processo de desenvolvimento (análise e projeto), além de não ser especificamente construído para o desenvolvimento de software baseado em componentes, sendo mais utilizado no contexto de projetos de engenharia.

- As pesquisas realizadas na Universidade de Amsterdam, através do projeto Kactus [LAR96], [HAR98], que também se baseia na utilização de ontologias para a especificação de aplicações em domínios específicos, é também bastante interessante. O projeto utiliza uma biblioteca de ontologias, em diversos níveis de abstração, para auxiliar na especificação de aplicações dos mais variados domínios. O Kactus possui ainda um conjunto de ferramentas, o VOID, que auxilia na reutilização desta biblioteca de ontologias. Neste sentido, o Kactus propõe uma abordagem semelhante a da nossa proposta, onde podem ser especificadas aplicações em domínios variados. Entretanto, apesar de ser uma proposta interessante, o Kactus não pode ser classificado como realmente uma infra-estrutura de reutilização, pois somente específica modelos conceituais, além de não adotar nenhum método que sistematize a utilização das ontologias.

Trabalhos mais relacionados com a especificação de **arquiteturas de software** para domínios específicos são também descritos na literatura:

- No contexto dos projetos espaciais da NASA, foi desenvolvido o ambiente, AMPHION [LOW95], [LOW97], que é um ambiente orientado a domínio, baseado em especificações formais e síntese dedutiva de aplicações no domínio. A grande vantagem do AMPHION, segundo os autores, está na especificação formal, provendo uma representação abstrata e sem ambigüidades dos requisitos do usuário. Assim como o KBSEE, é também um ambiente genérico que pode ser especializado para domínios específicos através do uso de teorias acerca do domínio e táticas para a prova destas teorias. O modelo de domínio utilizado pelo AMPHION é uma espécie de modelo arquitetural, com grande formalismo, que possui uma interface gráfica para facilitar a sua manipulação.
- O grupo de Garlan e Shaw [GAR96] desenvolveu também um ambiente para a especificação de descrições arquiteturais, denominado AESOP. O AESOP é um

ambiente arquitetural no sentido que provê suporte à representação, customização e análise de arquiteturas de software. Seu principal objetivo é prover um arcabouço que combine os benefícios das ADLs *(Architecture Description Languages*) genéricas e permita a especificação de arquiteturas específicas para diferentes domínios. O AESOP permite ao arquiteto de software definir um dado estilo e rapidamente construir um ambiente de projeto que suporte aquele estilo. Portanto, a abordagem do AESOP reduz os custos de se produzir um ambiente de desenvolvimento de software orientado a domínio, permitindo o desenvolvimento incremental de novos estilos e reutilizando a infra-estrutura de suporte do ambiente para a criação de novas ferramentas específicas para um dado estilo.

- Outro trabalho interessante é o trabalho desenvolvido no contexto do projeto OSVA [CZA97], cujo objetivo é permitir a reutilização de componentes em ambientes distribuídos, utilizando arquiteturas de software OO. Conceitos como *framework*s, arquiteturas de software e análise de domínio são utilizados na pesquisa. No entanto, as pesquisas ainda estão sendo iniciadas e resultados concretos ainda são incipientes.

Analisando os trabalhos acima podemos notar algumas similaridades em relação à nossa proposta. O nosso trabalho assemelha-se com os trabalhos apresentados em [GAR96] e [LOW97] no sentido de especificar arquiteturas de software para domínios específicos. No entanto, ambos os projetos baseiam suas descrições arquiteturais em abordagens formais, utilizando ADLs (*Architectural Description Languages*), no caso do AESOP [GAR96], e teoremas matemáticos, no caso do AMPHION [LOW97]. O AMPHION utiliza ainda uma abordagem gerativa para a especificação de aplicações do domínio. A descrição arquitetural utilizando ADLs possui diversos seguidores, no entanto, alguns autores, como Bosh [BOS97] acreditam que a utilização de construções como padrões de projeto facilitam a transição das construções arquiteturais para as estruturas codificadas em linguagens de programação, principalmente linguagens de programação OO. Como o nosso trabalho baseia-se principalmente no uso do paradigma OO, como descreveremos no capítulo 2, optamos por utilizar construções arquiteturais já bastante utilizadas em especificações orientadas a objetos como padrões [BUS96] e *framework*s [FAY97]. Sendo assim, apesar das interseções com os trabalhos descritos acima, o nosso trabalho possui como diferencial, além da especificação de padrões baseados tanto em estilos arquiteturais,

quanto baseados em informações específicas do domínio de aplicação [FOW97], a especificação de uma infra-estrutura de reutilização completa, que, além de especificar arquiteturas software, utiliza representações em níveis mais altos de abstração, como modelos conceituais. Além disso, a nossa proposta preocupa-se, também, com a especificação das operações que irão ser realizadas sobre os modelos e a sistematização destas operações, o que permitirá a criação consistente dos modelos.

Em relação à utilização de um **método de ED**, existem também algumas pesquisas que devem ser analisadas:

- o trabalho desenvolvido no centro de pesquisas em telecomunicações CSELT-STET, na Itália, pela pesquisadora Maccario [MAC97], é bastante semelhante ao nosso. O trabalho de Maccario concentra-se na especificação de um método de engenharia de domínio que contemple questões como distribuição e modelagem OO. Além disso, o método é customizado para atuar sobre um *framework* de componentes reutilizáveis, na área de telecomunicações, já existente no CSELT, que é o *TINA architectural framework.* A diferença em relação ao método que será especificado no contexto da nossa proposta, é que o trabalho de Maccario não explora conceitos OO em níveis mais altos de abstração, como padrões [BUS96]. Além disso, a abordagem de Maccario restringe-se à utilização de modelos OO para a representação do conhecimento do domínio, ao passo que, na nossa proposta, outros conceitos também serão utilizados, como por exemplo, o uso de técnicas de IA.
- Outra abordagem interessante em relação à especificação de métodos é a proposta de Irving [IRV96]. Irving propõe a utilização do conceito de padrões em um método de desenvolvimento acoplado a um modelo de ciclo de vida dual, baseado em reutilização. No entanto, Irving concentra-se na especificação de um mecanismo de classificação de padrões a serem utilizados no contexto do método, contemplando, nesta classificação, apenas padrões de projeto. No contexto da nossa proposta, a utilização de padrões será realizada em um contexto mais amplo, utilizando padrões de análise e arquiteturais, além de padrões de projeto.
- Shu [SHU96] aborda também, em uma proposta bastante interessante, a adoção sistemática de conceitos de padrões em métodos OO como o OMT [RUM91] e OOSE [JAC94]. A abordagem de Shu, apesar de ser específica para métodos de desenvolvimento OO, ao passo que a nossa abordagem concentra-se na

especificação de um método de engenharia de domínio, mostra-se promissora no sentido de prover os passos básicos para a inserção de conceitos de padrões em um método genérico.

1.5 Organização do documento

Com o objetivo de demonstrar a viabilidade desta proposta, este documento está organizado nos seguintes capítulos.

Neste capítulo, apresentamos as principais motivações, objetivos e resultados esperados, além de analisar alguns trabalhos cujas abordagens possuem semelhanças com o nosso.

No capítulo 2 é apresentada a proposta da infra-estrutura de reutilização, descrevendo seus principais componentes e que tecnologias serão utilizadas para a especificação destes componentes, além de analisar os requisitos básicos de gerenciamento e armazenamento que uma infra-estrutura deste tipo pode necessitar.

O capítulo 3 apresenta algumas conclusões e como será o desenvolvimento deste trabalho daqui por diante.

O Anexo contém uma descrição resumida dos principais conceitos envolvidos na engenharia de domínio e das principais formas de representação e utilização de modelos de domínio atualmente disponíveis

2. Proposta de uma Infra-estrutura de reutilização baseada em modelos de domínio

2.1 Introdução

Apresentamos neste capítulo, a proposta da Infra-estrutura de Reutilização Baseada em Modelos de Domínio (IRBMD). Conforme descrito no capítulo anterior, a IRBMD será composta por um conjunto de modelos de domínio, em diferentes níveis de abstração, que ajudarão na reutilização de conceitos de domínio (modelos conceituais), na reutilização dos componentes codificados (modelo implementacional), que representam estes conceitos do domínio, passando pela reutilização da estrutura que coordena o inter-relacionamento entre estes conceitos (arquitetura de software do domínio). No entanto, o processo de especificação desta infra-estrutura não é trivial. Os conceitos e tecnologias envolvidos precisam ser detalhados e avaliados de forma a verificar a viabilidade da proposta. Sendo assim, abordamos ainda neste capítulo, quais as tecnologias[2] que, no momento, parecem mais adequadas para o nosso contexto, e qual a direção que iremos seguir em relação a estas tecnologias. Além disso, a fim de que possamos implementar a IRBMD de forma eficiente, apresentamos uma descrição dos principais requisitos de gerenciamento e armazenamento da IRBMD, apresentando algumas tecnologias que podem ser utilizadas para a implementação destes requisitos.

Para isso, o capítulo está dividido da seguinte forma: na seção 2.2 apresentamos quais tecnologias serão envolvidas na representação dos modelos de domínio; na seção 2.3, descrevemos algumas tecnologias envolvidas no processo de criação e utilização dos modelos de domínio; na seção 2.4 abordamos as tecnologias e conceitos a serem utilizados na especificação de um método de ED adequado à IRBMD; a seção 2.5 trata de um estudo de quais são os requisitos de gerenciamento e armazenamento mais adequados para o suporte a IRBMD; e na seção 2.6 apresentamos uma primeira visão diagramática da infra-estrutura.

2.2 Tecnologias a serem utilizadas na representação dos modelos de domínio

Segundo Vasconcelos [VAS95] existem três aspectos principais que devem ser observados na representação do conhecimento, de forma genérica:

[2] Detalhes sobre os conceitos e tecnologias envolvidas podem ser encontrados no anexo

1. **Linguagem**: que diz respeito ao vocabulário e sintaxe utilizados na representação;
2. **Implementação**: refere-se à forma de armazenamento desta representação;
3. **Apresentação**: refere-se a como a representação será apresentada ao usuário.

Analisando os conceitos envolvidos na representação dos modelos de domínio da IRBMD, podemos observar que o paradigma OO apresenta uma série de vantagens que nos leva a considerar a sua utilização como forma de representação básica para estes modelos de domínio, principalmente em relação à implementação da representação. Algumas das vantagens da utilização de conceitos OO na representação dos modelos de domínio são:

- a modelagem OO, por ser considerada uma abordagem semi-formal, permite a checagem e análise de certas **consistências** em um modelo especificado segundo este paradigma. No contexto da IRBMD, esta é uma característica importante, pois garante, pelo menos em parte, a especificação de estruturas consistentes para a representação do conhecimento sobre o domínio;
- a modelagem OO, como já exaustivamente descrito na literatura, facilita a **transição** entre modelos de análise e projeto. Como a essência dos modelos de domínio, é a mesma dos modelos de análise, projeto e implementação convencionais, a modelagem OO pode também facilitar a transição dos modelos conceituais do domínio para a arquitetura de software do domínio e da arquitetura para o *framework* de componentes reutilizáveis;
- atualmente, o desenvolvimento OO está sendo considerado como o **paradigma padrão** a ser adotado no desenvolvimento de software [PRE97]. No mercado americano, principalmente, a adoção da OO como paradigma de desenvolvimento já é uma realidade há algum tempo e, no Brasil, as pesquisas [DEV97] apontam para um crescimento cada vez maior. Com isso, uma técnica de suporte a reutilização que seja baseada em construções OO terá grande probabilidade de ser utilizada pelo seu principal público alvo, ou seja, as empresas.
- outra característica interessante da OO, que mostra-se adequada na representação dos modelos, é a facilidade de **evolução** de suas construções, principalmente através dos mecanismos de herança e composição, o que possibilitará a definição de novos conceitos do domínio.

No entanto, apesar destas vantagens e de ter um estreito relacionamento com os modelos que irão ser criados pelo desenvolvedor de aplicações no domínio[3], a modelagem OO não captura toda a semântica do domínio, ou seja, conforme destaca Ottinger [OTT97], o modelo OO não é a resolução de todos os problemas de modelagem de conceitos, mas pode ser um ponto de partida para a representação destes modelos. Neste sentido, os aspectos da representação descritos por Vasconcelos [VAS95] como linguagem e apresentação podem necessitar de outras formas de expressão como, por exemplo, o uso de redes semânticas, como linguagem de representação, e hipermídia, como forma de apresentação da representação, entre outros.

Sendo assim, quando analisamos o contexto de uma infra-estrutura de reutilização, onde a forma de apresentação e expressão (linguagem) dos modelos é uma característica importante, principalmente para facilitar o entendimento dos usuários, o uso de modelos OO puros, isto é, modelos que utilizam apenas os conceitos básicos do paradigma, não garante a representação de forma adequada dos conceitos do domínio. Desta forma, precisamos utilizar outros tipos de construções para modelar os conceitos do domínio tais como representação hipermídia, padrões [GAM94], conceitos de IA [RUS95], entre outros.

A seguir, apresentamos as tecnologias envolvidas na representação dos modelos de domínio (nos seus três níveis de abstração, isto é, modelos conceituais, arquitetura de software do domínio e modelo implementacional), que, a princípio, pretendemos utilizar.

2.2.1 Modelos Conceituais

Na representação dos modelos conceituais, devemos ter uma preocupação especial com a facilidade de entendimento e reconhecimento dos conceitos por parte dos usuários da IRBMD, ou seja, por parte dos especialistas do domínio, engenheiros do domínio e engenheiros de software de aplicações no domínio.

A literatura especializada, principalmente na área de Sistemas Baseados em Conhecimento (SBC) [ANG96], ressalta que uma das maneiras de facilitar a interação do usuário com os conceitos de um dado domínio é apresentar estes conceitos na

[3] É importante ressaltar que a infra-estrutura de reutilização será utilizada como apoio no desenvolvimento de software OO.

forma de um modelo hipermídia. Algumas vantagens da utilização de hipermídia ressaltada pela literatura em geral [BRA95], [VAS95], [ANG97] são: proporciona conectividade entre as informações; oferece uma interface amigável, facilita a busca por conceitos específicos, entre outras.

A forma de expressão (linguagem) dos conceitos do domínio também é importante. Em relação a este aspecto, pretendemos agregar técnicas já descritas na literatura. A forma básica que, a princípio, iremos utilizar é de uma estrutura similar aos padrões de análise[4] [FOW97]. A vantagem da utilização dos padrões de análise é que estes permitem, além de se ter um conhecimento informal de conceitos do domínio, embutir no padrão um modelo OO sobre aquele conceito, em um nível de abstração adequado, o que permite uma maior formalização do conceito, além de facilitar a transição para a arquitetura de software do domínio.

No entanto, o uso de padrões de análise, apesar dos mesmos serem interrelacionados, formando um sistema de padrões, não conseguem capturar, de forma adequada, a visão global dos conceitos do domínio.

Esta visão global, necessária tanto para o entendimento do domínio, quanto para facilitar a transição para o modelo arquitetural, será expressa através da utilização de modelos de *features* [COH94], diagramas de classes, de atividades e de empacotamento, sendo este último utilizado para a captura de informações arquiteturais já no modelo conceitual, dando uma visão inicial do relacionamento entre os subdomínios que compõem o domínio. Sendo assim, nesta visão global, o modelo OO estará sendo utilizado, segundo o conceito de representação descrito em [VAS95], tanto como linguagem de representação (uso da sintaxe do modelo OO) quanto como forma de apresentação (diagramas OO).

Além disso, a fim de que possamos inferir novas informações e tomar decisões em relação aos conceitos do domínio a serem utilizados, em um dado contexto, necessitamos representar o conhecimento em uma forma que permita a atuação destes mecanismos. Sendo assim, utilizaremos também a representação na forma de regras de lógica, com o objetivo de facilitar a mineração de conceitos do domínio, utilizando os modelos de domínio.

[4] Padrões de análise são grupos de conceitos que representam uma descrição comum em uma modelagem de um domínio. Os padrões de análise provêem diretivas de como modelar conceitos de um dado domínio utilizando o paradigma OO.

Portanto, de forma geral, segundo os três aspectos descritos em [VAS95], utilizaremos as seguintes tecnologias de representação do conhecimento para a representação dos modelos conceituais:

- **linguagem** de representação: uso de padrões de análise, modelos OO e modelos baseados em regras;
- **apresentação** da representação: interface hipermídia e diagramas OO;
- **implementação** da representação: modelo de metadados OO e representações na forma de regras de lógica, através do uso do modelo de armazenamento de um gerente de objetos[5];

2.2.2 Arquitetura de Software do Domínio

Na representação da arquitetura de software do domínio também será utilizado, inicialmente, o conceito de padrões como linguagem de representação. A arquitetura genérica[6], ou seja, os estilos arquiteturais relevantes para o domínio, serão representados por estruturas semelhantes aos padrões arquiteturais de Buschmann [BUS96] e através dos padrões de projeto de Gamma [GAM94]. Os padrões arquiteturais e de projeto, juntos, formarão um sistema de padrões, onde os padrões de projeto serão utilizados para explicar, em maior detalhe, um estilo arquitetural representado por um padrão arquitetural. Na Figura 2-1 apresentamos esta organização. Devemos ressaltar que, tanto os padrões arquiteturais quanto os padrões de projeto, terão, embutidos na sua estrutura, modelos OO.

É importante ressaltar que, em termos de representações arquiteturais, o que teremos efetivamente armazenado como modelo, serão: i) os padrões de projeto e arquiteturais, que descrevem os estilos; ii) os padrões de análise, representando os conceitos do domínio; e iii) um conjunto de arquiteturas de software de aplicações do domínio já especificadas. Note que, na verdade, não temos armazenada uma arquitetura de software do domínio. Para cada aplicação especificada utilizando a infra-estrutura, será criada uma arquitetura de software do domínio customizada para a aplicação, ou seja, embutindo conceitos do domínio e estilos arquiteturais relevantes para aquela aplicação particular, ou seja, na especificação da arquitetura da aplicação, não são necessariamente utilizados todos os conceitos do domínio.

[5] O modelo de armazenamento será detalhado na seção 2.5
[6] O conceito de arquitetura genérica está descrito no Anexo

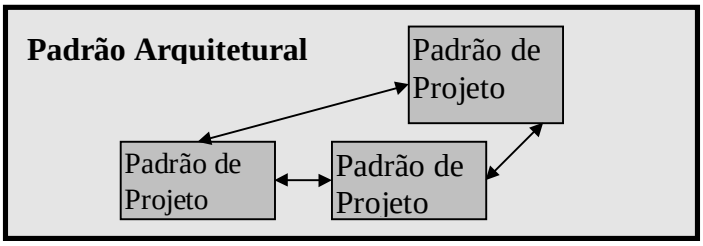

Figura 2-1- Formação de um padrão arquitetural e seu relacionamento com padrões de projeto

Estas arquiteturas de software das aplicações, que serão armazenadas na infra-estrutura, serão, essencialmente, modelos OO, representando os relacionamentos e interação entre o conjunto de componentes reutilizáveis. Neste contexto, a utilização de diagramas de distribuição e diagramas de classes (descritos na notação UML [FOW97a]) para a apresentação da representação serão importantes.

Desta forma, na representação da arquitetura de software do domínio utilizaremos as seguintes técnicas: como **linguagem** da representação, utilizaremos padrões arquiteturais, de projeto e de análise, além do modelo OO; como **apresentação** da representação, utilizaremos os diagramas OO, templates dos padrões e interface hipermídia; e como **implementação** da representação, utilizaremos metadados segundo o paradigma OO, através do uso do modelo do gerente de objetos.

2.2.3 Modelo implementacional *(Framework)*

O modelo implementacional, isto é, o *framework*, neste contexto, será um conjunto de classes/objetos codificados em uma linguagem de programação, segundo uma abordagem de *framework* "caixa branca" [JOH93], ou seja, o *framework* poderá ser estendido e poderão ser adicionadas outras classes/objetos que não as componentes do conjunto de componentes reutilizáveis, para a formação da aplicação final. Neste sentido a representação do framework será especificada da seguinte forma: como **linguagem** de representação, utilizaremos o modelo OO, seguindo as restrições impostas pela linguagem de programação a ser utilizada; a **apresentação** será feita utilizando diagramas OO e a **implementação** será feita através do uso do modelo de dados do gerente de objetos.

2.3 Especificação e utilização dos Modelos do Domínio

Além da representação dos modelos de domínio, a IRBMD deve também disponibilizar ferramentas que permitam aos seus usuários especificar e utilizar estes modelos de domínio. Neste sentido, ferramentas para elicitação de requisitos, gerenciamento de padrões, gerenciamento de componentes reutilizáveis, entre outras, devem ser especificadas.

Os métodos de ED atualmente disponíveis propõem algumas heurísticas, principalmente para a elicitação de requisitos, que podem ser analisadas. No entanto, analisaremos também algumas técnicas utilizadas em Sistemas Baseados em Conhecimento, e em outras áreas de IA, como agentes inteligentes, com o objetivo de prover maior facilidade para o usuário na utilização destas ferramentas.

Em relação ao gerenciamento de padrões, já existem alguns estudos [BUS96] no sentido de disponibilizar ferramentas para auxílio na definição, evolução e busca de padrões. Estas ferramentas são baseadas, principalmente, em esquemas de classificação, que são aplicados aos padrões. Além disso, heurísticas de como utilizar padrões no processo de desenvolvimento de software como um todo, também já foram definidas em trabalhos como os de [IRV96] e [BUS96]. Sendo assim, na especificação de ferramentas para o gerenciamento de padrões, levaremos em conta os estudos realizados, adaptando-os para a nossa proposta.

No gerenciamento de componentes reutilizáveis, existem também alguns estudos realizados, como em [GAC95] e [HEN95], que apresentam-se promissores, principalmente na utilização de técnicas para busca de componentes. Estes estudos também serão analisados no contexto da nossa proposta.

No contexto de utilização dos modelos de domínio, analisando alguns trabalhos na área de Sistemas Baseados em Conhecimento [HAY94], podemos notar que, uma das formas mais utilizadas de interação do usuário com o conhecimento do domínio, é através da utilização de interfaces hipermídia. No entanto, acreditamos que somente a utilização da hipermídia, não garante que o usuário irá realmente assimilar os conceitos do domínio e nem que a navegação que o usuário irá realizar pela hipermídia, o levará a conceitos que realmente o interessam. Apesar das pesquisas em hipermídia apontarem para o uso de técnicas de autoria e do conceito de caminho para facilitar a navegação em um sistema hipermídia, acreditamos que esta ajuda não atende totalmente às nossas necessidades, por serem estas técnicas estáticas, ou seja, não

levam em consideração a evolução do conhecimento armazenado e nem o conhecimento adquirido pelo usuário durante a navegação. Por isso, além da utilização de uma interface hipermídia, achamos interessante que exista uma forma de guiar, de maneira dinâmica, o usuário na sua navegação. Acreditamos que este guia deva, a partir das necessidades iniciais do usuário[7], em relação à aplicação a ser desenvolvida e às escolhas e informações que o usuário disponibilize para a infra-estrutura, apresentar quais os melhores caminhos de navegação a serem seguidos e que tipo de conhecimento é mais adequado para ele. Na verdade, este guia será um agente inteligente [RUS95], nos moldes dos agentes de informação ou robôs de informação usados para guiar na navegação Internet. Para que este agente inteligente possa atuar sobre os nós hipermídia e sobre o conhecimento disponibilizado pelo usuário, uma proposta interessante é a apresentada em [LUK97], que propõe, para busca na rede (Internet), que cada página possua uma descrição, baseada em uma ontologia definida para a busca. Este descrição permite ao agente inferir se realmente o conhecimento embutido na página é de interesse do usuário. Acreditamos que uma proposta similar a esta seja plausível para o nosso contexto, disponibilizando, em cada nó hipermídia, uma ontologia que facilite a busca do usuário por informações.

Outra questão importante, a qual acreditamos ser crucial para o sucesso da infra-estrutura é a capacidade do usuário, a partir do conhecimento adquirido através dos modelos conceituais, definir quais elementos da arquitetura de software do domínio são adequados para a aplicação a ser desenvolvida. Para isso, necessitamos disponibilizar formas de combinar os conceitos embutidos nos padrões de análise e nos diagramas conceituais OO, que é onde estão os conceitos do domínio, e nos padrões arquiteturais, que é onde estão as informações sobre os estilos arquiteturais mais adequados para aquele domínio. Acreditamos que esta "combinação" de conceitos possa ser realizada por um agente inteligente, que com a ajuda de arquiteturas de aplicações já desenvolvidas no domínio, que devem também ser armazenadas na infra-estrutura, e do engenheiro de software, permitirá a especificação da arquitetura de software mais adequada para a aplicação a ser desenvolvida. Portanto, o agente que irá atuar nesta fase utilizará, para a criação da arquitetura de software da aplicação, os padrões de análise, os modelos conceituais OO, os padrões arquiteturais, informações

[7] Estamos considerando como principal usuário deste guia, o desenvolvedor da aplicação.

disponibilizadas pelo engenheiro de software e a base de arquiteturas de software de aplicações anteriores. A eficiência do agente irá aumentando a medida que mais aplicações forem desenvolvidas utilizando a infra-estrutura. A disponibilização de informações por parte do engenheiro de software vai diminuindo a medida que a base de arquiteturas de aplicações for aumentando, passando de uma atuação enfática, no início da utilização da IRBMD (necessitando possivelmente, neste início, da ajuda de um engenheiro de domínio), para uma atuação menor, a medida que a base de arquiteturas de aplicações for aumentando.

2.4 Tecnologias e conceitos envolvidos na especificação de um método de ED

No Anexo deste trabalho, destacamos a importância da especificação de um método de ED para auxiliar na construção e evolução dos modelos de domínio de uma infra-estrutura de reutilização. Apresentamos também uma análise das principais características de alguns métodos de ED considerados mais proeminentes no mercado, a saber: FODA [COH94], ODM [ASS96], EDLC [GOM96] e KAPTUR [KLI95].

Analisando estes métodos e as necessidades da IRBMD, podemos considerar que nenhum dos métodos apresentados é totalmente adequado para as nossas necessidades. Com isso, consideramos necessário para os nossos propósitos, a especificação de um método que cubra as deficiências[8] encontradas nos métodos analisados e absorva suas vantagens.

Dentre as deficiências encontradas nos métodos em relação ao suporte à IRBMD temos:

- nenhum dos métodos cobre, com igual ênfase, todas as fases da ED. Como a IRBMD utiliza modelos que abrangem todas as fases da ED, necessitamos de um método que permita a sistematização de todas as fases;
- nenhum dos métodos contempla, de forma detalhada, uma fase de análise de risco do domínio. Este conceito de análise de risco é importante no contexto de uma infra-estrutura de reutilização genérica, como a que estamos propondo, uma vez que os custos de se desenvolver modelos de domínio para um determinado domínio de aplicação podem ser altos e, por isso, a infra-estrutura só será viável, se o ganho

[8] É importante ressaltar que as deficiências que encontramos nos métodos analisados são em relação às necessidades específicas da IRBMD.

com o desenvolvimento de aplicações naquele domínio suplantar os custos do desenvolvimento dos modelos. Por isso, uma fase de análise de risco é premissa básica para um método a ser utilizado pela infra-estrutura;

- conforme já ressaltado por outros autores, uma das dificuldades encontradas na reutilização baseada em modelos, é a transição dos modelos conceituais para os modelos arquiteturais. Alguns métodos analisados no capítulo 2 propõem algumas heurísticas para esta transição e alguns trabalhos como [CAP97] propuseram também algumas abordagens para esta transição. No entanto, nenhuma abordagem sistemática para esta transição foi proposta. Acreditamos que seja interessante esta transição sistemática no contexto da IRBMD, uma vez que o objetivo desta é permitir a reutilização em altos níveis de abstração e que esta reutilização possa ser propagada até os níveis mais baixos de abstração. Para que isto seja realidade, a transição entre os modelos tem que ser eficaz.

Além de abranger as deficiências levantadas em relação aos outros métodos, a abordagem que adotaremos privilegiará a utilização do paradigma OO como base para o desenvolvimento do método. A utilização do paradigma OO se justifica pelas mesmas razões explicitadas na seção 2.2.

No entanto, apesar de utilizarmos a OO como base para a especificação da infra-estrutura de reutilização, a utilização de métodos OO convencionais, como OMT [RUM91] e OOSE [JAC94], não garante a definição de modelos de domínio [SHU96], [MAC97], de forma adequada. Por isso, o que necessitamos é de uma adaptação das técnicas utilizadas pelos métodos de ED para a utilização do paradigma OO.

Apesar de já existirem métodos como o EDLC que utilizam a OO, estes métodos não disponibilizam, em termos de construções OO, todas as possibilidades que necessitamos para modelar o domínio. Shu [SHU96] ressalta esta deficiência nos métodos de desenvolvimento OO convencionais, alegando que as construções OO providas pelos métodos atuais não permitem a especificação de conhecimento em mais alto nível de abstração do que o de classes/ objetos e suas composições. As construções OO atuais provêem princípios e diretrizes para o desenvolvimento OO. No entanto, estas construções não “dizem” precisamente como especificar bons modelos, em contextos específicos, ou seja, não é provido nenhum tipo de mecanismo que facilite e direcione a especificação de bons projetos OO em um domínio específico. Acreditamos que, a tecnologia de padrões seja um caminho para a especificação deste

tipo de mecanismo. Shu ressalta esta abordagem, descrevendo os métodos OO como sendo provedores de regras gerais que aplicam-se a todas as situações, e o uso de padrões seria o complemento que faltava aos métodos OO para que sejam também providas regras que se apliquem a problemas específicos, especificando também soluções para este problemas específicos.

A abordagem de Shu, que se aplica apenas ao contexto de alguns métodos de desenvolvimento OO, como OMT [RUM91] e OOSE [JAC94], pode e deve ser estendida para o contexto dos métodos de ED, onde a utilização de padrões ganha uma dimensão muito maior do que a de Shu, que utiliza apenas a noção de padrões de projeto, devido principalmente à necessidade, no contexto dos métodos de ED, de estruturas que representem abstrações a nível de modelos conceituais e arquiteturais. Padrões de análise [FOM97] e arquiteturais [BUS96] poderiam ser utilizados neste contexto.

Além dos conceitos descritos nos parágrafos anteriores, acreditamos que, para a especificação de um método de ED, devemos basear o nosso trabalho em conceitos que já foram consagrados em outros métodos de ED e se aplicam também ao nosso contexto.

Algumas estruturas e conceitos, utilizados por alguns métodos analisados no anexo, parecem adequados para o propósito da infra-estrutura de reutilização. Dentre estes métodos, podemos destacar o FODA e ODM. O modelo de *features*, utilizado no FODA, é o modelo ideal para capturar as similaridades e deficiências entre os conceitos do domínio, além de permitir a modelagem dos serviços no contexto do domínio. No contexto da modelagem OO, o modelo de *features* poderia ser comparado aos *use cases*. A distinção realizada no ODM entre modelos descritivos e prescritivos é outro conceito interessante. Esta utilização de modelos prescritivos, ainda na fase de análise do domínio, permite a inserção de descrições arquiteturais no modelo conceitual, o que facilita a transição para os modelos arquiteturais. Além disso, tanto o ODM quanto o FODA têm grande preocupação com a facilidade de entendimento de seus modelos por parte dos usuários e esta é uma característica que queremos privilegiar na infra-estrutura de reutilização. Como consideração final em relação a utilização do FODA e ODM para a especificação do método de ED, devemos ressaltar que o FODA e o ODM são métodos de ED já consagrados na

literatura, sendo utilizados em diversos contextos (ex.: SEI e projeto CARDS), o que atesta sua maturidade.

Portanto, pelas razões explicitadas acima, pretendemos basear nosso método de ED em uma compilação dos métodos FODA e ODM. No entanto, como já ressaltamos, adotaremos o paradigma OO como base do nosso modelo. Para isso, necessitamos também agregar ao nosso método de ED, um método OO que contemple as construções básicas do paradigma, incluindo, principalmente, uma notação básica a ser utilizada. O UML (*Unified Modeling Language*) [FOW97a] é o método que escolhemos para prover construções e notações OO para o nosso método de ED. Dentre os motivos que nos levaram a optar pelo UML, podemos destacar:

- O UML será o método padrão (UML + Objectory (processo para sistematização do UML)), para a modelagem OO a ser adotado pelo OMG (Object Management Group) [FOW97a];
- o UML agrega conceitos de *use cases*, que, conforme já ressaltado neste trabalho, possui similaridades com o modelo de *features*, o que facilitará a composição do UML com o FODA e ODM;
- Algumas técnicas utilizadas no UML, e também em outros métodos OO, são adequadas para a especificação dos modelos de domínio, em diferentes níveis de abstração. Dentre estas técnicas, podemos destacar: os diagramas de classe que, em uma perspectiva conceitual, mostram-se adequados para capturar conceitos do domínio; diagramas de atividade, que complementam o diagrama de classes, descrevendo o *workflow* do domínio; e *use cases*, que auxiliam o engenheiro do domínio, com a ajuda de um especialista, a descrever algo que possa ter impacto nos modelos de domínio. Todas estas técnicas auxiliam ainda na captura de similaridades e diferenças entre os conceitos do domínio;
- construções como os diagramas de empacotamento, juntamente com o diagrama de classes, são uma possibilidade para a especificação de considerações arquiteturais no modelo conceitual e posterior especificação da arquitetura de software do domínio. Sendo assim, estas construções podem facilitar no processo de sistematização da transição entre os modelos conceituais e a arquitetura de software do domínio.

Além da composição inicial destes métodos (FODA, ODM e UML) e outros que se mostrem interessantes ao longo do processo, o método de ED que propomos terá definida uma fase de análise de risco. Analisando alguns trabalhos na área [QSOR95], pudemos notar que um caminho a ser seguido para esta análise de risco é a adoção de critérios de seleção para o domínio, atribuindo pesos para cada um dos critérios. Estes critérios devem ser definidos de forma a contemplar as necessidades da empresa onde será utilizada a infra-estrutura de reutilização.

Em suma, o método de ED para suporte a IRBMD contemplará os seguintes conceitos e tecnologias:

- será dividido em 4 fases principais, a serem detalhadas posteriormente: análise de risco do domínio, análise do domínio, projeto do domínio e implementação do domínio;
- o detalhamento das fases acima será baseado na compilação dos métodos FODA, ODM e UML, e outros que possuírem estruturas interessantes para a infra-estrutura de reutilização;
- definição de um conjunto de ferramentas que permitam a sistematização das atividades do método e a definição dos modelos do domínio.

2.5 Utilização de um gerenciador de objetos para o armazenamento dos modelos de domínio

Nas seções anteriores, descrevemos as principais tecnologias envolvidas na especificação da IRBMD. Apresentamos, resumidamente, as tecnologias que pretendemos utilizar na especificação da estrutura da IRBMD, através da representação dos modelos de domínio; operações, através da especificação e utilização destes modelos; e regras de integridade, através da especificação de um método de ED que sistematizará a criação e evolução dos modelos de domínio, de forma a garantir que estes sejam consistentes.

No entanto, para a implementação efetiva destas tecnologias, agrupadas no contexto da IRBMD, necessitamos da disponibilização de alguns serviços básicos de gerência e armazenamento de informações. Dentre os requisitos de gerência e armazenamento que julgamos que devem ser atendidos por estes serviços básicos, podemos destacar:

- necessidade de uma estrutura de representação do modelo da IRBMD, a fim de que possamos representar os modelos de domínio de forma adequada. Este modelo deve permitir expressar todos os conceitos do domínio, de forma adequada;
- necessidade de de manipulação que permitam a especificação de consultas de acordo com as estruturas utilizadas para a representação dos modelos de domínio;
- necessidade de um conjunto de serviços básicos que garantam a consistência da definição e manipulação dos modelos de domínio representados. Dentre estes serviços, podemos destacar a necessidade de mecanismos para a evolução das estruturas dos modelos de domínio, controle de acesso concorrente, entre outros.

Analisando a necessidade de disponibilização destes serviços para a IRBMD, poderíamos pensar, em um primeiro momento, em implementar estes serviços, partindo do zero, como um componente a mais da própria IRBMD. No entanto, a implementação destes serviços básicos implicaria no desenvolvimento de um componente de gerência tão complexo quanto o próprio desenvolvimento da IRBMD. Sendo assim, devemos analisar a possibilidade de reutilizar um componente de gerência que já tenha estes serviços disponibilizados, como é o caso dos Sistemas de Gerenciamento de Banco de Dados (SGBDs), alia a um modelo de representação poderoso, mecanismos de acesso adequados a este modelo e serviços de gerência adequados ao contexto.

Analisando o estado da arte atual dos SGBDs, podemos considerar como adequada a utilização de SGBDs Orientados a Objetos (SGBDOOs) para o suporte a gerência de dados da IRBMD, pois estes aliam um modelo de representação poderoso, utilizando o paradigma orientado a objetos, com a disponibilização de mecanismos de acesso e gerência de dados adequados ao modelo OO.

No entanto, apesar de toda a funcionalidade disponibilizada pelos SGBDOOs, algumas vezes o desempenho impõe um preço muito alto para a sua utilização. Um dos pontos que podem comprometer a eficiência e o desempenho nos SGBDOOs é a estrutura pesada que estes carregam, disponibilizando um grande número de funcionalidades como, mecanismos de versões sofisticados, controle de concorrência configuráveis, entre outros, que acabam por comprometer o desempenho do SGBDOO e também a sua portabilidade. Além disso, características como distribuição e paralelismo, ainda não foram muito bem exploradas no contexto dos SGBDOOs. Estas características são marcantes para a melhoria do desempenho dos SGBDs [TAV96].

Sendo assim, uma alternativa para amenizar estes problemas do uso de um SGBDOO, poderia ser utilizar um gerenciador aberto de objetos que possa ser configurado apenas com as características necessárias ao contexto onde será utilizado, como por exemplo, o sistema GOA++ [MAU97].

Em recente workshop onde foram discutidas as atuais tendências em SGBDs, Silberchatz e Zdornik [SIL97] destacam, no contexto dos novos tipos de aplicações que requerem os serviços de um SGBD, as vantagens de se utilizar um gerente de objetos flexível, que possua apenas os serviços que são necessários à aplicação, deixando de lado funcionalidades que tornariam a estrutura da aplicação pesada, sem serem utilizadas pela mesma, comprometendo a sua portabilidade e desempenho.

Baseados nesta premissa, pretendemos utilizar a estrutura de um gerente de objetos flexível, ou seja, que possa ser composto apenas por serviços que atendam realmente às necessidades da IRBMD, tornando-a desta forma mais eficiente e portável.

Para isso, analisaremos a seguir, baseados nos requisitos listados acima, quais os serviços que devem estar disponíveis em um gerente deste tipo de forma a atender as necessidades da IRBMD.

2.5.1 Modelo de Representação

Conforme descrito ao longo de todo o trabalho e principalmente na seção 2.2, a representação dos modelos de domínio requer a implementação de um conjunto de técnicas de representação do conhecimento sofisticadas, tais como: modelos de classes/objetos, estrutura de padrões, redes hipermídia, regras, entre outras.

Analisando os três aspectos de representação do conhecimento, descritos em [VAS95], podemos classificar as técnicas de representação do conhecimento descritas acima da seguinte forma: no aspecto de linguagem, temos o uso do modelo de classes/objetos, estrutura de padrões e modelo de regras; no aspecto de apresentação, temos o uso de redes hipermídia, diagramas OO e no aspecto de implementação temos o modelo básico de objetos e construções em lógica.

Levando em consideração as características do modelo OO tais como polimorfismo, relacionamentos recursivos, herança múltipla, objetos complexos, entre outras, podemos considerá-lo, no aspecto implementacional da representação como em parte adequado para a representação dos modelos de domínio. O uso de construções

baseadas em lógica, cobriria as necessidades não preenchidas pelo modelo OO, com a especificação de regras a serem utilizadas pelas ferramentas da IRBMD baseadas em técnicas de IA. Acreditamos que o uso de construções OO, aliado ao uso de construções baseadas em lógica, a fim de permitir a especificação de regras, cobriria por completo as necessidades de implementação da representação dos modelos de domínio, uma vez que o uso de regras permitiria a especificação de mecanismos de inferência necessários às ferramentas baseadas em agentes.

Em relação à especificação dos outros aspectos da representação do conhecimento, acreditamos que o modelo OO possa também servir de base para tal. As construções do modelo OO permitem a especificação de dados complexos, como dados multimídia e a interligação destes dados em uma rede, permitindo assim a formação de uma rede hipermídia, o que cobriria as necessidades básicas de apresentação da representação. Da mesma forma, o modelo OO permite a especificação de estruturas adequadas à especificação de padrões e os relacionamentos entre estes, em um sistema de padrões, atendendo os requisitos básicos em relação à linguagem da representação.

Sendo assim, o uso de um modelo de objetos completo, aliado a representações na forma de regras, seria adequado como modelo de representação de um gerente de objetos para suporte a IRBMD.

Analisando o padrão da ODMG (*Object Database Management Group*) [ODM97] para modelo de objetos, podemos considerar que, um gerente de objetos que atenda ao padrão ODMG em relação ao seu modelo, atenderá também aos nossos requisitos, pois o padrão ODMG comtempla todas as construções básicas e avançadas do modelo OO tais como: uso de IDO único, herança múltipla, conceito de chave, persistência por extensão da classe, relacionamentos inversos, entre outros. Além disso, a utilização de um modelo de objetos baseado no padrão ODMG adiciona uma vantagem a mais na estrutura de representação como um todo, que é a sua portabilidade, uma vez que, o ODMG é o padrão utilizado pela maioria dos SGBDOOs, facilitando assim, se for necessário, a troca do gerente de objetos por outro que atenda ao mesmo padrão. Outra consequência é a facilidade de interoperação com outras bases através do uso de uma arquitetura de interoperação, como a HIMPAR [PIR97], que disponibiliza serviços do modelo ODMG através do padrão CORBA.

No entanto, além do modelo de objetos, devemos agregar conceitos de lógica ao modelo de representação como um todo, a fim de atender à necessidades de mecanismos de inferência, como os que serão utilizados pelas ferramentas de elicitação dos requisitos. Para isso, pretendemos definir alguns serviços específicos que permitam agregar conceitos de lógica sobre o modelo de objetos do ODMG. Estes conceitos (construções OO segundo o ODMG [ODM97] e lógica) seriam portanto todos englobados na linguagem de definição de dados (LDD) do gerente de objetos, disponibilizando assim um mecanismo adequado para a especificação dos modelos de domínio da IRBMD.

2.5.2 Manipulação de Objetos

Em relação ao acesso aos modelos de domínio, o gerente de objetos deve prover uma linguagem de manipulação de dados (LMD) que permita o acesso fácil e eficiente aos modelos de domínio e seus componentes.

Como o modelo de representação a ser utilizado será baseado nos conceitos de OO e modelo de regras, a LMD também deve ser baseada nestes conceitos, provendo mecanismos que permitam a recuperação dos componentes representados utilizando OO e regras.

Além disso, uma característica que seria desejável em uma LMD a ser utilizada no contexto da IRBMD é a capacidade de realização de consultas incrementais e/ou parciais. Em [HEN97], [BRU97] foram realizados alguns estudos a este respeito. Outra característica que poderia ser interessante, principalmente para melhorar o desempenho da IRBMD como um todo, é a utilização de mecanismos para o processamento paralelo de consultas.

A utilização do padrão de LMD definido pelo ODMG [ODM97], através da definição das estruturas básicas de uma OQL (*Object Query Language*) atende às necessidades de recuperação de componentes representados segundo o modelo de objetos. Entretanto, seria necessário fazer extensões na linguagem no sentido de permitir a recuperação das informações representadas na forma de regras, nos moldes do trabalho de mineração de dados descrito em [MAU97], que estende a OQL do gerente de objetos GOA++, de modo a tratar regras.

Sendo assim, o gerente de manipulação a ser utilizado por um gerente de objetos para o suporte a IRBMD deve atender aos seguintes requisitos:

- ser eficiente, utilizando possivelmente processamento paralelo de consultas;
- permitir a recuperação dos componentes dos modelos de domínio utilizando construções OO e regras;
- uso de consultas incrementais;
- aderência da LMD ao padrão ODMG.

2.5.3 Conjunto de serviços básicos

Além de um modelo para a representação dos modelos de domínio e de um modelo de acesso eficiente, necessitamos também de outros controles que devem ser disponibilizados pelo gerente de objetos.

Uma característica básica dos modelos de domínio é a sua constante evolução em relação ao conhecimento que representam. Por isso, é necessário que o gerente de objetos disponibilize um mecanismo eficiente que suporte a evolução dos modelos de domínio armazenados no seu contexto, ou seja, suporte à evolução do esquema (metadados) dos modelos do domínio. O uso do modelo de objetos para a representação dos modelos por si só facilita a especificação deste mecanismo de evolução, uma vez que os mecanismos de herança múltipla e polimorfismo facilitariam a evolução, através da especialização dos componentes já representados. No entanto, devemos especificar como implementar efetivamente esta evolução. Atualmente, no contexto dos SGBDs são utilizados dois tipos de estratégias para permitir a evolução do esquema em SGBDs:

- atualização imediata: todos os objetos da base são automaticamente atualizados após a execução de uma função de conversão. Uma desvantagem desta estratégia é que o sistema tem que ser paralisado para se fazer as mudanças.
- atualização adiada: nesta estratégia, os objetos são atualizados apenas quando forem utilizados. Uma vantagem desta estratégia é que podemos economizar processamento e como desvantagem temos que guardar um histórico de todas as atualizações realizadas no esquema.

Analisando estas duas estratégias, parece-nos, a princípio mais adequado a utilização da estratégia de atualização imediata no contexto da IRBMD, uma vez que a evolução deste conhecimento será realizada apenas quando houver concordância em relação a esta evolução por parte do engenheiro do domínio e do especialista do domínio. Sendo assim, esta evolução poderá ser realizada *off line*.

Outros controles, tais como acesso concorrente e integridade dos dados, também são importantes no contexto da IRBMD. Uma das formas mais simples de preservar a integridade de um objeto é através do encapsulamento dos dados, que é uma técnica inerente ao paradigma da orientação a objetos. Um objeto, a fim de preservar as regras de encapsulamento, não pode ser acessado por um programa ou por outro objeto a não ser que seja através de sua interface. O encapsulamento é uma forma de preservar a integridade em uma granularidade fina, ou seja, a granularidade de um método. No entanto, temos outros mecanismos, como o uso de relacionamentos inversos, entre outros, que auxiliam também na especificação de restrições de integridade. Entretanto, a especificação de controle de integridade mais específicos, de acordo com as restrições de integridade do domínio, podem também ser necessários. No contexto dos modelos de objetos, o uso do padrão da ODMG, já garante controle de integridade de dados automática e customizada. No entanto, o uso de um sistema de regras (para a especificação de mecanismos de gatilho), que já era necessário para a representação de alguns componentes do modelo de domínio, pode facilitar ainda mais o controle da integridade dos dados.

Outro componente essencial do gerente de objetos para o suporte a IRBMD é o controle do acesso concorrente aos modelos de domínio. Geralmente, as atividades desenvolvidas no contexto de uma infra-estrutura de reutilização são caracterizadas por serem complexas, de longa duração e interativas. Para permitir a durabilidade e finalização correta de todas estas atividades, deve haver um mecanismo de controle de transações que permita a modelagem, monitoramento e controle de todos os acessos a componentes dos modelos de domínio. Alguns autores [BAR95] defendem, em casos similares ao de uma IRBMD, o uso de mecanismos de controle de transação sofisticados, levando em consideração que os acessos a serem realizados serão altamente concorrentes e prioritariamente com o objetivo de realizar atualizações. No entanto, analisando o contexto de utilização da IRBMD, podemos considerar que os acessos aos modelos de domínio e seus componentes serão, prioritariamente, acessos para leitura de dados, uma vez que a IRBMD será utilizada, principalmente, para "obter" e reutilizar conhecimento a respeito de um domínio de aplicação. Os acessos para escrita somente serão realizados, quando for necessário adicionar novos conhecimentos à base e/ou evoluir o conhecimento já existente. Sendo assim, podemos concluir que este tipo de acesso não será uma constante na IRBMD. Além disso, o

acesso concorrente para atualização a um mesmo item de dados raramente ocorrerá. Desta forma, acreditamos que um mecanismo de controle de concorrência padrão, por exemplo baseado no protocolo de duas fases [CAT94] poderá ser suficiente como forma de controlar o acesso concorrente.

O controle de segurança no acesso aos dados é outro mecanismo importante do gerente de objetos. Para permitir a implementação do controle de acesso, o gerente de objetos deve ser capaz de: identificar usuários e grupos de trabalho [EMM95]; e definir e modificar as responsabilidades sobre os modelos de domínio e seus componentes (quem criou, quem pode utilizar, que tipos de acesso são permitidos, etc.). Acreditamos que os controles de segurança utilizados pelos SGBDs em geral, são suficientes para os nossos propósitos, aliados a mecanismos de segurança definidos no contexto da própria IRBMD, para a utilização de ferramentas específicas.

2.5.4 Conjunto de serviços avançados

Os controles acima são os controles básicos, que acreditamos que um gerente de objetos para o suporte a uma IRBMD deva possuir. No entanto, existem outros controles como distribuição e integração aliada a interoperabilidade, que são desejáveis, senão essenciais em uma IRBMD, principalmente analisando o contexto do desenvolvimento de grandes projetos de software, onde características como distribuição de tarefas e interoperação de múltiplas plataformas são uma constante. Como estes controles, geralmente, não são descritos de forma detalhada na literatura, sendo ainda altamente dependentes do modelo de dados a ser utilizado, daremos uma maior ênfase aos mesmos.

- Distribuição

  Atualmente, o desenvolvimento de grandes projetos de software é distribuído entre diversos grupos de desenvolvedores que, geralmente, não estão alocados fisicamente em um mesmo lugar. Às vezes, estes grupos de desenvolvedores estão separados por grandes distâncias. Neste novo panorama do processo de desenvolvimento de software, os conceitos de distribuição no contexto da IRBMD assumem grande importância. Mecanismos para a distribuição dos modelos de domínio e ferramentas para a criação e utilização destes modelos, entre grupos de usuários, devem ser estudados, de forma a garantir eficiência, trabalho cooperativo e transparência.

Sendo assim, um dos requisitos desejáveis a uma infra-estrutura de reutilização, para dar suporte ao processo de desenvolvimento de software, é que esta permita o acesso distribuído e concorrente de múltiplos usuários aos dados armazenados. Na Tabela 2-1, apresentamos, de acordo com o tamanho do projeto e o tipo de arquitetura cliente/servidor, diferentes formas de se aplicar a distribuição aos modelos de domínio de uma infra-estrutura de reutilização.

No caso do servidor distribuído, um dos aspectos mais importantes para o funcionamento eficiente da infra-estrutura de reutilização, é a minimização de tráfego em rede e melhoria do desempenho das consultas realizadas à base, ou seja, como distribuir os componentes dos modelos de domínio, de forma a garantir um bom desempenho no acesso aos dados. Para isso, é necessário fazer um bom projeto de distribuição de objetos.

Tipo de Arquitetura Cliente/Servidor	Aplicação	Complexidade
Gerente de Dados totalmente no servidor	Indicada quando as ferramentas de acesso aos modelos de domínio não fazem grandes modificações e acessos constantes a diferentes tipos de dados.	Baixa
Gerente de Dados dividido entre cliente e servidor	Indicada quando as ferramentas são intensivas em acesso e modificação dos modelos de domínio e seus dados.	Média
Servidor Distribuído	Indicada para utilização em grandes projetos, quando os modelos de domínio podem ser distribuídos em mais de uma base de dados.	Alta

Tabela 2-1- Tipos de arquitetura cliente/servidor para modelos de domínio, de acordo com o tamanho do projeto.

Embora recente no contexto OO, já existem alguns estudos para o projeto de distribuição de objetos. Um dos estudos apresentados na literatura é a proposta apresentada por Baião [BAI97]. Baião leva em consideração, na sua proposta, aspectos importantes como quais classes possuem extensão, frequência de acessos e relacionamentos entre as classes, propondo, além disso, heurísticas para a resolução de conflitos na distribuição do modelo de classes/objetos entre as bases distribuídas. A proposta de Baião realiza ainda análises qualitativas e quantitativas em relação ao modelo de classes/objetos, o que garante estratégias de distribuição que priorizem a semântica do modelo. Portanto, no contexto de uma infra-estrutura de reutilização, onde a semântica dos modelos possui papel preponderante, a estratégia de distribuição de Baião apresenta-se promissora para a distribuição dos modelos.

- Integração e Interoperabilidade

  Podemos adotar duas abordagens principais em relação a utilização de ferramentas acopladas à base de objetos de uma infra-estrutura de reutilização. A primeira, diz respeito à integração total das ferramentas a uma base de objetos centralizada, tanto em termos de plataforma utilizada quando em termos do modelo de dados. Uma segunda abordagem, emprega o conceito de reutilização de ferramentas já prontas. Neste contexto, as ferramentas podem estar rodando em diversas plataformas e podem utilizar base de dados proprietárias ou não para o armazenamento dos modelos.

  Apesar da abordagem de base de dados centralizada permitir uma maior integração entre as ferramentas que atuam sobre a mesma (podendo inclusive utilizar a própria LMD para o acesso aos dados, de forma unificada), pode existir a necessidade de integração de diversos tipos de ferramentas heterogêneas. Uma base de objetos do domínio que venha atender a esta possibilidade de integração de diversos tipos de ferramentas, deve permitir que dados em formatos variados e distribuídos sejam utilizados. Portanto, a base de objetos do domínio deve permitir a integração de diversos meios de armazenamento que devem interoperar como uma única base de dados lógica, apresentando uma interface unificada para novas ferramentas que venham a atuar sobre a mesma. Portanto, para atender os requisitos acima, a base de objetos do domínio deve adotar uma arquitetura aberta em relação ao gerenciamento de dados [BAR95]. Para isso, a base de objetos do domínio deve ser uma combinação de diferentes meios de armazenamento, incluindo sistemas de arquivo, banco de dados comerciais, bases de conhecimento de sistemas especialistas, etc., ou seja, estes diversos meios de armazenamento devem interoperar como uma única base de objetos do domínio lógico, apresentando uma interface unificada para os outros componentes da infra-estrutura de reutilização.

Em [PIR97] é apresentada uma abordagem interessante para este tipo de necessidade, utilizando a tecnologia de mediadores[9] (*mediators*) [WIE97] e CORBA [OMG95], denominada HIMPAR (*Heterogeneous, Interoperable, Mediator, and Parallelism Architecture*). A arquitetura HIMPAR possibilita a interoperação e otimização de recursos em ambientes distribuídos e heterogêneos compostos de sistemas com diferentes funcionalidades. Outra característica de destaque da arquitetura HIMPAR é que seu modelo de dados é baseado no padrão ODMG [CAT94], o que facilita a utilização da arquitetura em outros projetos. Uma estrutura como a da arquitetura HIMPAR seria adequada às necessidades de uma infra-estrutura de reutilização com um conjunto de ferramentas heterogêneas em relação aos tipos de dados utilizados. No entanto, a abordagem do HIMPAR, conforme ressaltado pelos próprios autores, não caracteriza-se pelo tratamento das diferenças semânticas entre modelos de dados. No contexto de uma infra-estrutura de reutilização, o tratamento da semântica dos diferentes modelos é um requisito importante. Portanto, para utilizarmos a arquitetura HIMPAR como suporte a interoperação de diferentes bases de dados, devemos estendê-la a fim de abordar o tratamento da semântica das diversas bases de dados que podem compor a infra-estrutura, nos moldes dos projetos Garlic, Disco e TSIMNIS [PIR97].

[9] Um mediador é um componente de software que explora o conhecimento armazenado em um conjunto ou subconjunto de dados para gerar informações para aplicações residentes em uma camada superior.

2.6 Visão Global da Infra-estrutura de Reutilização

Na Figura 2-2 apresentamos uma visão global dos componentes da IRBMD. Os modelos de domínio constantes da infra-estrutura serão especificados e posteriormente evoluídos segundo as atividades a serem definidas no método de ED. Para isso, serão utilizadas as ferramentas para especificação e evolução do conhecimento do domínio, apresentadas na Figura 2-2. Dentre estas ferramentas, podemos citar como uma das mais importantes a ferramenta de elicitação dos requisitos, que será responsável por definir os conceitos iniciais do domínio e sua posterior evolução, em um alto nível de abstração.

A forma e representação dos modelos de domínio, detalhadas na seção 2.2, também serão definidas pelo método de ED, através do uso de ferramentas para especificação de padrões, gerenciamento de componentes reutilizáveis, ferramentas para definição dos modelos OO, entre outros. Ainda em relação aos modelos de domínio, cabe ressaltar que, na Figura 2-2, a arquitetura de software do domínio customizada para aplicação está representada no desenho por uma caixa pontilhada por ela não ser efetivamente armazenada na infra-estrutura de reutilização. Ela é criada com a ajuda do agente transformacional e depois é disponibilizada para o engenheiro de software para ser utilizada na especificação da sua aplicação. Esta arquitetura somente será armazenada de fato na infra-estrutura, se o engenheiro do domínio considerá-la como um exemplo adequado para ser armazenado no conjunto de arquiteturas OO de aplicações do domínio.

Os principais usuários da infra-estrutura serão o engenheiro do domínio, o especialista do domínio e o engenheiro de software responsável pelo desenvolvimento de aplicações no domínio. O engenheiro do domínio e o especialista utilizarão a infra-estrutura, principalmente para especificar e evoluir os conceitos do domínio. O engenheiro de software utilizará a infra-estrutura para obter conhecimento sobre o domínio da aplicação e reutilizar este conhecimento na especificação de sua aplicação. Por isso, os principais meios de interação do engenheiro de software com a infra-estrutura serão através do agente de informações, que permitirá que o engenheiro adquira conhecimento sobre o domínio em altos níveis de abstração, selecionando o conhecimento que achar relevante para a sua aplicação; e através do agente de transformação arquitetural, que permitirá a transformação dos conceitos iniciais do

domínio, selecionados através do agente de informação, para um modelo arquitetural. Estas ferramentas baseadas em agentes utilizarão os serviços do gerente de objetos para acessar os modelos de domínio armazenados na base de objetos do domínio. Com o uso deste acesso através do gerente de objetos, acreditamos que o desempenho da IRBMD será sensivelmente melhorado, uma vez que as ferramentas não precisarão se preocupar com questões de acesso concorrente, controle de redundância e consistência de informações, entre outros. Neste aspecto, a IRBMD, quando comparada a estruturas similares, como Sistemas Baseados em Conhecimento, possui um diferencial interessante.

Com a especificação desta infra-estrutura de reutilização baseada em modelos de domínio, pretendemos atingir os seguintes objetivos:

- a sistematização da transição dos modelos conceituais para o modelo arquitetural, através do desenvolvimento do agente de transformação arquitetural, que irá auxiliar nesta transição;
- a redução da distância cognitiva entre os conceitos iniciais de uma aplicação a ser desenvolvida até a sua codificação, uma vez que o engenheiro de software parte dos conceitos iniciais do domínio, através da interface hipermídia e os padrões de análise, até chegar ao framework de componentes da aplicação;
- utilização de conceitos OO em altos níveis de abstração, como é o caso de padrões, para a representação dos modelos, uma vez que os padrões permitem a representação da modelagem OO de forma mais amigável, permitindo capturar informações como o porquê do modelo ter sido especificado daquela forma, quais foram as circunstâncias que levaram àquela representação e em que contexto ele pode ser utilizado.
- uso de um gerente de objetos flexível, que realizará controles básicos de acesso aos modelos de domínio armazenados em sua base, garantindo a consistência dos modelos, livrando as ferramentas da IRBMD de tarefas relacionadas à manutenção destas consistências. Além disso, como o gerente de objetos disponibilizará apenas os serviços necessários à IRBMD, a estrutura da IRBMD como um todo não ficará sobrecarregada, como ficaria se, por exemplo, estivéssemos utilizando um SGBDOO completo.

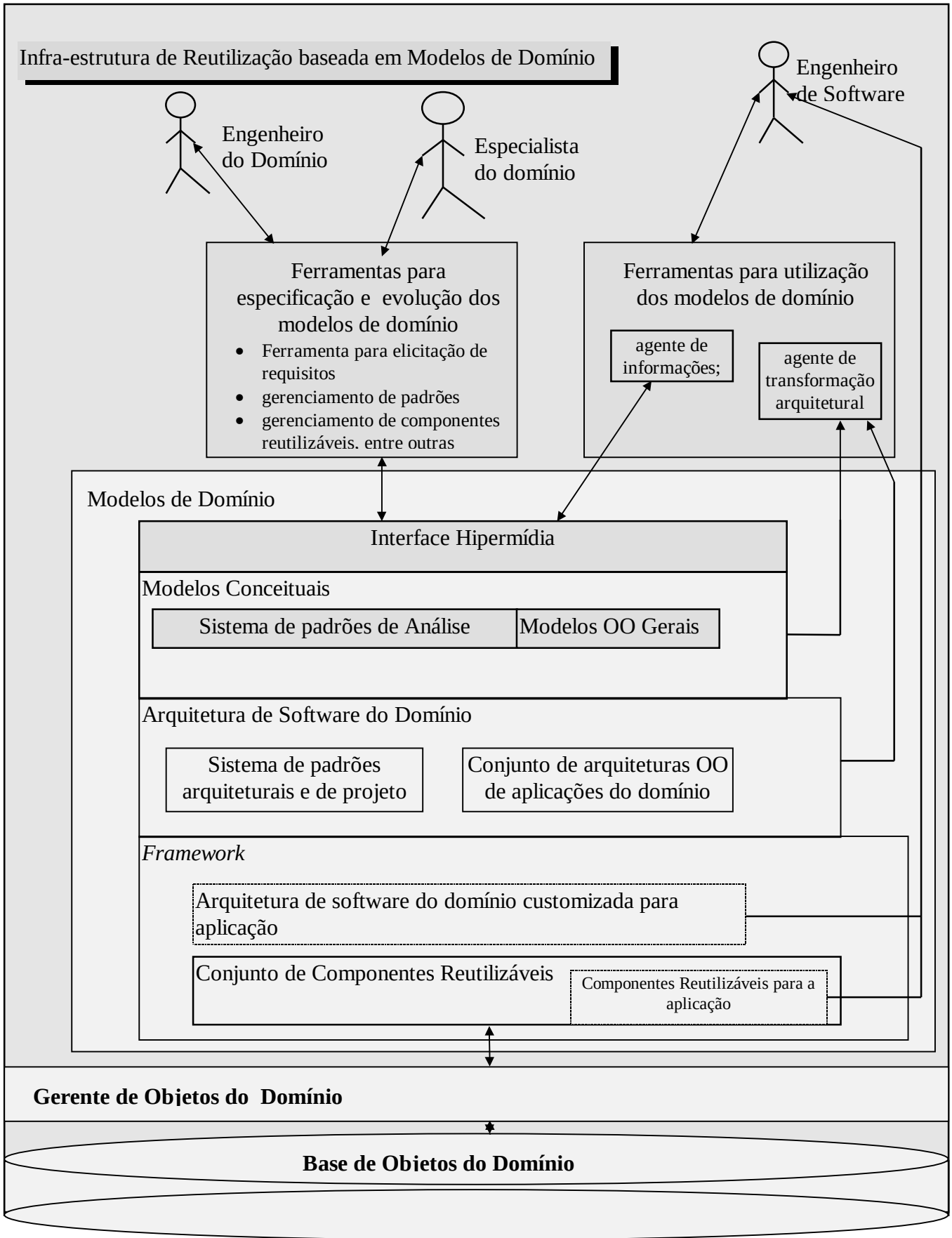

Figura 2-2- Visão diagramática dos componentes da infra-estrutura de reutilização

3. Considerações Finais

3.1 Principais contribuições da nossa proposta

Apresentamos, neste documento, uma proposta para o suporte ao desenvolvimento de aplicações baseado em componentes, através da especificação de uma infra-estrutura de reutilização baseada em modelos de domínio (IRBMD).

Conforme apresentamos no capítulo de introdução, segundo [BER97], temos atualmente três abordagens básicas para o desenvolvimento baseado em componentes: *componentware*, desenvolvimento baseado em conhecimento e desenvolvimento baseado em *frameworks*. O nosso trabalho poderia ser encaixado como uma composição das três abordagens, com especial ênfase no desenvolvimento baseado em *frameworks*, auxiliado por técnicas baseadas em conhecimento.

Acreditamos que a união destas três abordagens possa levar a resultados concretos para aumentar a reutilização no processo de desenvolvimento de software. As técnicas baseadas em conhecimento auxiliando na reutilização de conceitos do domínio em altos níveis de abstração, e o desenvolvimento baseado em frameworks como apoio à "concretização" do conhecimento embutido nos conceitos do domínio, através da especificação dos componentes reutilizáveis. As técnicas de *componentware* entrariam como apoio neste processo de reutilização, através dos serviços do SGBD encapsulados na utilização do protocolo CORBA no suporte à possível distribuição do conhecimento do domínio. A união destas abordagens, com certeza, trará benefícios para as pesquisas em reutilização.

Acreditamos que, com a especificação da infra-estrutura de reutilização, teremos algumas contribuições interessantes, principalmente nos seguintes pontos:

- contribuições na área da reutilização orientada a domínios específicos, que, segundo a literatura especializada [GAC95], [CZA97], [MAC97] terá um grande crescimento nos próximos anos;
- desenvolvimento de um método de engenharia de domínio que dê suporte a todas as fases do processo, incluindo ainda uma fase de análise de risco, com o objetivo de atestar a viabilidade de se aplicar a reutilização orientada a modelos naquele domínio;
- utilização sistemática de construções OO de mais alto nível, como padrões, e sua inserção em um método de ED;

- sistematização da transição entre modelos conceituais e arquiteturais.

Abordagens similares à nossa, apesar de apresentarem resultados interessantes, não especificam todos os aspectos de auxílio ao desenvolvimento baseado em componentes como a nossa, negligenciando um ou outro aspecto a ser definido pela IRBMD. Sendo assim, acreditamos que, com o auxílio da IRBMD, o desenvolvedor de software conseguirá aplicar a reutilização desde o início do processo de desenvolvimento da aplicação, ao passo que nas outras abordagens, geralmente, a ênfase é em uma ou outra fase do processo.

A fim de que possamos atingir concretamente estas contribuições, nossos estudos serão direcionados daqui para frente para um maior detalhamento das tecnologias envolvidas na especificação desta proposta e avaliação das mesmas através do desenvolvimento de protótipos. Acreditamos que, com isso, possamos construir uma infra-estrutura de reutilização baseada em modelos de domínio que atenda aos objetivos propostos na introdução deste documento, podendo assim ser validada em domínios cujas contribuições na sua utilização sejam efetivas.

Como contribuição final, devemos ressaltar que, com o uso desta infra-estrutura, esperamos que o desenvolvimento de software baseado em componentes possa ser sistematizado de forma adequada, aumentando a produtividade do desenvolvedor de software e reduzindo os custos do processo como um todo.

3.2 Análise de Viabilidade da proposta

Apresentamos ao longo deste documento as tecnologias envolvidas na especificação da infra-estrutura de reutilização, apresentando também como estas tecnologias podem ser compostas para formar esta infra-estrutura. Este estudo permitiu uma análise sobre a viabilidade da proposta, através da descrição das possibilidades de composição das tecnologias para a especificação da IRBMD.

Do ponto de vista prático de desenvolvimento da infra-estrutura, poderemos contar com resultados já obtidosna COPPE/Sistemas no uso das diversas tecnoloigas envolvidas na IRBMD. Em especial, no desenvolvimento baseado em frameworks [CIM96], uso de padrões para a especificação de conhecimento sobre processos de desenvolvimento de software [VAS97] e técnicas para aquisição de conhecimento no contexto da análise de domínio [ZOP97], várias soluções poderão ser adaptadas.

Além destes resultados prévios, também temos a disponibilidade de dois SGBDOOs no contexto dos projetos da COPPE/Sistemas: o O_2 [O_2 T95], representante nato de um SGBDOO, e o GOA++ [MAU97], que pode ser classificado como um gerenciador aberto de objetos. A disponibilidade destes produtos adiciona um componente a mais na viabilidade da proposta, uma vez que já temos disponível um conjunto de serviços básicos sobre os quais podemos trabalhar, customizando-os com o objetivo de atender às necessidades da IRBMD.

Referências Bibliográficas

[ALL96] Robert Allen and David Garlan,A Case Study in Architectural Modelling: The AEGIS System, Proceedings of the Eighth International Workshop on Software Specification and Design (IWSSD-8), Março 1996.

[ANG96] J. Angele, D. Fensel und R. Studer: Domain and Task Modeling in MIKE. In: A. Sutcliffe, D. Benyon, F. van Assche (Eds.): Domain Knowledge for Interactive System Design, Proceedings of IFIP 8.1/13.2 Joint Working Conference, Genebra, Maio 1996, Chapman & Hall, 1996.

[ASS96] Asset Home Pages - ASSET_A_1336 Organization Domain Modeling (ODM) Tutorial at http://www.asset.com/WSRD/indices/domains/ DOMAIN_ANALYSIS_AND_ENGINEERING.html

[BAI97] Fernanda Baião, A Mixed Fragmentation Approach to the Distributed Design of OO Databases, a ser publicado nos Anais do II Workshop on CSCW in Design, Bangkok, Tailândia, novembro 1997.

[BAR95] N. S. Barghouti, W. Emmerich, W. Schäfer and A. H. Skarra. Information Management in Process-Centered Software Engineering Environments. Technical Report No. 23 , ESPRIT-III Project GOODSTEP (6115), Janeiro 1995.

[BER97] Berg, Klaus - The Component Corner Component-Based Development: No Silver Bullet- Object Magazine, março, 1997

[BOH97] Kathy Bohrer- Middleware Isolates Business Logic, Object Magazine, novembro, 1997

[BOS97] Bosh, J. - Reusable Specification of Architectural Fragments - University of Karlskrona/Ronneby, Sweden,, 1997- at http://www.pt.hk-r.se/~bosh

[BRA95] Braga, R.; Mattoso, M.- Protótipo de um Sistema de Informação de Escritório Integrado-modelo e implementação, Tese de Mestrado-COPPE /UFRJ, 1995

[BRA96] Braga, R.M.M.; Travassos, G. H. - Arquiteturas de Software - Uma abordagem OO - Monografia do curso de Desenvolvimento Orientado a Objetos da COPPE/Sistemas, Agosto/96

[BRU97] Thaís Saldunbides Brugger, Cláudia Maria Lima Werner - Um estudo sobre os requisitos de SGBDs para Suporte a Ambientes de Desenvolvimento de Software Baseados em Reutilização, Relatório Técnico do Projeto Memphis, COPPE/UFRJ 5/97,1997

[BUS96] Frank Buschmann et al - Pattern-Oriented Software Architecture - A system of patterns - John Wiley Publising, 1996

[CAP97] Rafael Capilla - Application of Domain Analysis to Knowledge Reuse- WISR8, 1997 at http://www.umcs.maine.edu/~ftp/wisr/wisr8/papers.html

[CAR94] Estep, J.; Hissan, S.; Wallnau, K.; Martin, L. - Informal Technical Report - STARS, Air Force Material Command, Unisys, Reston Virginia, 1994

[CAT94] Catell, R.: Object Data Mangement , Addison-Wesley Publishing Company 1994

[CIM95] Cima, A. M. ; Werner, C. M. L. - A Escolha de um método de análise de domínio, Relatório Técnico, COPPE/UFRJ, 1995.

[CIM96] Alberto M. Cima - Desenvolvimento de Software com Reutilização Baseada em “Frameworks” Orientados a Objetos, Tese de Mestrado, COPPE/Sistemas, dezembro, 1996

[CLE96] Clements - Coming Attractions in Software Architecture - Technnical Report - CMU/SEI-96-TR-008, janeiro, 1996

[COH94] Cohen,S; ”Feature-Oriented Domain Analysis: Domain Modeling”, Tutorial Notes; 3rd Int. Conference on Software Reuse, Rio de Janeiro, Nov 1994.

[CZA97] Krzysztof Czarnecki - Leveraging Reuse Through Domain-Specific Software Architectures- WISR8, 1997 at http://www.umcs.maine.edu/~ftp/wisr/wisr8/papers.html

[DEV97] Developers Magazine, Axcel books, Agosto, 1997

[EMM95] W. Emmerich. Tool Construction for Process-Centered Software Development Environments based on Object Databases. PhD Thesis, Dept. of Mathematics and Computer Science, University of Paderborn, Germany, 1995

[EZE95] Ezeife, C.; Barker, K. - A comprehensive approach to horizontal class fragmentation in distributed object base system - Distributed and Parallel Databases, Kluwer Academic Publishers, 1995

[FAY97] Mohamed Fayad, Douglas Schmidt, Object-Oriented Application Frameworks, Communications of the ACM, Outubro 1997

[FIS95b] G. Fischer, K. Nakakoji and J. Ostwald, " Supporting the Evolution of Design Artifacts with Representations of Context and Intent," In proceedings of Designing Interactive Systems (DIS '95), Ann Arbor, MI, 1995, pp. 7-15.

[FIS96] Gerhard Fischer - "Seeding, Evolutionary Growth and Reseeding: Constructing, Capturing and Evolving Knowledge in Domain-Oriented Design Environments", Proceedings of IFIP WG 8.1/13.2 Joint

Working Conference, A. Sutcliffe, D. Benyon and F. van Assche (eds): "Domain Knowledge for Interactive System Design", IFIP Series, Chapman & Hall, London, Geneva, Switzerland, Maio 1996, pp 1-16

[FOW97] Martin Fowler - Analysis Patterns - Reusable Object Models - Addison Wesley Publications, 1997

[FOW97a] Martin Fowler - UML Distilled: Applying Teh Standard Object Modeling Language, Addison-Wesley, 1997

[FRA97] Bill Frakes - Automating Domain Engineering- WISR8, 1997 at http://www.umcs.maine.edu/~ftp/wisr/wisr8/papers.html

[GAC95] Gacek, C., A. Abd-Allah, B.K. Clark, and B.W. Boehm, "On the Definition of Software System Architecture", in Proceedings of the First International Workshop on Architectures for Software Systems, Seattle, Wa., 24-25 April 1995, pp.85-95.

[GAM94] Erich Gamma et al - Design *Patterns*: Reuse of Object Oriented Design Addison Wesley Publishing, 1994

[GAR93] D.Garlan, M.Shaw, "An Introduction to Software Architecture." In V. Ambriola and G. Tortora (ed.), Advances in Software Engineering and Knowledge Engineering , Series on Software Engineering and Knowledge Engineering, Vol 2, World Scientific Publishing Company 1993, pp. 1-39.

[GAR96] David Garlan, Andrew Kompanek, Ralph Melton, and Robert Monroe , Architectural Style: An Object-Oriented Approach, David Garlan, Andrew Kompanek, Ralph Melton, and Robert Monroe Submitted for publication , Fevereiro 1996.

[GOM96] Gomaa, H et al - A Knowledge-Based Software Engineering Environment for Reusable Software Requirements and Architectures - Automated software Engineering 3, 285-307, 1996

[GRU92] T. Gruber, J. Tenenbaum - Toward a Knowledge Medium for Collaborative Product Development in Artificial Inteligence in Design’92, junho, 1992

[HAR98] Frank van Harmelen, Annette ten Teije - Characterising Problem Solving Methods by gradual requirements:overcoming the yes/no distinction to appear in Proceedings of the Eleventh Workshop on Knowledge Acquisition for Knowledge-Based Systems (KAW' 98), Banff, Alberta, April 1998.

[HAY94] F. Hayes-Roth, N. Jacobstein - The state of Knowledge Based Systems, CACM, v. 37 n3, 1994

[HAY94] Frederick Hayes-Roth - Architecture-Based Acquisition and Development of Software: Guidelines and Recommendations from the

ARPA DSSA Program, Teknowledge Technical Document, fevereiro, 1994

[HEN95] S. Henninger, "Developing Domain Knowledge Through the Reuse of Project Experiences," Symposium on Software Reusability (SSR ' 95), Seattle WA, April 1995.

[HEN96] S. Henninger, "Case-Based Knowledge Management Tools for Software Development," Automated Software Engineering: An International Journal, 4(1), 1997.

[HEN97] S. Henninger, "An Evolutionary Approach to Constructing Effective Software Reuse Repositories," ACM Transactions on Software Engineering Methodology, abril, 1997

[JAC94] Ivar Jacobson, Object-Oriented Software Engineering: A use case driven approach, Addison-Wesley, 1994

[JOH91] Johnson, R; Foote, B. - Designing Reusable Classes, reimpresso do Journal of object oriented programming, 1991

[JOH93] Johnson, R; - "Designing Frameworks" - Notes from OOPSLA'93

[KAZ95] Rick Kazman, Gregory Abowd, Len Bass, and Paul Clements. "Scenario-based Analysis of Software Architecture," (Accepted for publication by IEEE in 1996) em http://www.stars.reston.unisysgsg.com/arch/guide.html

[KIE95] N. Kiesel, A. Schuerr, and B. Westfechtel "GRAS, A Graph-Oriented (Software) Engineering Database System" Information Systems , 20(1):21-52, 1995.

[KLI95] Klingler, Carol D., and Schwarting, Dan, "A Practical Approach to Process Definition," Proceedings of the Seventh Annual Software Technology Conference, Salt Lake City, Utah, April 1995

[KRU92] C. Krueger, Software Reuse, ACM Computing Surveys, vol. 24, no 2, junho 1992, pp. 131-183

[LAR97] Deborah Larorme, Maria E. Stropky - AN AUTOMATED MECHANISM FOR EFFECTIVELY APPLYING DOMAIN ENGINEERING IN REUSE ACTIVITIES - Army Reuse Center White Papers at http://arc_www.belvoir.army.mil/htmldocs/arc/da_papers/applying_domain_engineering.htm

[LIS97] Maria Lúcia Blank Lisbôa - Arquiteturas de Meta-nível - Tutorial Notes em XI SBES, Fortaleza, 1997

[LOW95] Lowry, M., Philpot, A., Pressburger, T., Underwood, I., "A Formal Approach to Domain-Oriented Software Design Environments", in

Proc. 9th Knowledge-Based Software Engineering Conference, Monterey, CA, setembro. 20-23, 1994, pp. 2, 48-57.

[LOW97] Lowry, M., Van Baalen J., "Meta-Amphion: Synthesis of Efficient Domain-Specific Program Synthesis Systems", Automated Software Engineering, 4, 1997, pp. 199-241

[LUK97] Sean Luke et al - Ontology-based Web Agents - Proceedings of First International conference on Autonomous Agents, 1997

[MAC97] Pia Maria Maccario - The Domain Analysis Integrated in an Object Oriented Development Methodology- WISR8, 1997 at http://www.umcs.maine.edu/~ftp/wisr/wisr8/papers.html

[MAU97] Mauro R C, Silva G Z, Brügger T S, Tavares F O, Duran M, Lima A A B, Pires P F, Bezerra E, Soares J A, Baião F A, Mattoso M L Q, Xexéo G - GOA++: Tecnologia, Implementação e Extensões aos Serviços de Gerência de Objetos - Anais do SBBD, Fortaleza, Outubro de 1997

[MIG95] Might, R. - Domain Models What are They? How they are used? - George Mason University, 1996

[NAK96] K. Nakakoji, B.N. Reeves, A. Aoki, eMMa: An Environment for Designing "Good" Multimedia Presentations," Proceedings of the First Asia Pacific Conference on Computer Human Interaction, pp. 397-409, Information Technology Institute, Singapore, Junho, 1996.

[ODM97] The Object Database Standard: ODMG 2.0 Edited by R.G.G. Cattell, Douglas, Barry, Dirk Bartels, Mark Berler, Jeff, Eastman, Sophie Gamerman, David Jordan, Adam Springer, Henry Strickland, and Wade The Morgan Kaufmann Series in Data Management Systems, Jim Gray Series Editor 1997

[OLIV95] Oliveira, K. - Ambientes de Desenvolvimento Orientados a Domínio, Relatório Técnico de Reutilização de Software, 1995

[OMG95] Object Management Group, “The Common Object Request Broker Architecture and Specification”, Revisão 2.0, julho 1995.

[OTT97] Tim Ottinger, Robert Martin - What is wrong with OO?, Object Magazine, novembro, 1997

[PIR97] Paulo F. Pires - Himpar, uma arquitetura para interoperabilidade de objetos distribuídos, Tese de Mestrado , COPPE/UFRJ, 1997

[PRE97] Roger S. Pressman, Software Engineering - A practioner’s approach, McGraw-Hill Editions, 1997

[PRI87] Prieto-Diaz; Freeman - Classifying software for reusability, IEEE software, Janeiro, 1987

[QSOR95] SORT - Software Optimization and Reuse Technology at http://sort.ivv.nasa.gov/intro.htm, 1995

[RUM91] James Rumbaugh et at - Object-Oriented Modeling and Design. Prentice Hall, 1991

[RUS95] Stuart Russell, Peter Norvig , Artificial Intelligence: A Modern Approach, Prentice Hall, 1995

[SEI96] SEI - Domain Engeneering home pages- http://www.sei.cmu.edu/techonology/architecture

[SHA93] D.Garlan, M.Shaw, "An Introduction to Software Architecture." In V. Ambriola and G. Tortora (ed.), Advances in Software Engineering and Knowledge Engineering , Series on Software Engineering and Knowledge Engineering, Vol 2, World Scientific Publishing Company 1993, pp. 1-39.

[SHA95a] M.Shaw, R.DeLine, D.Klein, T.Ross, D.Young, G.Zelesnik, "Abstractions for Software Architecture and Tools to Support Them" IEEE Transactions on Software Engineering, April 1995.

[SHA95b] M.Shaw, "Beyond Objects: A Software Design Paradigm Based on Process Control," ACM Software Eng. Notes, Vol.20 No.1, Janeiro 1995.

[SHA96] M.Shaw, "Some *Patterns* for Software Architecture." Second Annual Conference on Pattern Languages of Programming, Setembro 1995. In Pattern Languages of Program Design, vol 2, John Vlissides, James Copllien, Norman Kerth (ed), Addison-Wesley 1996, pp. 255-269.

[SHU96] Xin Shu- Fitting Design Patterns into Object-Oriented Methods, Doctoral Thesis, University of Illinois, Chicago, 1996

[SIL97] A. Silberschatz, S. Zdonik - Database Systems - Breaking Out teh Box - SIGMOD Record, Vol.26, no. 3, setembro, 1997

[SOL96] James Solderitsch , - BRIDGING THE GAP BETWEEN DOMAIN MODELING AND DOMAIN ARCHITECTURE DEFINITION, LOCKHEED MARTIN TACTICAL DEFENSE SYSTEMS at http://www.asset.com/stars, 1996

[SUM96] The High-Tech Toolbelt: A Study of Designers in the Workplace T. Sumner, Human Factors in Computing Systems (CHI `95), Denver, CO (Maio 7-11), 1995, pp. 178-185.

[TAV96] Soares, J. A. Tavares, F.O., "PARGOA2: Análise de Desempenho no Paralelismo Virtual com o Servidor de Objetos GOA", Anais da XVII Jornada Interna de Iniciação Científica, UFRJ, novembro 1995.

[TRA94] Tracz, W. - Domain-Specific Software Architecture Frequently Asked Questions- ACM Software Engineering Notes, Vol 9, no 2, Abril, 1994, pp. 52-56

[TSI82] D. Tsichritzis, F. Lochovsky - Data Models - Prentice Hall Software Series, 1982.

[VAS95] F. M. de Vasconcelos Jr., M. F. da Silva - Técnicas de Representação do Conhecimento Aplicáveis à Reutilização, Relatório Técnico TABA RT-14/95, COPPE/UFRJ, 1995

[VAS97] F. Vasconcelos - Reutilização de Processos Baseada em Padrões, Tese de Mestrado, COPPE/UFRJ, 1997

[VES93] Vestal, S. - A cursory overview and comparison of four architecture description language - Honeywell system, 1993

[VIL97] K. Villela - Requisitos de um Ambiente de Desenvolvimento de Software para o Domínio de Cardiologia - Monografia do Curso de Tópicos Especiais em Engenharia de Software I, COPPE/Sistemas, 1997

[WIE97] Gio Wiederhold, INTEGRATION OF INFORMATION FROM HETEROGENEOUS SOURCES, Invited Talk at XII SBBD, Fortaleza, 1997

[ZOP97] M. Zopellari - Técnicas para aquisição de conhecimento no contexto da análise de domínio, Tese de Mestrado (em andamento), 1997

ANEXO - Conceitos Básicos relevantes à proposta

3.3 Introdução

Existem diversos tipos de tecnologias envolvidas na especificação de uma infra-estrutura de reutilização. Seguindo o modelo de Tsichritzis [TSI82], apresentamos neste anexo as principais tecnologias envolvidas na especificação de um modelo para uma infra-estrutura de reutilização, ou seja, apresentamos as tecnologias envolvidas na especificação da **parte estrutural** do modelo, através da apresentação dos principais conceitos relativos a modelos de domínio; das **operações** do modelo, através da descrição do processo de engenharia de domínio; e das **regras de integridade**, através da descrição dos principais conceitos envolvidos na especificação de um método de engenharia de domínio.

As tecnologias envolvidas na disponibilização de serviços básicos que facilitem a implementação do modelo da infra-estrutura, são tecnologias já largamente discutidas na literatura especializada. Dentre estas tecnologias podemos destacar: mecanismos para controle de concorrência, evolução de esquema, linguagens de definição e manipulação de dados, entre outras. Para o leitor não familiarizado com estas tecnologias, temos a disposição um estudo prévio [BRA95], realizado em um contexto correlato, que pode auxiliar em um maior entendimento sobre estes serviços básicos. No entanto, devemos ressaltar que, na seção 2.5 apresentamos, de forma específica, uma descrição do uso de tecnologias para a disponibilização de serviços de gerência de dados para a IRBMD.

3.4 Estrutura: Modelos de Domínio

Para que uma infra-estrutura de reutilização seja especificada e utilizada de maneira consistente, ela deve ser povoada com modelos de domínio que capturem de forma adequada a semântica do domínio, nos diversos níveis de abstração requeridos.

Estes modelos de domínio devem ser especificados de forma sistemática, utilizando para isso um processo consistente de ED, que leve em consideração não apenas a necessidade de componentes reutilizáveis, mas o conhecimento do domínio como um todo, nos diversos níveis de abstração. O conhecimento do domínio, principalmente em altos níveis de abstração, é importante para facilitar o processo de reutilização, pois ajuda a guiar o usuário na utilização dos componentes codificados.

Neste contexto, apresentamos três tipos de modelos de domínio, onde cada tipo representa um nível de abstração específico, de acordo com a fase do processo de ED, ou seja, na fase de análise de domínio, teremos como resultado a especificação de **modelos conceituais**, que modelem os principais conceitos do domínio, em um nível de abstração alto; no projeto do domínio, o resultado será uma **arquitetura de software do domínio**, que apresente como os diversos conceitos do domínio podem ser mapeados para os componentes reutilizáveis e como estes componentes relacionam-se entre si; na implementação do domínio serão especificados e/ou adaptados componentes préexistentes, a fim de formar um **modelo implementacional**, disponibilizado através de um *framework*, que represente os componentes reutilizáveis do domínio e seus relacionamentos, codificados em uma linguagem de programação.

3.4.1 Modelos Conceituais

Um modelo conceitual de domínio, segundo Czarnecki [CZA97], é uma forma de se definir os conceitos e princípios de um domínio, estabelecendo um vocabulário básico do domínio e uma taxonomia das abstrações do mesmo. Might [MIG95] destaca que um modelo conceitual não tem como objetivo a associação de funções à componentes, nem a especificação de como as interfaces dos componentes se comunicam. O modelo conceitual deve especificar apenas que as interfaces existem e o que flui entre estas interfaces.

No contexto de uma infra-estrutura de reutilização, onde o modelo conceitual será utilizado na especificação da arquitetura de software do domínio, devemos ressaltar também a importância da transição entre estes modelos. Em [SOL96] este problema é abordado. Segundo os autores, uma das formas para amenizar a dificuldade na transição entre os modelos, é, na especificação dos modelos conceituais, já levar em consideração aspectos arquiteturais.

Consultando a literatura especializada, podemos notar que os modelos conceituais são classificados em dois tipos principais:

- modelos conceituais formais, que são modelos construídos sobre estruturas bem definidas, baseadas em teorias formais como fórmulas matemáticas, regras de lógica, entre outras. Geralmente, estes modelos permitem a realização de análises formais sobre seus componentes, de forma a garantir sua corretude, sendo esta sua principal vantagem. Como desvantagem principal temos a dificuldade de

entendimento gerada por este tipo de modelo, pois geralmente suas estruturas são de difícil entendimento.

- modelos informais ou semi-formais: sua principal característica é a facilidade de entendimento. Não possuem facilidades para análises formais, o que dificulta a checagem da sua corretude. Exemplos deste tipo de modelos são: sistemas hipermídia, descrições textuais, modelos de classes/objetos, entre outros. O modelo de objetos pode ser classificado como semi-formal, pois permite de certa forma a checagem da corretude, através da análise da consistência entre os diferentes modelos. Geralmente utilizam uma notação gráfica para sua representação.

Na prática, o que podemos observar é que os pesquisadores em modelagem conceitual, utilizam uma mesclagem de representações para a descrição de seus modelos, aliando a facilidade de entendimento dos modelos informais com a corretude dos modelos formais. As representações mais utilizadas são:

- sistemas hipermídia acoplados a uma base de regras, como o projeto MIKE [ANG96]. Geralmente, este tipo de construção é utilizada para a elicitação do conhecimento do domínio;
- modelos de classes/objetos e/ou modelos E-R, para a representação dos elementos do domínio, facilitando a transição posterior para a arquitetura de software do domínio. Exemplo de utilização deste tipo de representação é o projeto BORE [HEN96], que utiliza representações multimídia conjuntamente com diagramas OO;
- frames e regras, para a representação do conhecimento formal. Este tipo de formalismo, geralmente, é utilizado em sistemas baseados em conhecimento. O ambiente KBSEE [GOM96] utiliza uma base de regras para a representação de um dos seus modelos. No entanto, o KBSEE utiliza também outras representações, como, por exemplo, representações OO. A transição entre estas representações se dá através do uso de ferramentas especificadas no contexto do KBSEE.

Analisando estes trabalhos e as tecnologias existentes, no contexto de uma infra-estrutura de reutilização, é preciso utilizar os dois tipos de representação: a informal, para facilitar o entendimento do usuário que está utilizando a infra-estrutura; e o formal para auxiliar na transição para modelos de projeto e garantir a corretude.

3.4.2 Arquiteturas de software do domínio

Não existe um consenso entre os pesquisadores da área em relação a melhor terminologia para a especificação das estruturas da arquitetura e nem mesmo a melhor definição do que seria uma arquitetura de software. Em [CLE96], é apresentada uma definição de arquitetura de software que tenta abranger todos os aspectos tratados por definições anteriores [SHA93] [GAC95]: "Arquitetura de software é a estrutura dos componentes do sistema, seus inter-relacionamentos, princípios e diretrizes que governam seu projeto e evolução ao longo do tempo". A estrutura de que trata esta definição e as maneiras de representá-la é que variará de acordo com o tipo de sistema a ser desenvolvido (**estilo**), o interesse dos usuários da arquitetura (**visões**) e algumas vezes o domínio para o qual foi desenvolvida (**arquiteturas de software do domínio**).

Apesar das pesquisas em relação à arquiteturas de software estarem ganhando cada vez mais adeptos, a maioria das arquiteturas de aplicações são descritas de maneira informal, sem qualquer compromisso com fundamentações teóricas, com estilos de arquitetura, notações ou terminologias, entre outros aspectos. Esta informalidade leva, segundo [ALL96], a uma representação ambígua, difícil de ser analisada e automatizada. Além disso, estas descrições não possuem o suporte adequado de ferramentas para sua especificação.

No entanto, observando as similaridades entre estas descrições arquiteturais informais, Shaw e Garlan [SHA93], [SHA95a] [SHA95b] identificaram regularidades nestas descrições, gerando assim um conjunto de **estilos arquiteturais**. Um estilo de arquitetura define, portanto, uma família de sistemas em termos de seus padrões de organização estrutural [SHA93]. Baseando-se na noção de arquiteturas de software como sendo um conjunto de restrições a como componentes e conectores podem ser combinados, Garlan e Shaw identificaram um conjunto básico de estilos que podem ser combinados e/ou especializados para originar novos estilos. Cada um dos estilos identificados por Shaw e Garlan são apropriados para alguma classe de problemas, mas nenhum dos estilos é adequado para todos os problemas [SHA96].

Além da especificação de estilos arquiteturais, Clements [CLE96] ressalta que, para garantirmos a adequabilidade de uma arquitetura para um domínio de aplicação, devemos utilizar o processo de análise de domínio, que ajudará a elucidar questões arquiteturais tais como: Qual(is) estrutura(s) (estilo) devo adotar para a arquitetura? Quais são os elementos arquiteturais para este domínio? Que restrições e regras

devemos adotar para a estrutura em geral e para os elementos em particular? Que tipo de visões arquiteturais são importantes no domínio de aplicação?

A partir dos resultados da análise de domínio e dos estilos arquiteturais, podemos então definir a arquitetura de software adequada para o domínio.

Might [MIG95] apresenta de maneira interessante o processo de criação de uma arquitetura de software específica do domínio (*DSSA - Domain Specific Software Architecture*). Para Might, uma arquitetura de software do domínio é uma extensão do modelo conceitual, descrevendo como o modelo conceitual pode ser solucionado, incluindo as funções e suas interfaces. O pesquisador ressalta ainda que a arquitetura existe em diferentes níveis de abstração: **a arquitetura genérica** (estilo(s) de arquitetura mais adequado(s) para o domínio em questão) e **a arquitetura específica do domínio**, que é a adequação da arquitetura genérica aos requisitos, restrições e regras do domínio.

A arquitetura genérica é representada pela escolha do(s) estilo(s) arquitetural(is) [GAR93] mais adequado(s) para o domínio da aplicação. Diferentes tipos de problemas nos levam para diferentes estilos arquiteturais. O conhecimento adquirido com a análise do domínio, através dos modelos conceituais, será utilizado tanto para auxiliar na escolha do estilo que mais se adequa ao domínio quanto para especializar o estilo escolhido, para melhor se adaptar aos elementos do domínio e também para especificar regras e restrições adequadas ao domínio.

No entanto, para podermos reutilizar efetivamente a arquitetura de software do domínio, devemos ter meios de representá-la. Atualmente, existem três abordagens principais [GAR96] para representação de arquiteturas de software:

- A primeira é usar linguagens de programação tradicionais, com algumas computações adicionais. Nesta abordagem, as arquiteturas são codificadas utilizando os termos providos pela linguagem de programação. Temos como principal vantagem, o uso de uma linguagem já familiar para os desenvolvedores do sistema. No entanto, tem a desvantagem de limitar a representação dos elementos arquiteturais a módulos e chamadas de procedimentos. Atualmente, a abordagem conhecida como Computação Reflexiva [LIS97] está sendo utilizada para a especificação das chamadas “arquiteturas de meta-nível”;
- A segunda é utilizar notações que tenham sido especialmente desenvolvidas para representação arquitetural. Estas notações estão especificadas nas chamadas ADLs

(*Architecture Description Languages*) [VES93]. Esta abordagem possui melhor suporte para análises arquiteturais e melhor representação das intenções do arquiteto através de representações do sistema.

- A terceira é utilizar construções em alto nível de abstração como padrões arquiteturais [BUS96], de análise [FOW97] e de projeto [GAM94].

3.4.3 - Frameworks

Um *framework* é um conjunto de classes que incorpora um projeto abstrato de soluções para um conjunto de problemas relacionados, suportando a reutilização em uma granularidade maior que a de classes [JOH91]. O conjunto de classes é representado por um conjunto de componentes reutilizáveis, codificados em alguma linguagem de programação, sejam estes componentes do tipo classes abstratas ou concretas; e a forma como estas classes estão relacionadas é representada pela arquitetura de software do domínio.

Os principais objetivos de um *framework* são: construir aplicações utilizando componentes preexistentes, utilizar estes componentes extensivamente e escrever menos código possível [JOH93]. A utilização de *framework*s é, portanto, um tópico importante. Temos duas maneiras de utilizar um *framework*:

- definindo novas classes necessárias, através da extensão das hierarquias de classes do *framework*;
- configuração de um conjunto de objetos existentes através de parâmetros e ligações.

A primeira forma de utilização de um *framework* é mais complexa, necessitando que o desenvolvedor tenha conhecimento a respeito da estrutura interna do mesmo. *Framework*s que são utilizados desta maneira são chamados *frameworks* "caixa branca" [JOH91]. O problema em se utilizar *framework*s "caixa branca" é que cada aplicação requer a criação de várias novas subclasses e o desenvolvedor tem que aprender como foi a sua implementação. A segunda forma de se utilizar o *framework* caracteriza-se pela simplicidade no uso. O desenvolvedor necessita apenas de entender a interface externa dos componentes. Este tipo de *framework* é denominado "caixa preta".

A escolha do tipo de *framework* a utilizar, isto é, *framework* "caixa branca" ou "caixa preta", depende do domínio da aplicação. Existem domínios de aplicação já

suficientemente maduros onde é possível utilizar *framework*s "caixa preta". No entanto, existem domínios de aplicação, onde as necessidades das aplicações estão em constante evolução, requerendo constantes mudanças e especializações dos componentes do *framework*. Para este tipo de aplicação, *framework*s "caixa branca" são os mais adequados.

No contexto de uma infra-estrutura de reutilização, o *framework* cumpre um papel importante, pois é a partir dele que se estabelece os componentes reutilizáveis que serão utilizados no desenvolvimento de novas aplicações do domínio. Neste contexto, o framework direciona como seus componentes colaboram para a especificação de uma aplicação do domínio, podendo englobar conceitos tanto de *framework* "caixa-branca" quanto de *framework* "caixa preta", dependendo do domínio da aplicação.

Apesar do *framework* estar classificado no contexto da nossa infra-estrutura de reutilização, como sendo um modelo implementacional, na verdade ele pode ser classificado também como modelo arquitetural, pois utiliza a arquitetura de software do domínio para a representação dos relacionamentos entre seus componentes. Sendo assim, o *framework* engloba a arquitetura de software do domínio, garantindo que as aplicações especificadas utilizando os componentes reutilizáveis do *framework* serão regidas por esta arquitetura, o que garantirá a conformidade da aplicação com as regras gerais do domínio, assegurando a sua corretude e permitindo a sua evolução e manutenção de forma facilitada.

3.5 Operações: a Engenharia do Domínio

A engenharia de domínio (ED) é importante no contexto de uma infra-estrutura de reutilização por ser ela quem irá organizar os passos necessários para a criação e evolução dos modelos a serem utilizados no processo de especificação de aplicações do domínio, ou seja, quais serão as **operações** que atuarão sobre os modelos de domínio.

Analisando a literatura especializada, podemos comprovar que a maioria das abordagens que pregam a adoção de uma infra-estrutura de reutilização são criadas de maneira "ad-hoc", ou seja, sem um processo de ED definido, nem contando com o auxílio de modelos de domínio para guiar o processo de reutilização de componentes, dificultando assim a identificação de oportunidades de reutilização. Na abordagem

utilizada neste trabalho, a adoção de um processo de ED sistemático é premissa básica para a especificação de uma infra-estrutura de reutilização. Acreditamos que, sem esta sistematização, não existem garantias de que a infra-estrutura de reutilização seja realmente adequada para prover modelos que irão auxiliar na especificação de aplicações no domínio. Por isso, a importância de estudarmos adequadamente o processo de ED e métodos para a sua sistematização.

Segundo a *Army Reuse Center* [LAR97], a ED é o processo de busca, análise, manipulação e formalização das informações do domínio relevantes para a implementação de um programa de reutilização, que promova desenvolvimento de aplicações do domínio a um custo reduzido.

A ED tem como principal objetivo a utilização de uma abordagem baseada em modelos de domínio (conceituais, arquiteturais e implementacionais) para facilitar a instituição da reutilização e sua utilização em cadeia no ciclo de desenvolvimento de software. Além disso, a ED deve permitir a criação e posterior evolução dos modelos criados, devendo, portanto, ser um processo intuitivo, interativo e iterativo. Neste sentido, as informações do domínio coletadas durante o processo devem ser organizadas, manipuladas e validadas pelos engenheiros do domínio durante todo o processo, para assegurar o sucesso com resultados confiáveis.

Existe um consenso entre os pesquisadores da área [SEI96], [TRA94], [LAR97], [FRA97] que a ED é composta basicamente das seguintes fases:

1. Análise do domínio, que determina os requisitos comuns em um domínio com o objetivo de identificar as oportunidades de reutilização.
2. Projeto do domínio: Utiliza os resultados da análise do domínio para identificar e generalizar soluções para os requisitos comuns através da especificação de uma arquitetura de software do domínio. As oportunidades de reutilização identificadas na análise do domínio são refinadas de forma a especificar as restrições do projeto.
3. Implementação do domínio: Transforma as oportunidades de reutilização e soluções do projeto para um modelo implementacional, que inclui serviços tais como: a identificação, reengenharia e/ou construção, e manutenção de componentes reutilizáveis que suportem estes requisitos e soluções do projeto.

Os produtos gerados nestas fases são os modelos conceituais do domínio (ou modelos do domínio do problema), modelos do projeto (ou arquiteturas de software

do domínio) e modelos implementacionais (ou *frameworks*[10]) [SEI96]. A Figura A.1 apresenta as principais etapas e os produtos gerados por elas.

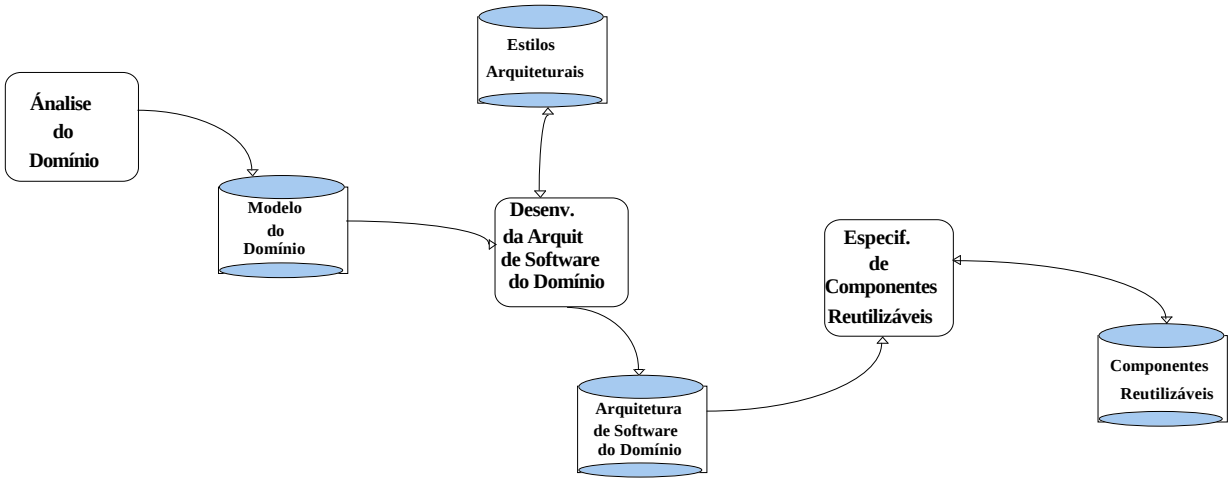

Figura A.1- Principais Etapas da Engenharia do Domínio

Analisando as principais fases da ED, podemos concluir que os resultados das três fases principais podem ser utilizados na criação de uma infra-estrutura de reutilização, através da especificação dos diversos modelos de domínio, sistematizando assim a criação dos modelos conceituais, arquitetura de software do domínio e (*framework* de) componentes reutilizáveis, os quais guiarão o desenvolvimento de aplicações no domínio.

3.6 Regras de integridade: Métodos para a sistematização da Engenharia de Domínio

Conforme já ressaltado, a sistematização da ED, em passos bem definidos e com resultados que possam ser reutilizados concretamente no desenvolvimento de aplicações em universos de domínios correlatos, é uma das principais necessidades para o desenvolvimento de uma infra-estrutura de reutilização cujos modelos de domínio serão utilizados para ajudar na especificação de aplicações.

Além disso, a ED geralmente só é viável economicamente quando é aplicada em um domínio cujo nível de complexidade e número de aplicações é grande e, por isso, os custos de se desenvolver uma infra-estrutura de reutilização para o domínio, considerando estruturas de alto nível como arquiteturas de software e modelos

[10] No contexto deste trabalho, descrevemos *frameworks* como sendo um conjunto de componentes reutilizáveis, cujos relacionamentos são regidos pela arquitetura de software do domínio.

conceituais, são viáveis. Sem a sistematização do processo de ED, corremos o risco de especificar uma infra-estrutura de reutilização que não contemple todos os conceitos do domínio, que não permita a transformação entre os diferentes níveis de abstração dos modelos e que, ao final, não produza componentes reutilizáveis e/ou aplicações concretas.

Na literatura existem diversos métodos para sistematizar a ED, cada um enfatizando uma ou mais fases do processo. A maioria deles enfatiza a fase inicial do processo, ou seja, a análise de domínio, como é o caso do FODA e ODM, entre outros. Outros, como o KAPTUR, enfatizam a fase de projeto do domínio.

Em [CAR94], são destacados alguns pontos básicos que devem ser analisados na escolha de um método de ED. São eles:

- permitir delimitar claramente as fronteiras do domínio;
- possuir mecanismos que permitam ressaltar as similaridades e diferenças entre as aplicações do domínio;
- permitir o entendimento claro dos relacionamentos entre os vários elementos do domínio;
- permitir a representação do conhecimento adquirido de maneira clara e que facilite o entendimento do usuário.

Analisando estes pontos básicos, concordamos com Cima [CIM95] que enfatiza que a escolha de um método de análise de domínio depende muito do contexto onde este será aplicado. Ressaltamos ainda que esta escolha depende, principalmente, da cultura de desenvolvimento da empresa que irá adotá-lo. Por exemplo, em um contexto de desenvolvimento baseado em orientação a objetos, devemos enfatizar a escolha de um método de ED que produza modelos que sejam facilmente utilizados em um processo de desenvolvimento OO.

Apresentamos a seguir uma tabela comparativa (Tabela A.1) entre os métodos de ED que mais se destacam no mercado. Esta tabela é baseada nos pontos básicos que devem ser analisados na escolha de um método de ED, descritos em [CAR94].

A partir da tabela 1.1 podemos fazer as seguintes considerações em relação aos métodos de ED apresentados:

- a maioria dos métodos enfoca uma ou outra fase da engenharia do domínio, ou seja, não existe um único método que seja utilizado em todas as fases do processo.

Alguns esforços estão sendo realizados neste sentido, como as extensões feitas pelo SEI [SEI96], utilizando o FODA; o DAGAR [KLI95]; e o método DADO [MAC97]. No entanto, estes esforços unem métodos diferentes para a análise, projeto e implementação do domínio, o que dificulta a transição entre as fases. Além disso, são poucos os métodos que definem claramente uma fase para a análise de risco do domínio. Este planejamento é importante, pois muitas vezes características como a abrangência do domínio, sua imaturidade, entre outras, inviabiliza a construção de uma infra-estrutura de reutilização para aquele domínio.

- existe uma tendência dos métodos em utilizar construções do paradigma OO. No entanto, nenhum deles utiliza construções OO que enfatizem a reutilização em alto nível, como padrões [GAM94] e *frameworks* [FAY97].
- por enfocarem com mais ênfase uma ou outra fase da ED, os métodos não enfatizam a importância da transição dos modelos de domínio para as representações arquiteturais, de forma natural, pois esta transição não é trivial. Somente o DAGAR [KLI95] apresenta esta preocupação.

Nome	Delimita as fronteiras do domínio	Distinção entre similaridades e diferenças	Entendimento dos modelos pelo usuário	Fase da ED que dá maior ênfase	Paradigma de desenvolvimento adotado	Planejamento do domínio	Gerenciamento componentes reutilizáveis
FODA	sim	sim, principalmente através do modelo de features	trabalha com diversos modelos e possui uma taxonomia de termos do domínio	análise do domínio	estruturado com OO	não	não
ODM	sim	sim, através dos modelos descritivos	utiliza múltiplas visões para a mesma informação	análise do domínio, já preparando para projeto do domínio	possui similaridades com o modelo de objetos	sim	não. No ambiente DAGAR sim.
EDLC	sim	sim, principalmente através de hierarquias de gen/esp.	Através do entendimento dos modelos OO	Análise de domínio e projeto. Com o KBSEE, enfatiza melhor o projeto e a implementação	OO	não	sim, no KBSEE
KAPTUR	sim, na primeira fase do método	não tem esta preocupação enfatizada	não tem esta preocupação enfatizada	projeto e implementação do domínio	tendência para OO	não	sim

Tabela A.1- Comparação entre os principais métodos de ED