Universidade Federal do Rio de Janeiro
Coordenação do Programa de Pós-Graduação em Engenharia
Programa de Engenharia de Sistemas e Computação

Exame de Qualificação

UMA PROPOSTA DE PROJETO ARQUITETURAL BASEADO EM COMPONENTES NO CONTEXTO DE ENGENHARIA DE DOMÍNIO

Aluna: Ana Paula Terra Bacelo Blois

Orientadoras:
Cláudia Maria Lima Werner e Karin Becker

COPPE – RJ
Janeiro de 2004

Índice

Índice de Figuras

Índice de Tabelas

1 Introdução

Observa-se que as Ciências Exatas, que têm forte base em tecnologia, apresentam pesquisam com um ciclo de vida mais iterativo do que comparado a outras Ciências onde evoluções ocorrem com menor velocidade. Em Engenharia de Software, grande parte das pesquisas é desenvolvida com base nas abordagens de desenvolvimento de software que vêem evoluindo.

Até a década de 80, o paradigma estruturado para desenvolvimento de software se destacava, onde a organização de programas era baseada em processos vinculados ao problema a ser resolvido, com ênfase na modularização de sistemas em termos de subsistemas vinculados a processos. A reutilização dos artefatos gerados (procedimentos e funções) tornava-se complicada, uma vez que a solução para um processo do negócio dependia se uma série de artefatos espalhados pelo sistema.

Em meados da década de 80 surgiu o paradigma de orientação a objetos (OO) aplicado em maior escala nos anos 90. Na OO, um sistema é organizado em termos de classes que encapsulam seus dados e comportamentos. Esta abordagem de desenvolvimento também privilegia a modularização de um sistema, mas em termos de classes. No entanto, outros benefícios não existentes no paradigma estruturado podem ser citados. Um deles é o uso de uma linguagem de modelagem unificada – Unified Modeling Language (UML), a qual se trata de uma combinação de várias propostas de notações para projeto OO, onde modelos de diferentes níveis de abstração e artefatos são propostos para que, de forma combinada, possam dar melhor apoio ao projeto OO. Com o projeto OO, tornam-se mais concretos os benefícios para prover reutilização, em especial através de tecnologias como *frameworks* (FAYAD, 1997) (através do conceito de classes abstratas), padrões (GAMMA *et al.*, 1994) (como uma proposta de reutilização de experiências recorrentes de projeto) e componentes (pela sua principal característica de ser auto-contido e com interfaces bem definidas) (BROWN, 2000) (SAMETINGER, 1997).

Com o surgimento do paradigma OO, observou-se que o foco da reutilização vinha ocorrendo em termos de código fonte e, dentro desta linha, várias bibliotecas foram surgindo no mercado (e.g. bibliotecas para construção de interfaces gráficas). No entanto, a reutilização não é uma abordagem somente aplicável ao contexto de implementação de

um sistema, ou seja, é possível reutilizar artefatos de análise, projeto, desde que propostos de forma a prover esta reutilização.

Dentre as tecnologias de reutilização que foram bastante aplicadas no contexto de código fonte destaca-se a de componentes. Embora tal tecnologia tenha sido aplicada inicialmente com este propósito, percebe-se uma forte tendência no uso de componentes (SAMETINGER,1997) (BROWN, 2000) como artefatos de projeto e implementação, os quais possuem características que privilegiam a reutilização. Nesta área, existem métodos como UML Components (CHEESMAN e DANIELS, 2001), Kobra (ATKINSON *et al.*, 2002) e Catalysis (D'SOUZA e WILLS, 1999) que propõem processos para a construção de aplicações baseadas em componentes, alguns deles combinando o uso de *frameworks* e padrões. No entanto, em geral, estes métodos não possuem ferramentas de suporte apropriado, apresentam uma certa complexidade para a obtenção de requisitos de uma aplicação e também não deixam claro como ocorre o processo de reutilização dos componentes gerados.

1.1 Motivação e Histórico da Pesquisa

A proposta de trabalho que está sendo apresentada nesta monografia foi motivada por um conjunto de pesquisas desenvolvidas desde 2000 junto ao PPGCC-FACIN-PUCRS (BLOIS, 2000) (BLOIS, 2001). Neste contexto, foram estudados ambientes da área de Ensino Colaborativo Suportado por Computador (do inglês *Computer Supported Cooperative Learning* - CSCL), baseados em tecnologias de reutilização como *frameworks*, padrões e componentes. Tais ambientes foram estudados com o intuito de avaliar o apoio oferecido na criação e reutilização de aplicações por eles geradas. Foram estudados alguns *frameworks* e ambientes baseados em componentes para o domínio de CSCL (BECKER e BLOIS, 2001). Neste estudo, se observou dentre outras coisas, que as propostas de CSCL não eram flexíveis do ponto de vista de reutilização. Seus usuários tinham que se adaptar a proposta de estrutura e serviços disponíveis pelo ambiente, não permitindo que os profissionais responsáveis pela construção da aplicação gerada pudessem adapta-la as necessidades do seu público alvo.

Com base nesta motivação, foi proposta uma abordagem para apoio ao projeto arquitetural, baseado em componentes, no domínio de CSCL (BECKER e BLOIS, 2002),

com o objetivo de tornar o processo de projeto mais flexível, adaptado às necessidades dos projetistas de uma aplicação deste domínio. Esta abordagem foi apresentada na COPPE/Sistemas como proposta de Plano de Doutorado, em março de 2003, quando foi dado início a primeira atividade de pesquisa: levantamento dos requisitos do domínio de CSCL. O resultado desta tarefa foi um conjunto de requisitos para CSCL proposto em (BLOIS, 2003a), no contexto da disciplina de Tópicos em Engenharia de Software I.

Após esta atividade, foi proposto como desafio a avaliação da Infra-Estrutura de Reutilização Odyssey em desenvolvimento na COPPE desde 1997. O objetivo desta avaliação foi observar como a abordagem de Engenharia de Domínio (ED) existente no Odyssey poderia apoiar a proposta de projeto arquitetural para o contexto de CSCL. Neste sentido, foi criado um domínio de CSCL, com base no processo Odyssey-ED proposto por BRAGA (2000). Como resultado desta experiência, foi possível chegar a algumas conclusões iniciais, sendo elas:

- ☺ A abordagem de reutilização baseada em Engenharia de Domínio proposta na infra-estrutura Odyssey poderia ser utilizada para a construção de aplicações no domínio de CSCL;
- ☹ O foco do processo Odyssey-DE é na análise de domínio, onde as atividades previstas para apoio ao projeto de domínio estavam parcialmente disponibilizadas;
- ☹ Grande parte do apoio oferecido pelo Odyssey se dedica a etapa de análise de domínio e portanto, embora previsto, não oferece apoio ao projeto arquitetural do domínio;
- ☺ A infra-estrutura do Odyssey, principalmente o suporte já existente para análise de domínio, pode ser aproveitada e adaptada para suporte ao projeto arquitetural da proposta de pesquisa em CSCL.

Tendo como base estas observações iniciais, chegou-se a conclusão de que o Odyssey possui um ferramental base importante que poderia ser empregado nesta proposta de tese. O foco desta pesquisa está nas questões que envolvem o projeto arquitetural em si, baseado na tecnologia de componentes, dedicado ao contexto de ED.

Neste sentido, estudos a respeito das áreas de pesquisa relacionadas a esta proposta de trabalho foram realizados, os quais estão organizados nos Capítulos que se seguem. As áreas de pesquisa estudadas são:

- Desenvolvimento Baseado em Componentes. Esta área de pesquisa já havia sido explorada em (BECKER e BLOIS, 2002). No entanto, no contexto desta pesquisa, foi feita uma análise focando o suporte dos métodos de DBC ao projeto arquitetural de componentes, os fatores a serem considerados na construção de uma arquitetura de componentes, e sobretudo o apoio à reutilização, conforme apresentado no Capítulo 2 .
- Estilos Arquiteturais e Linguagens de Descrição Arquitetural: Este estudo foi desenvolvido para avaliar o tipo de suporte oferecido por estas abordagens para apoio a construção de uma arquitetura de software, dando destaque àquelas arquiteturas baseadas na tecnologia de componentes. O resultado deste trabalho é apresentado em (BLOIS, 2003b) e mais resumidamente na Seção 3. Estas tecnologias são consideradas como uma possibilidade de apoio a construção de arquiteturas de componentes desta proposta de trabalho, como registrado na Seção 6.2.
- Engenharia de Domínio: O objetivo foi obter informações a respeito dos métodos de ED existentes, os artefatos utilizados nas fases que compreendem a ED e obter uma análise crítica do suporte oferecido pelos processos descritos na literatura, com destaque ao Odyssey-DE no contexto da infra-estrutura de reutilização Odyssey. O estudo está apresentado aos longo dos Capítulos 4, 5 e 6.

Nos Capítulos 2, 3, 4, 5 e 6, os problemas detectados com relação a Métodos de DBC, Processos de ED e Projeto Arquitetural de Componentes são relatados. Ao final dos capítulos 5 e 6, são detalhadas as propostas de apoio ao processo e projeto arquitetural de componentes desta proposta de pesquisa, atendendo aos problemas detectados nas abordagens encontradas na literatura.

A solução de projeto arquitetural que se pretende desenvolver deve apoiar não só a construção de artefatos para o domínio de CSCL, mas também para qualquer outro domínio onde a ED é sugerida. Neste sentido, CSCL será o domínio a ser explorado durante a etapa de experimentação desta proposta de pesquisa. Tendo em vista a amplitude do domínio em questão, na etapa de experimentação poderá vir a ser explorada um conjunto específico de serviços que compreendem CSCL.

1.2 Questão de Pesquisa e Objetivos

Tendo em vista a evolução da motivação desta pesquisa, apresentada na Seção anterior, propõe-se a seguinte reformulação da questão de pesquisa:

Como pode ser melhorado o projeto arquitetural baseado em componentes no contexto de Engenharia de Domínio?

Para responder esta questão de pesquisa, este trabalho de propõe a:

- Definir um processo de Engenharia de Domínio (ED), com detalhamento das atividades que compreendem a construção do projeto arquitetural de componentes do domínio, o qual encontra-se parcialmente apresentado nesta monografia;
- Propor um conjunto de heurísticas para apoiar a especificação de componentes na atividade de projeto de Engenharia de Domínio;
- Propor um conjunto de heurísticas para a construção de microarquiteturas[1] de componentes na atividade de projeto de Engenharia de Domínio;
- Utilizar abordagens para a especificação de arquiteturas de componentes tais, como Estilos Arquiteturais e Linguagens de Descrição Arquitetural (HOEK, 2003) (DASHOFY, HOEK E TAYLOR, 2002) (SHAW e GARLAN, 1996) (TEIXEIRA, 2003). Estas abordagens apóiam a compreensão da arquitetura gerada pela abordagem de projeto arquitetural

[1] Microarquiteturas visam representar um conjunto de componentes que trocam um grande volume de mensagens no contexto de um domínio

discutida nesta monografia, sua implementação e sobretudo sua reutilização no contexto de uma aplicação, baseado no domínio modelado;

- Adotar esta proposta de projeto arquitetural numa infra-estrutura de reutilização já existente (e.g. Odyssey-SDE);
- Avaliar a proposta de pesquisa através de experimentos desenvolvidos na infra-estrutura.

1.3 Organização da Monografia

O restante desta monografia está organizado em sete Capítulos. Os dois primeiros apresentam o estado da arte da área de DBC e Arquiteturas de Software. O Capítulo 4 apresenta a Engenharia de Domínio com ênfase no uso dos artefatos para análise e projeto. O Capítulo 5 analisa as proposta de processo para ED existentes na literatura, seus problemas e uma proposta inicial de processo no contexto desta pesquisa. O Capítulo 6 mostra o estado da arte referente ao projeto arquitetural de componente e um conjunto de alternativas identificadas no contexto desta pesquisa para apoio ao projeto arquitetural, as quais serão exploradas no contexto desta pesquisa. O Capítulo 7 apresenta considerações finais e atividades que envolvem esta proposta de pesquisa como um todo.

Conforme mencionado anteriormente, no Capítulo 2, são discutidos os conceitos associados ao desenvolvimento baseado em componentes (DBC) e como suas arquiteturas e métodos têm viabilizado o projeto arquitetural de aplicações baseadas em componentes. São analisadas as propostas envolvendo a construção de arquiteturas para componentes (BROWN, 2000) e os principais métodos de DBC da literatura: UML Components (CHEESMAN e DANIELS, 2001), Kobra (ATKINSON *et al.*, 2002) (ATKINSON *et al*., 2001) e Catalysis (D'SOUZA e WILLS, 1999).

Arquiteturas de software buscam definir a forma como seus elementos colaboram e, uma vez que o projeto arquitetural de componentes precisa expressar tal informação, é preciso então avaliar a aplicação de mecanismos que expressam a coordenação de elementos arquiteturais. Assim, no Capítulo 3, são discutidas as arquiteturas de software, e mais especificamente estilos arquiteturais e linguagens de descrição arquiteturais para a especificação de arquiteturas baseadas em componentes.

Para fundamentar o conjunto de soluções para o projeto arquitetural em ED proposto nesta monografia, é apresentado, no Capítulo 4, uma análise da Engenharia de Domínio sob o ponto de vista dos artefatos utilizados em cada fase do processo de ED e como são relacionados os artefatos das diferentes fases.

No Capítulo 5, são detalhados alguns processos de ED com foco em DBC com intuito de identificar o apoio almejado e efetivamente disponibilizado para o desenvolvimento baseado em componentes. Tendo em vista os problemas identificados nos processos até então disponíveis, é proposto o processo ***CBD-Arch-ED***, o qual enfatiza o desenvolvimento baseado em componentes na Engenharia de Domínio, com foco na atividade de projeto arquitetural.

No Capítulo 6, são discutidas as alternativas de apoio oferecido pelos métodos de ED, de DBC e da área de pesquisa de linha de produto, para a atividade de projeto arquitetural. Ao final, é apresentado um conjunto de alternativas para a construção de uma proposta de apoio ao projeto arquitetural a ser explorado durante esta pesquisa, indicando soluções para os problemas identificados nas abordagens existentes.

Por último, no Capítulo 7 são resumidas a motivação, os objetivos, as atividades que compreendem a pesquisa proposta, identificando àquelas já desenvolvidas, as publicações produzidas sobre o tema de pesquisa, as propostas para avaliação, os resultados esperados com a proposta de pesquisa e, ainda, um cronograma de atividades previsto para o desenvolvimento do trabalho proposto.

2 Desenvolvimento Baseado em Componentes

Neste Capítulo são apresentados conceitos básicos relacionados ao Desenvolvimento Baseado em Componentes (DBC), as características que norteiam uma arquitetura de componentes e alguns métodos que apóiam o processo de DBC.

2.1 Definições Básicas

A tecnologia de componentes tem por objetivo apresentar soluções reutilizáveis para o processo de desenvolvimento de software, buscando uma maior produtividade com menor custo.

A necessidade de ter maior produtividade com menor custo no desenvolvimento de software e as mudanças que estão ocorrendo com a tecnologia de computação (e.g. computação distribuída, desenvolvimento para WEB, necessidade de estabelecimento de padrões para construção de aplicações) são alguns dos principais fatores que motivam o uso da abordagem de Desenvolvimento Baseado em Componentes (DBC).

Foram apresentadas e discutidas em BROWN e WALLNAU (1998) diferentes definições do que venha a ser um componente de software, as quais tratam de forma mais explícita ou não questões relativas à dependência de contexto (tipo de aplicação no qual pode ser utilizado) e nível de abstração. As definições discutidas são:

1) "um componente é uma **parte** do sistema a qual é **não trivial, independente e substituível**, e que possui uma função clara no contexto de uma **arquitetura bem definida**". (Philippe Krutchen, Rational Software);
2) "um componente de software executável é um pacote de um ou mais programas **ligados dinamicamente**, gerenciados com uma unidade e acessados através de **interfaces** documentadas, as quais podem ser **descobertas em tempo de execução**". (Gartner Group);
3) "um componente de software é uma **unidade de composição** com interfaces que possuem **contratos específicos** e **dependências de contextos** especificadas". (Clements Szyperski, Component Software);
4) "um componente de negócio representa a implementação de um **conceito autônomo de negócio ou processo**. Consiste de artefatos de software

> necessários para expressar, implementar e executar o conceito como um **elemento reutilizável de um grande sistema de negócio**". (Wojtek Kozaczynski, SSA)

As definições 1 e 3 enfatizam a necessidade de dependência de um contexto, sendo que esta característica pode ser inferida na definição 4, mas não na definição 2. As definições 1, 2 e 4 também salientam a questão da abstração do componente para prover o máximo de reutilização, o que não acontece claramente na definição 3. WERNER e BRAGA (2000) fizeram uma análise a respeito destas definições, e consideram que: 1) um componente pode ser visto como um elemento arquitetural que provê serviços através de um conjunto de interfaces; 2) um componente pode ser visto como um elemento implementacional que em tempo de execução pode ser acessado através de suas interfaces; 3) um componente é um elemento implementacional que faz parte de um contexto arquitetural e; 4) um componente representa a implementação de um conceito autônomo de um negócio ou um processo.

Uma outra definição bastante abrangente é a proposta por Sametinger (SAMETINGER, 1997), e que, segundo o autor, representa um consenso daquelas já existentes: "componentes de software reutilizáveis são artefatos autocontidos, claramente identificáveis, que descrevem ou realizam uma função específica e têm interfaces claras, documentação apropriada e um grau de reutilização definido". Um componente autocontido é aquele que não precisa de outro componente para ser reutilizável. Quando existe a necessidade de cooperação entre componentes, esta interação deve ser feita através de suas interfaces, ou seja, pelo conjunto de assinaturas de operações existentes no componente as quais podem ser invocadas por outro. Quando não existe a possibilidade de conexão entre eles, um terceiro componente deve ser utilizado, atuando como um conector para intermediar tal conexão. Para viabilizar a capacidade de reutilização de um componente, este deve ser claramente identificável, buscando facilitar a sua localização e, juntamente com a documentação, se ter conhecimento do potencial de reutilização do mesmo através de suas funcionalidades ali descritas (e.g. descrição das interfaces, casos de uso, localização, etc).

Segundo BROWN (2000), componentes e abordagens baseadas em componentes compartilham um conjunto de características similares. Dentre as características que se destacam estão:

1) Especificação: Define o comportamento do componente para situações específicas, restrições existentes para determinados estados em que se encontra um componente, e orientações para que clientes possam ter uma interação apropriada com o componente.
2) Uma ou mais implementações: Estas implementações devem estar em conformidade com a especificação. Cabe, portanto, a especificação permitir que diferentes implementações possam ser feitas a partir de uma mesma especificação.
3) Restrições quanto ao padrão de componentes: Componentes de software no contexto de um ambiente de implementação, podem fazer parte de um modelo de componentes (ex.: COM, J2EE, CORBA). Ao serem adotados no contexto de um modelo de componentes, suas especificações e implementações devem ser mapeadas para o contexto do modelo adotado, atendendo as exigências da abordagem de componentes utilizada. No contexto de um processo de DBC, a imposição de um modelo de componentes tem sido alvo de algumas discussões. Na comunidade de Engenharia de Software são comuns a discussões a respeito da independência ou não da atividade de projeto com relação à tecnologia adotada. Neste sentido, observa-se que algumas abordagens de desenvolvimento de software adotam o conceito de projeto de alto nível, para destacar um projeto sem vínculo à tecnologia de implementação e, projeto de baixo nível, já atendendo às imposições da tecnologia de desenvolvimento.
4) Abordagem de empacotamento: Componentes podem ser agrupados de diferentes formas. Geralmente, um pacote de componentes representa um grupo de serviços que têm um grau de relacionamento grande e que muitas vezes são utilizados em conjunto no contexto de um sistema.
5) Abordagem prática de um componente: Depois de empacotados, os componentes devem ser disponibilizados, através de uma ou mais instâncias do componente executável.

Embora os cinco elementos representem as características essenciais de um componente, segundo BROWN (2000), através de uma análise mais profunda, um componente pode ser avaliado sob três diferentes perspectivas:

1) Empacotamento: um componente visto como uma unidade a ser empacotada, distribuída e entregue, enfatizando a perspectiva de reutilização da tecnologia;
2) Serviço: componente é uma entidade de software que oferece serviços (operações e funções) para o usuário, através de interfaces bem definidas;
3) Integridade: componente é visto como uma unidade encapsulada, ou seja, um um conjunto de software que coletivamente mantém a integridade dos dados que ele gerencia, independente da implementação de outros componentes.

Em (TEIXEIRA, 2003), o autor afirma que estas perspectivas podem ter diferentes prioridades numa tecnologia de componentes. Tecnologias que utilizam componentes já compilados privilegiam a perspectiva de empacotamento, as que apóiam a construção de componentes genéricos privilegiam serviço, e aquelas tecnologias que trabalham com componentes registrados no ambiente de execução (ex.: componentes COM), enfatizam a perspectiva de integridade.

Nesta monografia, será adotado o conceito de componente proposto por SAMETINGER (1997). Este conceito enfatiza a questão de abrangência de um componente, contemplando artefatos que podem pertencer a diferentes fases de um ciclo de vida, a questão de encapsulamento, defendido pela comunidade de DBC, e sobretudo, a capacidade de reutilização em função, principalmente, das duas características acima citadas. Embora a pesquisa proposta nesta monografia tenha a intenção de apresentar soluções de projeto independentes de tecnologia de implementação, pode-se afirmar que as perspectivas de empacotamento e serviços, propostas por Brown (BROWN, 2000), serão contempladas em nível de projeto arquitetural.

2.2 Arquiteturas de Componentes

Assim como nos paradigmas estruturado e orientado a objetos, aplicações desenvolvidas através da abordagem de componentes devem fazer uso de métodos que

sistematizem o processo de desenvolvimento do software e, sobretudo, no contexto desse trabalho, que forneçam apoio à construção da arquitetura do software. Métodos como UML Components (CHEESMAN, 2001), Catalysis (D'SOUZA e WILLIS, 1999) e KobrA (ATKINSON *et al*., 2001) têm se destacado para apoio ao DBC e sobretudo em arquiteturas de software baseado em componentes. Tais métodos serão brevemente apresentados na Seção 2.3.

Segundo BROWN (2000), a partir do conhecimento inicial das necessidades do negócio, um método de DBC deve prever uma primeira etapa de entendimento do contexto, onde são definidos os requisitos iniciais da aplicação, e modelados os casos de uso que definem os usuários e as atividades por eles desempenhadas, bem como os tipos de negócio, ou seja, a representação dos principais elementos do negócio e seus relacionamentos no domínio. Para o desenvolvimento desta primeira etapa, o conhecimento do domínio e dos sistemas já existente é fator chave.

A etapa posterior envolve a definição da arquitetura, a qual é desenvolvida a partir dos aspectos estáticos e dinâmicos modelados na fase anterior. Atividades previstas na fase de definição de arquitetura são:

- **modelagem da arquitetura de componentes**, onde uma coleção de componentes relacionados são propostos e definidos. A arquitetura de componentes gerada pode organizá-los de acordo com o tipo de serviço oferecido (e.g. infra-estrutura, negócio);
- **modelagem do contexto**, para entender o escopo do software que está sendo desenvolvido. Em geral, este entendimento compreende uma especificação mais exata das responsabilidades dos componentes em relação aos casos de uso do domínio;
- **modelagem de interface**, para obter um conjunto de interfaces candidatas e descrevê-las em detalhe. Para dar início a esta atividade, sugere-se que o projetista considere os resultados da modelagem de contexto e dos tipos do negócio para eleger o primeiro conjunto de interfaces candidatas;
- **definição de interface**, na qual interfaces obtidas como resultado da atividade anterior são descritas em detalhe (e.g. com o uso da OCL – Object Constraint Language) (CHEESMAN e DANIELS, 2001).

A última etapa prevista num método de DBC é a proposta de solução para o problema, onde os componentes são efetivamente implementados. Nesta etapa, deve-se viabilizar a comunicação entre componentes gerados com os sistemas já existentes e ainda componentes de terceiros. Como resultado desta atividade obtém-se a entrega do sistema. A Figura 1 ilustra o esquema de funcionamento de um método de DBC. Este esquema apresenta uma perspectiva macro do conjunto de elementos e alternativas de solução para o DBC. Neste sentido, cabe então aos métodos e tecnologias para apoio ao DBC efetuar as devidas extensões nestes elementos macro e propor soluções de negócio e que ao mesmo tempo atendam as perspectivas da tecnologia de componente.

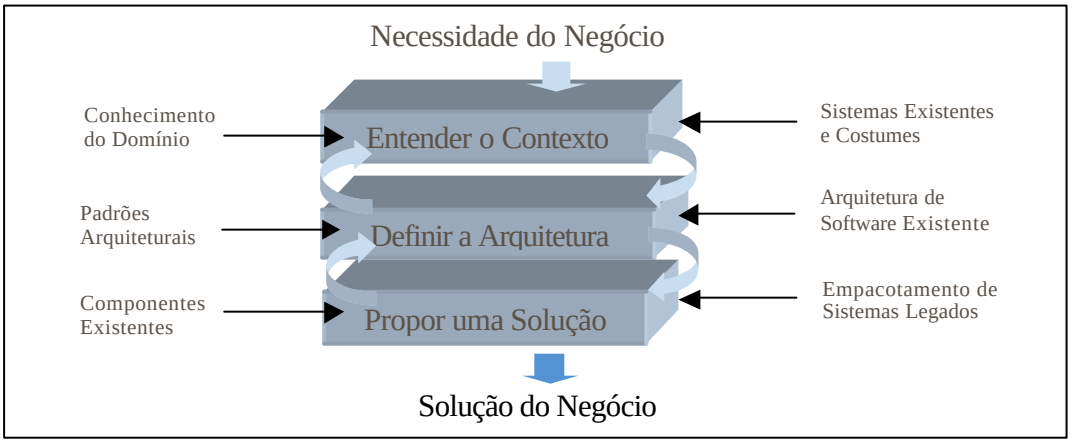

Figura 1: Elementos de uma Abordagem de DBC - extraído de (TEIXEIRA, 2003)

Do ponto de vista da organização dos componentes gerados por um método de DBC, alguns autores como (D'SOUZA e WILLIS, 1999), (BROWN, 2000) e (HERZUM e SIMS, 2000), sugerem que uma arquitetura de componentes deve ser organizada em diferentes camadas, onde os componentes possuem diferentes níveis de abstração, e as camadas inferiores prestam serviços para as camadas superiores. A primeira camada compreende os componentes de negócio, que implementam um conceito ou processo de negócio autônomo. Os componentes de negócio são artefatos necessários para representar, implementar e implantar um conceito de negócio, os quais refletem as dependências entre os diferentes conceitos do negócio e forneçam meios de interagir e colaborar entre si. A camada intermediária possui os componentes utilitários, que prestam serviços genéricos necessários ao desenvolvimento das aplicações. Componentes

utilitários não pertencem a nenhum domínio específico, mas são utilizados por inúmeras aplicações de negócio (e.g. controle de calendários, agenda de endereços, formulários genéricos). Do ponto de vista de reutilização, os componentes utilitários são os mais reutilizados, comparados aos componentes de negócio, por exemplo. A última camada compreende os componentes de infra-estrutura. Estes componentes são responsáveis por estabelecer a comunicação com as plataformas de execução do software, ou seja, estão mais vinculados às questões de tecnologia de desenvolvimento do que os componentes das camadas superiores. Dentre os serviços providos por componentes de infra-estrutura, estão: comunicação entre componentes distribuídos, persistência, tratamento de exceções e portabilidade. A Figura 2 mostra a arquitetura em camadas de componentes, salientando a camada de componentes de infra-estrutura que possuem uma forte relação com a tecnologia de componentes adotada na aplicação gerada.

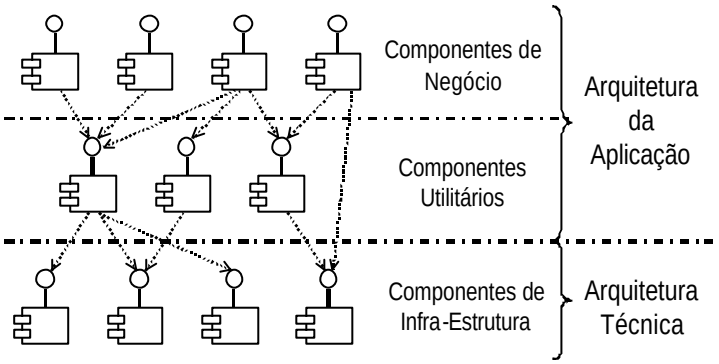

Figura 2: Componentes de uma Arquitetura em Camadas – extraído (TEIXEIRA, 2003)

Esta estrutura de divisão em camadas visa padronizar em três grandes categorias os componentes gerados/utilizados num processo de DBC, de acordo com o tipo de suporte oferecido e nível de abstração. Ao se adotar uma tecnologia de componentes para construção da aplicação, esta estrutura de camadas deverá sofrer adaptações para as características da tecnologia (e.g. J2EE, JavaBeans, COM). Cada tecnologia tem sua própria forma de estruturar e compartilhar os serviços dos componentes de uma aplicação, bem como uma forma particular de estabelecer a comunicação entre os diferentes componentes.

2.3 Métodos de DBC

Existem na literatura alguns métodos específicos para apoio ao processo de DBC, entre os quais UML *Components*, Catalysis e Kobra, brevemente analisados nas próximas seções.

2.3.1 UML *Components*

UML *Components* é um método para DBC que utiliza os recursos da UML para o contexto de componentes (CHEESMAN e DANELS, 2001). O método UML *Components* se propõe a dar apoio ao processo de DBC com ênfase nesta tecnologia se comparado aos métodos OO tradicionais, os quais utilizam o padrão UML. A UML original não dá apoio ao processo de DBC. O apoio a componentes na UML, e em métodos OO como OMT e RUP está praticamente restrito à fase de implementação de uma aplicação, onde componentes executáveis são criados ou utilizados (componentes de terceiros).

O método UML *Components* prevê duas etapas principais: a etapa de requisitos e a etapa de especificação.

Na etapa de requisitos, são construídos o diagrama conceitual do negócio e os casos de uso do domínio. O primeiro diagrama possibilita ao projetista obter conhecimento sobre os conceitos estáticos referentes à aplicação e os casos de uso dão a visão dos processos existentes no futuro sistema.

Com base nos artefatos gerados na etapa de requisitos, é que ocorre a etapa de especificação. Nesta etapa, é construído o diagrama de tipos do negócio que busca expressar as informações relevantes do modelo do negócio para a construção do sistema, e que está relacionado ao diagrama conceitual do negócio, construído na primeira etapa. Ainda na etapa de especificação, são construídos os diagramas de especificação de interfaces necessárias para a construção dos componentes, o diagrama de especificação dos componentes que expressa um conjunto de tipos de interfaces, sendo estes apoiados pelos artefatos já existentes no contexto da etapa de requisitos e no próprio diagrama conceitual do negócio. O diagrama de arquitetura de componentes é que define o contexto no qual ocorrerão as dependências entre interfaces especificadas nos diagramas acima. A Figura 3 mostra a proposta de organização de UML *Components*.

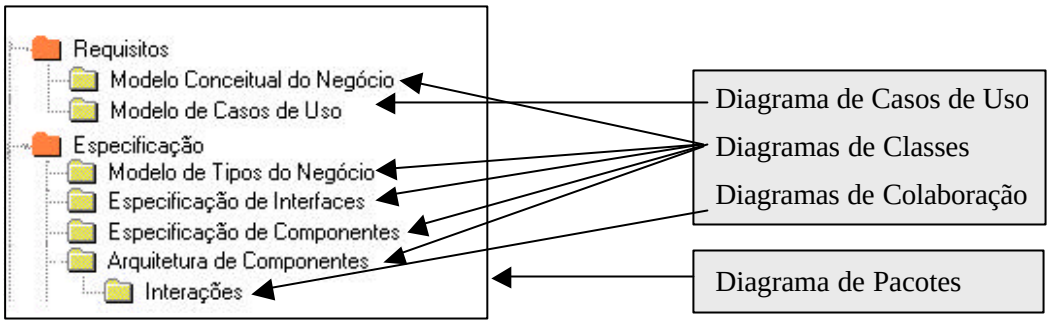

Figura 3: Proposta de DBC usando UML *Components*

UML *Components* ainda faz uso de mecanismos de extensão em UML através de estereótipos, permitindo que novos elementos possam ser criados ou reutilizados no modelo do negócio. Ainda utiliza recursos da OCL (Object Constraint Language) para representar, através de expressões lógicas restrições, pós e pré-condições existentes na aplicação, além de outros recursos da linguagem que possibilitam complementar o grau de precisão da modelagem efetuada.

Para apoio a reutilização, o método possui um diagrama de pacotes que agrega todos os diagramas das etapas de requisitos e especificação do modelo do negócio gerado. No entanto, na documentação analisada, não está explícito como ocorre este processo de desempacotamento dos componentes e de sua posterior reutilização.

Observa-se, também, que as etapas propostas pelo método equivalem às etapas de análise e projeto existentes em métodos OO convencionais (e.g. RUP), não existindo apoio ao processo de implementação para os componentes gerados em UML *Components.*

2.3.2 Catalysis

Catalysis é um método de desenvolvimento baseado em componentes baseado na notação da UML, principalmente no uso de diagramas de classes, casos de uso, interação, colaboração e de pacotes (*packages*) (D'SOUZA e WILLIS, 1999). Catalysis dá apoio à construção de projetos de grande escala e que exigem alto grau de precisão. Para tanto, o método procura dar suporte: ao DBC, na definição das interfaces dos componentes e nos refinamentos sucessivos deste processo de construção; ao processo de integração dos subsistemas de um mesmo modelo de negócio; à integridade do processo de refinamento

dos requisitos do negócio através das especificações dos componentes e; ao projeto orientado a objeto, mais especificamente no uso coerente e consistente da notação UML no processo de projeto.

O método Catalysis está baseado em três princípios de modelagem fundamentais: tipos, colaboração e refinamento. O conceito de tipo dá a idéia de comportamento externo do objeto da aplicação, ou seja, quais são os elementos visíveis externamente pelo usuário de uma dada aplicação. A idéia de refinamento possibilita que requisitos inicialmente levantados na aplicação possam ser refinados à medida que se tem um conhecimento mais aprofundado do domínio, elevando por sua vez o nível de abstração dos componentes. Por último, a idéia de colaboração parte do pressuposto que componentes de um domínio têm que de alguma forma colaborar e que, para isto, a comunicação entre as interfaces destes componentes deve ser previamente projetada. O método Catalysis pressupõe que estas três premissas sejam de alguma forma apoiadas através das diferentes fases do ciclo de vida de software, organizadas através das camadas apresentadas na Figura 4.

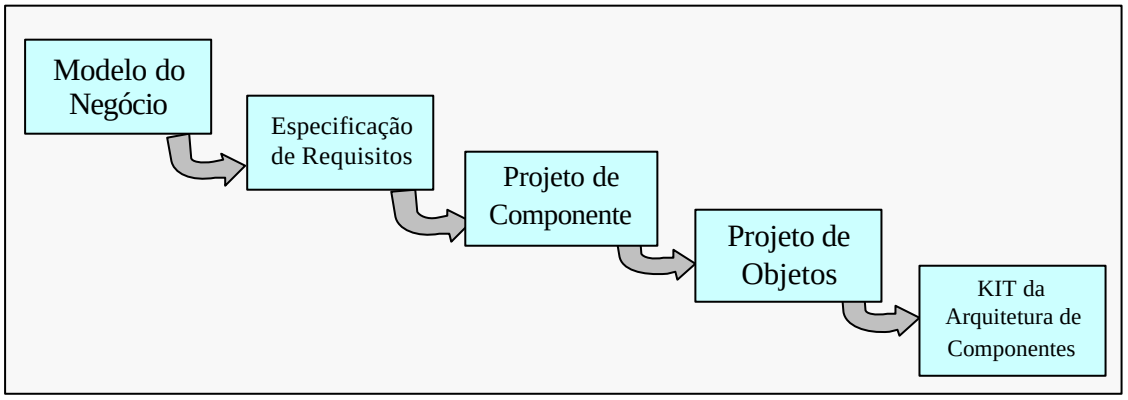

Figura 4: Camadas de desenvolvimento segundo o método Catalysis
(extraído de (D'SOUZA e WILLIS, 1999))

O Modelo do Negócio descreve os elementos que compreendem o ambiente do usuário, sem qualquer comprometimento notacional rotulado pelo método. Na especificação de requisitos, faz-se a análise dos elementos que devem ser posteriormente implementados no software, já identificando nesta etapa os componentes que mais se destacam no ambiente do negócio (*mandatory*), os quais serão projetados na fase

posterior. Na camada de projeto de componentes, se faz uma descrição num nível de abstração mais elevado de suas interfaces e da forma como a colaboração neste componente vai ocorrer, baseado na especificação de requisitos de cada componente. No projeto de objetos, baixa-se o nível de abstração dos componentes especificados na fase anterior para o nível das classes da linguagem de programação em que serão desenvolvidos para que possam ser codificados. Por último, a camada Kit da Arquitetura de Componentes prevê que os componentes sejam integrados na forma de uma arquitetura, para que os mesmos possam colaborar conforme previsto na especificação dos componentes.

No que se refere à reutilização, o método ainda não apresenta um conjunto de elementos que definam o quê e como reutilizar. Essa atividade deve ser feita pelo usuário do método pelo uso dos modelos existentes, os quais geralmente possuem uma especificação bastante consistente.

A proposta do método Catalysis é dar suporte ao desenvolvimento baseado em componentes através da definição das interfaces dos mesmos; integração de componentes através da adaptação das diferentes interfaces dos componentes para um mesmo modelo de negócio; especificação precisa do projeto e refinamento dos requisitos do negócio para que se possa chegar ao nível de codificação do componente do negócio; coerência e consistência nos diferentes níveis do processo de desenvolvimento do negócio com a notação UML. No entanto, pelo fato de não existirem ferramentas consolidadas no mercado (e.g. Rational Rose) que dêem apoio ao método, fica difícil avaliar efetivamente as facilidades previstas para o método, dependendo muito de um esforço da equipe do desenvolvimento do software que está aplicando na consistência das etapas, atividades e restrições previstas pelo mesmo.

2.3.3 Kobra

Kobra é um método de DBC (ATKINSON *et al*, 2002) voltado para linha de produtos[2], o qual utiliza o conceito de componente para todas as fases do ciclo de vida de um software e permite que componentes de alto nível, descritos em UML, sejam implementados usando tecnologias de componentes conhecidas (e.g. JavaBeans, CORBA, COM).

Uma das principais características de Kobra é a separação da descrição abstrata de um componente de sua implementação. Outra característica de Kobra que o difere dos outros métodos de DBC, é o fato de basear-se na premissa de que todo o artefato UML usado no desenvolvimento de um sistema deve ser estruturado e organizado em torno de componentes essenciais de um sistema. Neste sentido, todo o artefato (e.g. requisitos, arquiteturas, projeto) deve ser criado sob a perspectiva de um e somente um componente num sistema. Tal premissa não é um consenso na literatura, pois algumas abordagens partem da premissa que outros artefatos de análise devem ser utilizados para identificar componentes para a aplicação (BRAGA, 2000).

Conforme a Figura 5, a descrição de um componente em Kobra é composta de duas partes: **especificação** e **realização**, as quais correspondem as fases de análise e projeto de software tradicional, respectivamente. A especificação descreve as propriedades visíveis externamente de um componente em termos de um ou mais diagramas estáticos (modelo estrutural), um conjunto de especificações de operações obtido pelo modelo funcional e um diagrama de estados dando a visão comportamental. Por outro lado, a realização descreve como componentes realizam as propriedades especificadas em termos de interações com outros componentes. Esta realização é obtida através de modelo estrutural, apresentando uma visão estrutural no nível de projeto, um conjunto de diagramas de interação, dando uma visão orientada a interação, e um conjunto de diagrama de atividades, os quais dão uma visão algorítmica do componente.

As tarefas de especificação e realização de componentes Kobra fazem parte do contexto de engenharia do framework, a qual tem por objetivo construir uma estrutura

[2] Linha de produtos visa disponibilizar um conjunto de artefatos que fazem parte de uma linha de produto, e que por sua vez podem ser reutilizados na construção de vários produtos que pertençam àquela linha (ATKINSON *et al*., 2002).

genérica que compreende todas as variantes[3] de uma família de produtos, incluindo informações sobre suas características comuns e variáveis. A reutilização da atividade de engenharia de framework se dá através da atividade de engenharia da aplicação. Esta última, instancia o framework para atender as particularidades de uma família de produto, reunindo as necessidades especificas para diferentes consumidores.

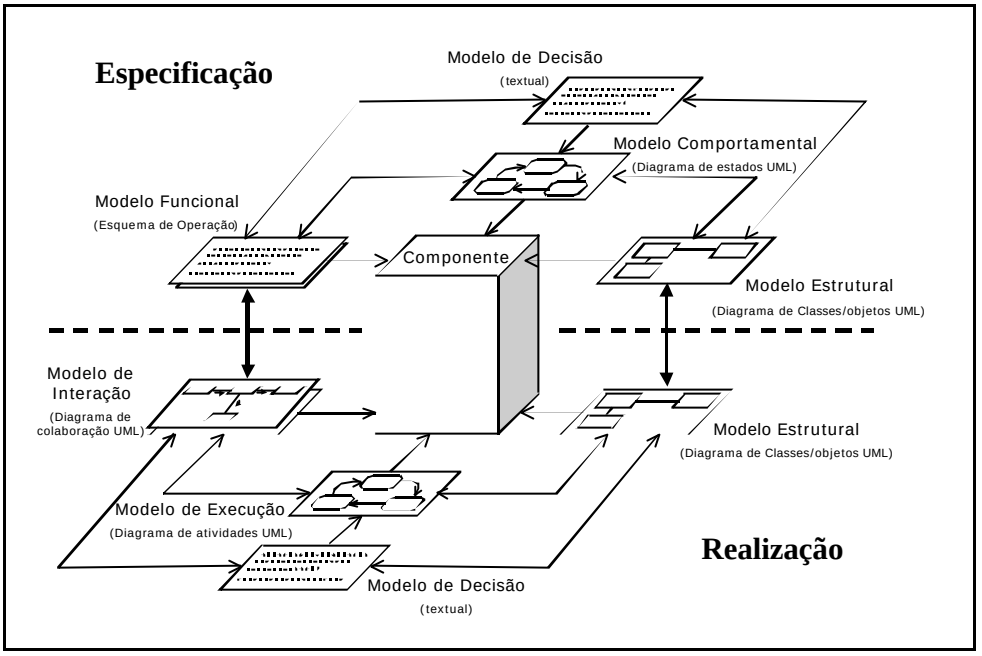

Figura 5: Especificações e Realizações de um Componente KOBRA

2.4 Considerações Finais

Os métodos de DBC apresentados acima representam as principais propostas para estruturação do processo de DBC, e sem dúvida, são de grande valor para a comunidade de componentes. Apesar de terem o propósito de reutilização, observa-se que as experiências com o uso dos métodos viabilizam, no máximo, a manutenção de artefatos gerados. Neste sentido a grande preocupação é no desenvolvimento para utilização, ou seja, prover os artefatos a serem utilizados, do que fazer reutilização como conseqüência da modelagem desenvolvida. Até o momento não existem relatos de experiência de pessoas, não vinculadas a construção do método, que o aplicaram num determinado contexto. Os relatos existentes são desenvolvidos por membros da equipe de

[3] Uma variante em linha de produto representa uma característica que poderá existir para um produto específico daquela linha e para outro não.

desenvolvimento do método, e mesmo assim, alguns problemas como os apresentados abaixo podem ser observados.

Um ponto negativo observado na descrição dos métodos é referente aos rastros existentes entre os artefatos de reutilização. No decorrer do processo sugerido pelos métodos, uma série de artefatos do maior ao menor nível de abstração são criados. No entanto, não fica claro como estes são ligados uns aos outros e como podem ser rastreados aqueles artefatos que foram gerados ao longo do processo, dando origem aos componentes da arquitetura proposta.

Embora não tenha sido o foco deste Capítulo a descrição detalhada dos métodos, observou-se durante a pesquisa que, para atender a proposta de apoio do método, é feito o uso de artefatos que não seguem o padrão de especificação da UML, embora esta última prevê este tipo de extensibilidade. Isto pode vir a ser um aspecto negativo desses métodos, uma vez que eles não possuem um ferramental de domínio público para suporte a especificação de seus artefatos e, neste sentido, aplicar o método no contexto de ferramentas como Rational Rose, ArgoUML, etc, se torna difícil. Na nova especificação da UML (Versão 2.0), o suporte ao projeto baseado em componentes está melhordefinido, diferente das propostas anteriores onde o componente só poderia representar um elemento implementacional, sem qualquer possibilidade de especificação arquitetural. Com esta nova especificação, ferramentas baseadas em UML terão melhor suporte a especificação de componentes e arquiteturas baseadas em componentes. No entanto, o suporte aos métodos de DBC ainda é parcialmente inviável.

3 Arquiteturas de Software

Neste Capítulo são apresentados conceitos básicos relacionados a Arquiteturas de Software, principalmente abordagens de estilos arquiteturais e linguagens de descrição arquitetural, com foco em DBC.

3.1 Conceitos Básicos

Em quaisquer das abordagens de DBC citadas na Seção 2.3, existem atividades onde espera-se que os artefatos criados trabalhem de forma coordenada, relacionados entre si, para satisfazer os requisitos da aplicação. Estes artefatos têm diferentes níveis de suporte, e, portanto, necessitam uns dos outros para que possam efetivamente contribuir para a solução de software. Quanto maior é o software que está sendo construído, maior será a complexidade existente aos papéis dos artefatos construídos, as relações existentes e sobretudo na estrutura final do sistema. A esta atividade, geralmente efetuada durante a fase de projeto da aplicação, dá-se o nome de desenvolvimento da arquitetura do software.

Existem algumas definições de arquiteturas de software que são destacadas neste trabalho. A primeira é a de D'SOUZA e WILLS (1999), os quais afirmam que a arquitetura de um sistema consiste da(s) estrutura(s) de suas partes (incluindo as partes de software e hardware envolvidas no tempo de execução, projeto e teste), a natureza e as propriedades relevantes que são externamente visíveis destas partes (módulos com interfaces, unidades de hardware, objetos), e os relacionamentos e restrições entre elas. A outra definição de arquitetura de software é a de SHAW e GARLAN (1996). Nela os autores afirmam que uma arquitetura de software envolve a descrição de elementos dos quais os sistemas serão construídos, interações entre esses elementos, padrões que guiam suas composições e restrições sobre estes padrões. A arquitetura de um software define em termos computacionais quais são seus componentes e como ocorre a interação entre eles. Componentes neste contexto podem ser bancos de dados, servidores, clientes, filtros, dentre outros, e a interação entre eles pode ocorrer através de chamadas de procedimentos, acesso a variáveis, uso de protocolos para acesso a clientes e servidores, bancos de dados, e eventos quaisquer.

Tanto na definição de D'SOUZA e WILLS (1999) quanto na de SHAW e GARLAN (1996), observa-se os conceitos de artefatos (módulos de software, hardware e componentes) e de relacionamento entre eles. Para este relacionamento é necessário ter uma estrutura, suportada na forma de padrões que viabilizam a arquitetura do software. Esta estrutura deve atender a um conjunto de restrições determinadas pela natureza da aplicação para a qual o software está sendo construído.

Existem diferentes formas de representar arquiteturas de software, dentre elas: diagramas de componentes e conectores, especificações formais, linguagens de descrição arquitetural, frameworks, notações específicas de um estilo arquitetural e os padrões arquiteturais. No escopo deste trabalho, será dado maior ênfase aos estilos arquiteturais e as linguagens de descrição arquitetural.

A estrutura dos elementos que envolvem uma arquitetura vai de alguma forma expressar o(s) estilo(s) arquitetural(ais) empregado(s) na aplicação. Existem na literatura estilos arquiteturais considerados genéricos, os quais podem ser utilizados por qualquer domínio de aplicação (e.g. camadas, *pipe and filter* (SHAW e GARLAN, 1996) (SHAW e CLEMENTS, 1996)), os orientados a domínio, que possuem uma forte relação com as restrições provenientes de aplicações do ramo (EMMERICH, 2001) e ainda os orientados ao DBC, os quais apóiam o processo de construção de componentes numa dada arquitetura (HERZUM e SIMS, 2000).

É consenso na literatura de Arquitetura de Software a inexistência de um conjunto de padrões gráficos para representar estas arquiteturas (SHAW e GARLAN, 1996). Aplicações OO geralmente utilizam diagramas de pacotes da UML para representar sua arquitetura. No entanto, tais diagramas têm uma semântica fraca para representar a forma como ocorrem as interações entre os elementos que compõem um pacote e, sobretudo, o uso de algum estilo arquitetural adotado para a aplicação. Por outro lado, observa-se que algumas arquiteturas de software utilizam simbologia própria para representar seus elementos, suas interações e o estilo arquitetural adotado. Geralmente esta simbologia também apresenta problemas em representar a semântica envolvida na arquitetura de software. Como resultado, as representações gráficas de arquiteturas de software pouco têm ajudado À equipe de desenvolvimento na etapa subseqüente, a qual envolve a implementação da aplicação, com diretrizes arquiteturais que as norteiam.

Se por um lado a representação gráfica é uma área de pesquisa em Arquitetura de Software que ainda deve ser devidamente estudada, por outro existe uma área onde propostas para formalizar adequadamente uma arquitetura (ADL – Architecture Description Language) têm sido apresentadas (SHAW e GARLAN, 1996) (DI NITTO e ROSENBLUM, 1999). As ADL's expressam mais fortemente um formalismo existente numa arquitetura de software. Tais ADL's geralmente estão voltadas a domínios de problemas específicos e de alguma forma privilegiam o uso de um ou outro estilo arquitetural.

Tanto estilos arquiteturais quanto linguagens de descrição arquitetural são discutidos a seguir, com ênfase àqueles voltados ao contexto de Desenvolvimento Baseado em Componentes (DBC).

3.2 Estilos Arquiteturais

Para o desenvolvimento da arquitetura de um software, podem ser utilizadas estruturas padrões para a criação de artefatos, as quais determinam a forma como tais artefatos irão se relacionar, o tipo de comunicação entre eles e os resultados (saídas) esperados de cada artefato de uma arquitetura. Com o intuito de apresentar soluções recorrentes para o projeto de arquiteturas, foram propostos alguns estilos (GARLAN e CLEMENTS, 1996) (SHAW e GARLAN, 1996), os quais têm sido utilizados na comunidade de Engenharia de Software.

Normalmente existe uma certa confusão para diferenciar padrões de estilos arquiteturais. Realmente não existe uma diferença muito clara. No entanto, os padrões arquiteturais apresentam soluções mais concretas em termos de tipos de classes e métodos que devem ser contemplados na aplicação para aplicar um dado estilo. Já os estilos arquiteturais são mais abstratos.

Um estilo arquitetural define um conjunto de regras de projeto que identificam tipos de componentes e seus conectores, podendo ser usados para compor uma família de sistemas e subsistemas, bem como as restrições existentes para a composição destes componentes (SHAW e GARLAN, 1996). Exemplos de estilos arquiteturais conhecidos são *Pipes and Filters*, Camadas, Baseado em Eventos e Organização Orientada a Objetos. Um conjunto mais completo de estilos arquiteturais pode ser obtido em (GARLAN e CLEMENTS, 1996) (SHAW e GARLAN, 1996).

Em geral, os estilos arquiteturais compreendem um vocabulário comum, compreendendo os seguintes elementos: tipos de componentes, tipos de conectores, estruturas de controle, comunicação de dados, interação entre dados e controle, regras e restrições. A Figura 6 apresenta uma descrição resumida de um estilo arquitetural.

Em função da quantidade e da variedade de propósitos dos estilos arquiteturais existentes, SHAW e CLEMENTS (1996) propuseram uma classificação de estilos de acordo com cinco categorias, respondendo às seguintes questões:

- quais tipos de componentes e conectores são usados no estilo;
- como o controle é compartilhado, alocado e transferido entre os componentes;
- como os dados são comunicados através do sistema;
- como os dados e controle interagem,
- qual tipo de raciocínio é compatível com o estilo.

Estilo: Pipe and Filter
Componentes: pipes ; **Conectores**: filters
Estrutura de Controle/Comunicação de dados ,Interação dados/controle:
- Cada Componente recebe conjuntos de dados (*streams*) como entrada e gera uma seqüência de dados na saída.
- Filtros aplicam transformações locais aos conjuntos de dados e realizam seu processamento de maneira incremental, de forma que as saídas começam a ser geradas antes que toda a entrada tenha sido consumida.
Regras e Restrições:
–Filtros são unidades Independentes
–Filtros não conhecem a identidade dos outros filtros
–A corretude da saída de um filtro não depende da ordem dos filtros.

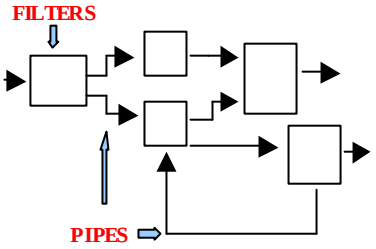

Figura 6: Descrição de um Estilo Arquitetural - *Pipes and Filters*

Como resultado, os autores propõem uma tabela com 25 estilos arquiteturais, devidamente classificados pelo propósito principal. A Tabela 1 mostra uma versão reduzida da tabela original proposta em (SHAW e CLEMENTS, 1996), descrevendo apenas 2 dos 25 estilos, os quais pertencem a grupos diferentes.

Os estilos arquiteturais citados até o momento têm o objetivo de apoiar a construção de arquiteturas de software, independente do domínio onde estes serão adotados. No entanto, existem na literatura, propostas de estilos arquiteturais orientados a domínio, os quais buscam atender a particularidades que não podem ser atendidas em estilos

arquiteturais genéricos, como aqueles discutidos em (GARLAN e CLEMENTS, 1996) (SHAW e GARLAN, 1996). Exemplos de estilos arquiteturais orientados a domínio podem ser obtidos em (EMMERICH ET AL, 2001) (MORISAWA e TORII, 2001) (FIELDING, 2000).

Tabela 1: Estilos Arquiteturais – Exemplos de (SHAW e CLEMENTS, 1996)

Estilo	Partes Constituintes		Questões de Controle			Questões de Dados				Interação Dado/Controle		Tipo de Raciocínio
	Componentes	Conectores	Topologia	Sincronicidade	Tempo de ligação	Topologia	Continuidade	Modo	Tempo de ligação	Forma isomórfica	Direção de Fluxo	
Estilos para Processo Interativo: padrões de comunicação entre processos independentes e usualmente concorrentes												
Baseado em Eventos	processos	Invocação implícita	Arbitrária	Assíncrono / oportunística	Tempo de Invocação e execução	Arbitraria	Esporadica	Transmissão	Tempo de invocação e execução	não	n/a	Não determinístico
Estilos de chamada e retorno: estilos que enfatizam a ordem de computação, usualmente com uma única thread de controle												
Tipos Abstratos de Dados	gerentes	Chamadas de procedimentos estáticos	Arbitraria	Sequencial	Tempo de escrita e compilação	Arbitraria	Esporádica	entrega	Escrita, compilação e execução	sim	mesma	Hierárquico

EMMERICH *et al.* (2001) propuseram um estilo arquitetural para a integração de aplicações empresariais, o qual deram o nome de TIGRA, originado do projeto *Trading room InteGRation Architecture*. O estilo TIGRA é baseado na separação de representação de dados, usando uma linguagem XML específica do domínio, do transporte desses dados com um *middleware* apropriado.

Outros exemplos são os estilos arquiteturais de linha de produto propostos por (MORISAWA e TORII, 2001), específicos para sistemas de processamento distribuído. Os autores propuseram um conjunto de estilos arquiteturais, os quais são classificados de acordo com o tipo de processamento de clientes e servidores (síncrono e assíncrono), tipos de dados (transação, consulta e notificação) e características de distribuição de dados e processamento. A contribuição deste trabalho é, além da identificação e detalhamentos dos estilos arquiteturais apresentados, a proposta de seleção de um estilo arquitetural para um dado tipo de sistema.

Por último, destaca-se os estilos arquiteturais orientados ao domínio de arquiteturas de software baseadas em redes (FIELDING, 2000), os quais estendem alguns estilos arquiteturais genéricos como *Pipe and Filter*, Cliente/Servidor, Camadas, Baseados em Eventos, para atender as particularidades do domínio. Para cada estilo arquitetural, o autor apresenta uma avaliação segundo alguns critérios de qualidade (e.g.

eficiência, simplicidade, reusabilidade, visibilidade, portabilidade). Como resultado, os autores propuseram um estilo arquitetural denominado REST (**Re**presentational **S**tate **T**ransfer). REST é um estilo arquitetural híbrido específico para sistemas de hipermídia distribuído, derivado do estilo arquitetural baseado em redes, combinado com restrições adicionais que definem uma interface de conexão hipermídia.

3.2.1 Estilos Arquiteturais e DBC

Na Seção 2.2, foi apresentada uma sistemática comum aos métodos de DBC. Nesta sistemática definiu-se uma forma macro de como é concebida uma arquitetura de aplicação baseada em componentes. Nesta Seção, veremos alguns estilos arquiteturais que podem ser aplicados para apoiar tanto o processo de detecção de componentes, a partir dos artefatos gerados durante a execução de um método de DBC, quanto à forma como estes componentes poderão se comunicar.

Existem dois tipos de estilos arquiteturais para apoio à geração de componentes de negócio, os quais foram documentados em (HERZUM e SIMS, 2000) e explorados em (TEIXEIRA, 2003): estilos baseados em tipos do negócio e estilos baseados em instância. Estes estilos podem ser detectados em métodos de DBC (D'SOUZA e WILLIS, 1999).

3.2.1.1 Estilo Arquitetural Baseado em Tipos

O estilo arquitetural baseado em tipos, o qual é empregado para geração de componentes no método UML *Components*, é também conhecido como baseado em serviços. Este estilo tem por objetivo encapsular todas as instâncias de um tipo de negócio, fornecendo acesso às suas informações através de suas interfaces. Este estilo arquitetural captura os tipos do negócio de um domínio ou aplicação, para que estes sejam representados através de componentes dentro de uma arquitetura, fornecendo interfaces para acesso às suas informações. Os componentes deste estilo arquitetural são chamados de gerentes de instância. Para que se tenha acesso numa aplicação a uma determinada instância de um tipo de negócio, o estilo arquitetural utiliza o conceito de chaves técnicas, representadas por valores alfanuméricos.

TEIXEIRA (2003), propõe-se uma forma de geração e conexão de componentes de negócio, gerados a partir de diferentes estilos arquiteturais para componentes, no contexto de uma infra-estrutura de reutilização baseada em modelos de domínio,

denominada Odyssey - SDE. No que se refere o estilo arquitetural baseado em tipos, Teixeira propõe que estes sejam gerados a partir da detecção dos tipos de negócio através de um diagrama de tipos (*business type diagram*) pré-existente.

Para que esta detecção aconteça de forma eficaz, algumas operações devem nortear o processo de obtenção dos componentes de negócio. Uma delas se refere ao mapeamento de tipos de negócio para componentes de negócio. Neste caso, o resultado, ou seja, os gerentes de tipos gerados, vão depender da forma como os tipos do negócio foram modelados no diagrama de tipos. Tal constatação se deve ao fato de que os tipos de um negócio, num diagrama, estão de alguma forma relacionados e, portanto, estes relacionamentos podem interferir nos componentes de negócio gerados, ou seja, componentes de negócio podem representar agrupamentos de tipos de negócio. Neste sentido, Teixeira propõe que componentes sejam agrupados de acordo com suas regras de herança (e.g. agregação, composição) modeladas no diagrama de tipos de negócio. No entanto, caberá ao arquiteto do software aceitar ou não a sugestão de agrupamento dos tipos.

Os gerentes de instância gerados devem armazenar e controlar seus dados, exigindo que seja feita uma documentação de sua estrutura de dados. Esta estrutura de dados vai englobar, dentre outros atributos, a sua chave técnica e representação de atributos de supertipos e subtipos.

O próximo passo é a criação das interfaces que acessam e manipulam os atributos dos gerentes de instância criados anteriormente. Cada componente gerente de instâncias terá associado a ele uma interface e algumas operações básicas são associadas a ela (e.g. create, delete, getId, getAll, get, set e find).

Um exemplo de criação de componentes de negócio utilizando o estilo arquitetural baseado em tipo no Odyssey pode ser visualizado na Figura 7. Nesta figura podem ser observadas situações distintas: primeiro o processo de criação dos componentes gerentes de instância, na qual observa-se que para os tipos pessoa e aluno, que representam uma relação de herança, foi gerado um único gerente de instância, denominado PessoaAlunoManager. Este gerente apresenta duas interfaces: IAlunoMgr e IPessoaMgr. Para cada interface de IPessoaMgr e IAlunoMgr, foram criadas operações compatíveis com a relação supertipo e subtipo.

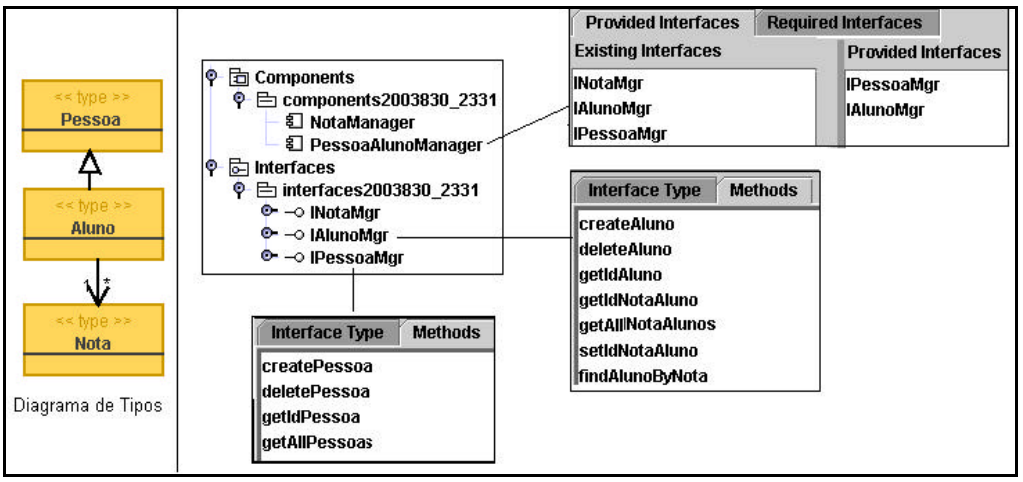

Figura 7: Representação da criação de gerentes de instâncias no Odyssey.

Os componentes gerentes de instâncias, como já diz o próprio nome, apenas armazenam as instâncias, que nada mais são do que elementos estáticos de um modelo. No entanto, é necessário representar também os aspectos dinâmicos ligados aos componentes gerentes de instâncias. Com o intuito de que os aspectos dinâmicos de uma aplicação não fiquem espalhados entre diferentes componentes, é proposto o componente de processo, que tem por objetivo oferecer interfaces com operações que estão diretamente relacionadas à execução dos processos do negócio. Os componentes de processo têm, portanto, o objetivo de apoiar os componentes gerentes de instância na execução dos processos do negócio, e para isso, é necessário que ele identifique as operações que apóiam o processo, bem como a seqüência destas operações.

Em TEIXEIRA (2003) os componentes de processo são criados a partir da realização de casos de uso, uma vez que estes artefatos permitem o detalhamento dos passos que descrevem as ações envolvidas na sua execução via software. Estes últimos não dão apoio a construção de seqüência de ações e não são tão descritivos como os casos de uso. Para tornar mais padronizada a linguagem expressa numa seqüência de ações, foram adotadas algumas operações (e.g. Create, Delete, Update, Connect, Disconnect, Enter, Show e Custom), sendo que as cinco primeiras visam especificar os passos dos casos de uso e as duas últimas para ações de entrada e saída de dados.

A Figura 8 mostra a seqüência de passos envolvidos na realização do caso de uso "consulta nota" e o resultado obtido ao aplicar o estilo arquitetural baseado em tipo, ou

seja, um diagrama de interação a partir da realização de um caso de uso vinculado ao componente PessoaAlunoMgr. Estas telas foram extraídas a partir da execução do estilo arquitetural no Odyssey SDE.

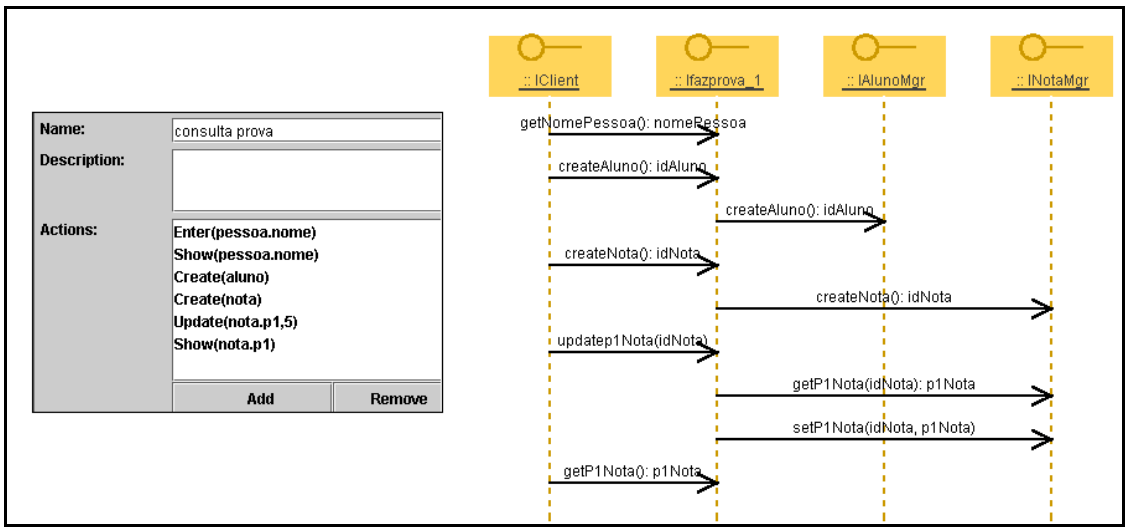

Figura 8: Janela de descrição de passos e casos de uso e Diagrama de Interação resultante da aplicação do estilo arquitetural

3.2.1.2 Estilo Arquitetural Baseado em Instâncias

Diferentemente do estilo baseado em tipos que, para se fazer acesso a uma de suas instâncias, é necessário fazê-lo através do gerente de instâncias, aqui as instâncias são gerenciadas e controladas diretamente. Neste estilo, existe o conceito de dois tipos de componentes: entidade e de coleção. Componentes entidades são aqueles que formam as instâncias e os componentes coleção servem como "molde" para a criação de componentes entidade, oferecendo serviços de busca de outros componentes pertencentes a mesma coleção.

Assim como o estilo baseado em tipos, em TEIXEIRA (2003), o estilo baseado em instância é gerado a partir dos tipos do negócio, sendo que as instâncias são representadas através de componentes de entidade e os tipos propriamente ditos através dos componentes coleção.

No estilo baseado em instância, o mapeamento de tipo para componente é um-para-um, ou seja, cada tipo terá o seu componente coleção e pelo menos um componente entidade gerado. Neste sentido, as relações existentes no diagrama de tipos de negócio

não vão implicar numa política de agrupamento de tipos para os componentes gerados, como ocorria no estilo arquitetural baseado em tipos.

O processo de geração dos componentes é bastante simples. No que tange ao componente coleção, cada um receberá uma interface denominada I<tipo do negócio>Collection. Através desta interface, operações para fabricação e busca de componentes de entidade serão disponibilizadas.

Os componentes entidades são gerados a partir dos atributos e navegações obtidas pelo diagrama de tipos de negócio. Cada interface criada possui o formato I<TipoNegócio> e possui operações para manipulação de uma instância (e.g. gets e sets).

Os componentes de entidade não possuem operação *create*, uma vez que esta fica a cargo do componente coleção, como uma forma de controlar e identificar as instâncias criadas a partir do seu tipo. Por outro lado, cabe ao componente entidade a exclusão de sua instância e informar tal operação ao componente coleção do qual foi gerado.

A Figura 9 mostra o resultado obtido do processo de geração dos componentes de entidade e coleção na infra-estrutura de reutilização Odyssey, a partir do mesmo diagrama de tipos das figuras vinculadas ao estilo baseado em instâncias. A Figura 10 mostra o modelo de componentes gerado a partir daqueles componentes criados, conforme mostra a Figura 9.

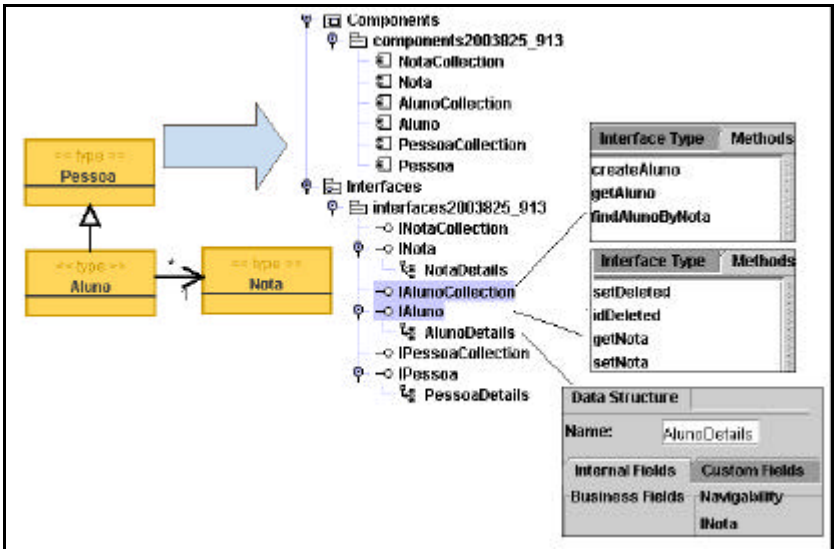

Figura 9: Componentes gerados a partir do estilo baseado em instância

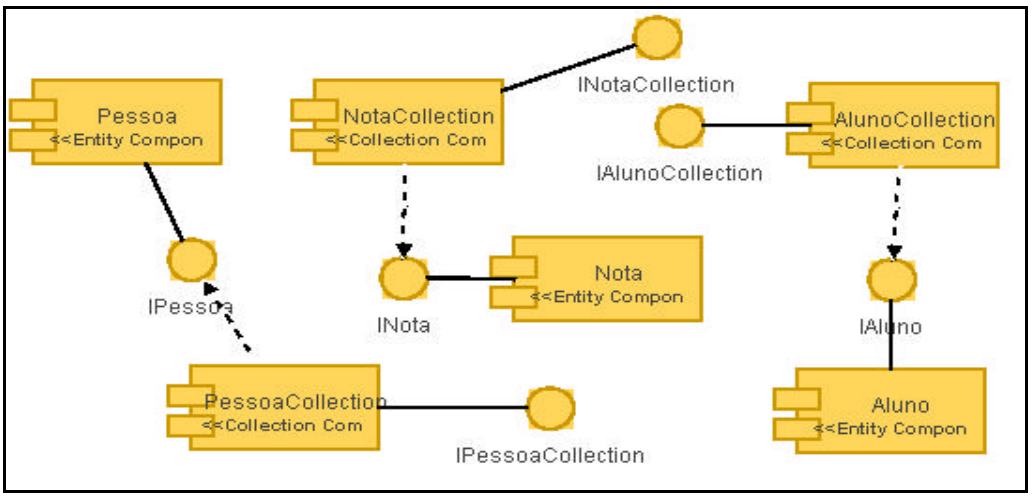

Figura 10: Diagrama de Componentes gerados com o uso do estilo arquitetural baseado em instância

3.2.2 Considerações sobre Estilos Arquiteturais

O estudo de alguns dos principais estilos arquiteturais genéricos existentes (e.g. camadas, *pipe and filter*, baseado em eventos) e daqueles existentes para o contexto de DBC possibilitou identificar algumas constatações com relação ao suporte que estilos arquiteturais provêem para a construção de uma arquitetura baseada em componentes.

Primeiro, é que os estilos arquiteturais para DBC dão suporte somente à geração de componentes de negócio de uma aplicação, ficando os demais componentes (utilitários e de infra-estrutura), desprovidos de tal apoio. Além disso, não se justifica até o momento em que se encontra o estado da arte em métodos de DBC, a existência de outros estilos arquiteturais do que o baseado em tipos e baseado em instâncias, uma vez que estes já capturam as informações mais comuns para a geração dos componentes a partir de modelos (modelo de tipos e casos de uso) que representam efetivamente o negócio de uma aplicação.

Segundo, é que não se tem uma idéia clara se o processo de geração de componentes de negócio poderia ser aplicado para componentes de infra-estrutura e utilitários, pois suas características não se encontram claramente num ou outro diagrama construído durante o processo de modelagem de uma aplicação baseada em componentes. Conforme descrito na Seção 2.2, componentes utilitários provêem serviços genéricos, independentes do domínio da aplicação, e os componentes de infra-estrutura estão diretamente relacionados a requisitos não-funcionais de uma aplicação. Neste sentido,

pesquisas a respeito de estilos arquiteturais para geração de componentes destas categorias poderiam vir a ser interessantes.

O terceiro aspecto é referente ao grau de abstração entre os diferentes componentes previstos numa arquitetura. Segundo TEIXEIRA (2003), uma arquitetura de componentes pode ser organizada na forma de camadas, ou seja, claramente um estilo arquitetural em camadas poderia ser aplicado para tratar das questões de comunicação entre componentes de diferentes níveis de abstração (camadas). Embora o estilo arquitetural camadas seja sugerido, outros estilos arquiteturais genéricos poderiam ser analisados para aplicar-se numa arquitetura de componentes, em especial para tratar das questões de comunicação entre componentes que pertençam a um mesmo nível de abstração (e.g. componente de negócio com um outro proveniente de um sistema legado) ou entre componentes de diferentes níveis de abstração (e.g. componente de negócio com um componente de infra-estrutura que trate de persistência de dados). Por outro lado, existem alguns padrões arquiteturais, como os propostos em SCHMIDT (2000), que poderiam dar apoio a este processo de comunicação de componentes de diferentes camadas de uma arquitetura. Além desse suporte, estes padrões poderiam efetivamente participar do processo de geração de componentes de infra-estrutura, que visam atender a requisitos não funcionais como persistência entre objetos concorrentes, tratamento de exceções e comunicação síncrona e assíncrona, através de padrões como Component Configurator, WrapperFaçade, Reactor e Proactor, e Half-Sync/Half-Async.

3.3 Linguagens de Descrição Arquitetural e DBC

Linguagens de Descrição Arquitetural (*Architecture Description Languages* - ADL) são uma outra forma de representação de arquiteturas de software que pode ser empregada em consonância com outras formas de representação, incluindo estilos arquiteturais.

Para dar apoio ao desenvolvimento de uma arquitetura de software, são necessárias notações formais para modelagem e ferramentas que operam com especificações arquiteturais. Com este intuito, ADLs e ferramentas associadas a elas têm sido propostas.

Segundo (SHAW e GARLAN, 1996), uma ADL compreende um conjunto de notações visando um modelo arquitetural de projetos com alto nível de abstração,

desenvolvidos em larga escala, os quais podem sofrer adaptações para implementações específicas. MEDVIDOVIC e TAYLOR (2000) afirmam que uma ADL tem maior foco nas estruturas de mais alto nível para uma aplicação do que nos seus detalhes de implementação.

Existe um número considerável de ADLs propostas para modelagem de arquiteturas, algumas delas com propósitos gerais, outras voltadas a domínios específicos. Algumas ADLs servem somente para descrever a arquitetura de um software e já possuem seu código fonte gerado. Outras possuem ferramentas para detecção de problemas que poderiam ocorrer em tempo de execução e/ou simulam a execução da aplicação. Em (MEDVIDOVIC e TAYLOR, 2000) e (FIELDING, 2000), podem ser encontradas breves descrições de algumas ADLs bastante conhecidas, sendo que algumas delas estão resumidamente definidas na Tabela 2.

Tabela 2: Exemplos de ADL's genéricas

ADL	**Aesop**	**C2**	**Darwin**	**Rapide**	**Unicon**	**Wright**
FOCO	Especificação de arquiteturas em estilos específicos	Arquiteturas de sistemas distribuídos e dinâmicos	Arquiteturas de sistemas distribuídos	Modelagem e simulação de comportamentos dinâmicos descritos numa arquitetura	Geração de código para interconexão de componentes usando protocolos de interação	Modelagem e análise de comportamento dinâmico de sistemas concorrentes

Uma ADL, usualmente, é composta de três principais elementos: componentes, conectores e configurações. Os componentes que representam unidades de computação quaisquer, conectores que são a forma de interação entre unidades de computação e as configurações determinam a maneira como componentes e conectores serão compostos para definir uma arquitetura específica.

Em (MEDVIDOVIC e TAYLOR, 2000), os autores propõem um framework de classificação e comparação de ADL, detalhando possíveis características que envolvem cada um dos principais elementos de uma ADL.

Ao elemento componente estão vinculados: interfaces, que definem os pontos de interação do componente em relação aos demais artefatos externos a ele; tipos, que são as abstrações que encapsulam funcionalidades e que podem ser instanciados várias vezes numa arquitetura de software; semântica, ou seja, comportamento do componente; restrições, que são propriedades de um sistema ou de suas partes que tornam a sua

execução inaceitável; evolução, ou seja, as modificações que ocorrem sobre as interfaces, semântica, tipos, etc, de um componente, e propriedades não funcionais, ou seja, segurança, portabilidade, etc.

O elemento conector possui características semelhantes ao componente, porém com ênfase um pouco distinta: interface, é um conjunto de pontos de interação entre o conector e os componentes e outros conectores relacionados a ele; tipos, são abstrações que encapsulam decisões de comunicação, coordenação e mediação com componentes; semântica, análogo aos tipos de componentes, aqui a semântica define o comportamento de conectores; restrições, asseguram adesão de conectores aos protocolos de interação e dependência intraconectores; evolução, análogo a evolução de componentes, evolução de conectores define as possíveis modificações que um conector pode sofrer e, propriedades não funcionais, que representam requisitos para a implementação correta de conectores.

Por último, as características vinculadas ao elemento configurações são: entendimento da especificação, que visa apoiar diferentes arquitetos de software no entendimento do projeto e as abstrações de alto nível especificadas através de uma ADL; composição, que consiste de um mecanismo onde uma arquitetura pode ser descrita em diferentes níveis de detalhe (explicitamente detalhados ou encapsulados num único componente ou conector); refinamento e rastreabilidade, permitir refinamento correto e consistente de arquiteturas para sistemas executáveis e rastreabilidade entre os artefatos gerados através do refinamento da arquitetura; heterogeneidade, fornecer facilidades para especificação e o desenvolvimento de arquiteturas de software com componentes e conectores heterogêneos (abertos); escalabilidade, dar suporte ao aumento de complexidade e tamanho de um sistema; evolução, suporte a evolução de componentes e conectores que pertencem a famílias de sistemas que estão em constante aprimoramento; dinamismo, suporte a modificações que devem ocorrer enquanto o sistema está executando; restrições, que podem existir em função de modificações que componentes e conectores podem ter sofrido, e propriedades não funcionais, que, em nível de configuração, podem ser análise de desempenho, mapeamento dos blocos de construção arquitetural para processadores, seleção de componentes e conectores apropriados.

Em (MEDVIDOVIC e TAYLOR, 2000), são apresentadas tabelas que apresentam as características referentes à modelagem de componentes, de conectores e configurações

para as ADLs Aesop, C2, Darwin, MetaH, Rapide, SADL, UniCon, Weaves e Wright. Além dessa comparação, os autores avaliam o suporte oferecido através de ferramentas para apoio à construção de arquiteturas de software com as ADLs acima citadas, e o resultado obtido é que existem ainda poucos ambientes com este tipo de suporte ou, aquele oferecido para algumas ADLs se restringe a alguns aspectos (e.g. visões, refinamento, geração de implementação).

Apesar de, em (MEDVIDOVIC e TAYLOR, 2000), ter sido identificado um conjunto de características comuns entre ADLs, se observa na literatura que existe uma falta de consenso entre o que é uma ADL e quais aspectos de uma arquitetura deveriam ser modelados por uma ADL (SHAW e GARLAN, 1996). Algumas das linguagens de descrição arquitetural existentes têm propósitos bem distintos como mostra a Tabela 2. No entanto, uma característica comum é que as ADLs possuem alguma sintaxe e semântica para representação de componentes, interconexão de componentes através de conectores e configurações que permeiam a arquitetura especificada.

Durante este estudo, também foi possível observar que existe uma falta de clareza com relação à distinção do que venha a ser uma ADL e do que é um estilo arquitetural. Observa-se que uma ADL está representando formalmente uma arquitetura que tem sua estrutura baseada em um ou mais estilos arquiteturais, mas, existem alguns casos, como o C2, que é considerado tanto um estilo arquitetural, como tratado em (DI NITTO e ROSENBLUM, 2000), quanto uma ADL, como em (MEDVIDOVIC e TAYLOR, 2000). Este uso combinado de conceitos se deve ao fato de que as ADLs geralmente têm uma forte tendência em formalizar uma arquitetura voltada a algum estilo arquitetural, como é o caso de C2. Neste sentido, são encontradas na literatura algumas pesquisas na tentativa de tornar as ADLs mais flexíveis no que diz respeito ao suporte a estilos arquiteturais.

Em (DI NITTO e ROSENBLUM, 2000), os autores propuseram um estudo de caso no qual, para um mesmo sistema, foram utilizados dois estilos arquiteturais, C2 e JEDI, separadamente, os quais foram especificados usando diferentes ADLs: ARMANI, Rapide, Darwin, Wright e Aesop. A Figura 11 mostra trechos de uma descrição arquitetural especificada em duas diferentes ADLs, para um sistema estruturado pelo estilo arquitetural C2.

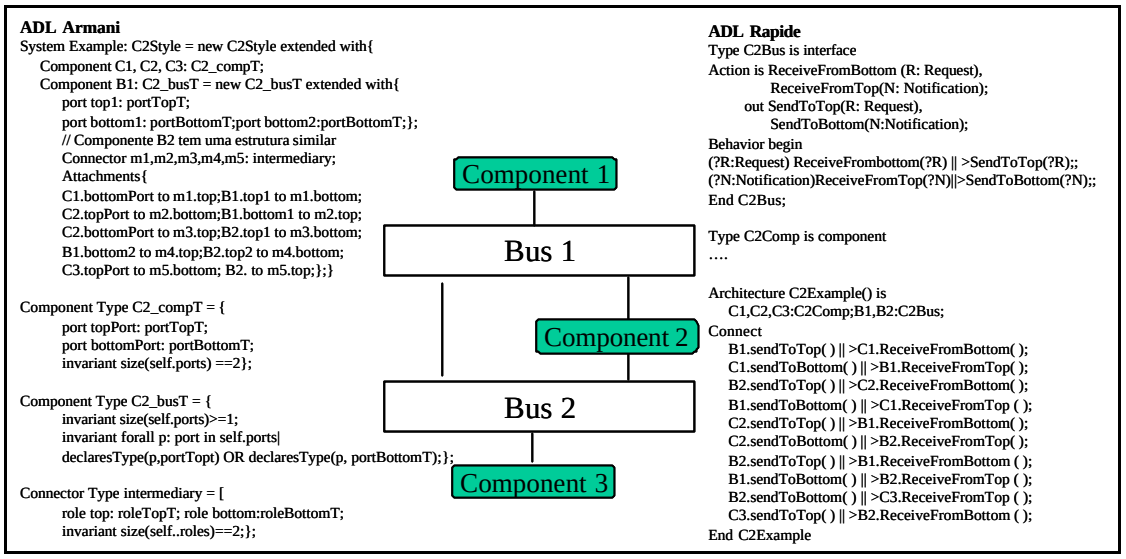

Figura 11: Sistema utilizando estilo C2 e especificado em duas ADLs – Rapide e Armani

Uma outra pesquisa que tenta diminuir o grau de dependência de uma ADL a estilos arquiteturais é através de uma ADL denominada ACME. A ACME busca dar suporte ao mapeamento de especificações arquiteturais de uma ADL para outra, visando a integração entre ADLs. Em (MEDVIDOVIC e TAYLOR, 2000), os autores afirmam que ACME não pode ser considerada uma ADL, uma vez que ela parte de uma especificação arquitetural para gerar uma outra, ou seja, não gera sua própria especificação arquitetural. No entanto, independente de ACME ser considerada ou não uma ADL, não se tira o mérito do suporte a integração e mapeamento entre ADLs, que é uma grande contribuição na área de arquiteturas de software, uma vez que um ambiente de desenvolvimento de software está em constante evolução, e conseqüentemente o suporte arquitetural deve estar preparado para estas modificações.

A cada ADL proposta, um conjunto de notações é definido e nem sempre estas ADLs têm suporte para ferramentas de apoio a utilização destas especificações. Observa-se também que não existe uma preocupação em integrar os modelos previamente gerados para o sistema (e.g. diagramas de casos de uso, classes, seqüência, colaboração), os quais em sua grande maioria, seguem a notação da UML (Unified Modeling Language). Atentos a esta lacuna gerada no processo de desenvolvimento de software é que (MEDVIDOVIC e TAYLOR, 2000), propuseram algumas diretrizes para uso da notação da UML para especificação de ADLs. Os autores utilizaram duas estratégias: 1) utilizar a

UML tal como foi proposta e 2) incorporar a UML características utilizadas nas ADLs através de extensões. Para a primeira estratégia, o autor utilizou diagramas de classes para modelar conectores, classes, interfaces e suas respectivas colaborações. A outra estratégia, a qual, segundo os autores, permite uma especificação mais compatível com aquela feita através da ADL, sugere o uso de estereótipos para representação de mensagens, componentes e conectores da ADL, explorando as restrições destes estereótipos através da OCL (Object Constraint Language) já existente na UML. Os autores efetuaram experiências com o uso das ADLs C2, Wright, Rapide, as quais receberam tratamentos distintos com relação aos estereótipos gerados. Neste sentido esta abordagem não pode ser comparada a proposta da ADL ACME, uma vez que esta última se propõe a dar apoio ao mapeamento de uma ADL para outra, e a proposta de Medvidovic *et al*. não pretende dar suporte, através da UML e extensões geradas, ao mapeamento de modelos arquiteturais para diferentes ADL, e sim, que para uma dada aplicação, possa haver suporte da UML para especificação de sua arquitetura.

3.3.1 Linguagens de Descrição Arquiteturais para DBC

As linguagens de descrições arquiteturais discutidas até agora poderiam ser adotadas em quaisquer aplicações, independente do paradigma de desenvolvimento de software adotado. O conceito de componente adotado nestas ADLs era de unidades de computação, ou seja, neste contexto representa tanto uma aplicação de software, um sistema legado como um processador. No entanto, existem propostas de ADLs voltadas a especificação arquitetural de aplicações baseadas em componentes. Para estas aplicações, um componente tem por objetivo representar um artefato da aplicação, autônomo, que possui interfaces definidas.

Foram encontradas na literatura três propostas de ADLs voltadas a especificação de arquiteturas para DBC: ABC/ADL (CHEN *et al*., 2002), uma linguagem de descrição arquitetural para o método de DBC ABC, a CDL (Component Description Language) (MENCL,1998), voltada à descrição de componentes e aplicações construídas a partir dos mesmos, e a xADL (DASHOFY, HOEK E TAYLOR, 2002) as quais serão brevemente discutidas nas seções que se seguem.

3.3.1.1 ABC/ADL

O modelo de componentes ABC é dividido em duas partes: interfaces externas que descrevem funções fornecidas e requeridas por um componente, e especificações internas que descreve restrições sobre a estrutura interior dos componentes, modelos semânticos, etc. A ADL para o modelo ABC consiste de três camadas: meta-camada que fornece abstrações para definir templates e estilos arquiteturais; uma camada de definição que fornece um construtor para definir componentes, conectores e arquiteturas genéricas e todas as definições fornecidas pela meta-camada, e camada de instância, que prove abstrações para definir interconexão de instâncias de componentes. Estas instâncias devem ser definidas usando o construtor da camada de definição. Um exemplo da ABC/ADL é apresentado nas Figuras 13,14 e 15. A Figura 12 apresenta a definição de um estilo arquitetural, segundo o modelo de componentes ABC. Neste trabalho, considera-se que o arquiteto de software pode definir seu próprio estilo arquitetural, atendendo as características estruturais esperadas para a aplicação que está desenvolvendo. Neste exemplo de estilo arquitetura1, são definidos dois componentes templates denominados *Blackboard* e *Client*, cada um com a especificação das interfaces requeridas e fornecidas, e a forma de conexão destes componentes entre si. A Figura 15 mostra um exemplo de definição de um componente baseado no estilo arquitetural Blackboard definido na ADL. Por último, a Figura 14 mostra parte da definição do sistema de agenda, o qual utiliza o componente *SchedulingManager* definido na Figura 15, bem como o estilo arquitetural da Figura 12.

```
Style BLACKBOARD_STYLE{
COMPONENT_TEMPLATE Blackboard{
    PROVIDE_PLAYER_TEMPLATE Entry {multiplicity=n};
   REQUEST_PLAYER_TEMPLATE Notify{multiplicity=n};}
COMPONENT_TEMPLATE Client{
   PROVIDE_PLAYER_TEMPLATE Notify {multiplicity=n};
   REQUEST_PLAYER_TEMPLATE Entry {multiplicity=1};}
// aqui não existe definição de conectores pois o estilo
//usa conectores padrão do modelo
CONNECTION{
   Client.Entry :: DEFAULT Connector.Callee
   DEFAULT.Connector.Caller::Blackboard.Entry}
CONNECTION{
   Blackboard.Notify::DEFAULT.Connector.Callee
      DEFAULT.Connector.Caller::Client.Notify}
}
```

Figura 12: Especificação de um estilo arquitetural na ABC/ADL

```
Component SchedulingManager is BLACKBOARD.Blackboard{
Interfaces{
    provide player SchedulingManager is BlackBoard.Entry{
        type-method{
            SchedulingManage findByPrimaryKey (Object id);
        }
    instance-method{….}
    }
   request player Agenda is BlackBoard.Notification { … …}}
   Attributes { …}
   Proprierties { …}
   Dependencies { …}
   SemanticDescription { …}
```

Figura 13: Especificação de um componente na ABC/ADL

3.3.1.2 Component Description Language (CDL)

CDL é uma proposta de linguagem de descrição arquitetural voltada para a descrição de componentes de software. Esta linguagem foi criada no contexto do projeto SOFA (**SOF**tware **A**ppliances). Uma aplicação baseada em SOFA, segundo (SOFA Project, 2003), deve ser composta por um conjunto de componentes que podem ser obtidos e alterados dinamicamente. A CDL proposta visa descrever componentes denominados DCUP (**D**ynamic **C**omponent **UP**dating).

DCUP permite componentes de uma aplicação serem alterados em tempo de execução, sem que outros componentes saibam das alterações. Para gerenciar as alterações, DCUP é composto por um componente gerente (CM – Component Manager) e um componente construtor (CB – Component Builder). O componente gerente é responsável pelas alterações e outros aspectos do ciclo de vida de um componente, fornecendo a outros componentes os serviços fornecidos pelo componente ao qual é gerente. O componente construtor está associado a uma versão concreta do componente, cuidando de sua inicialização, encerramento (*shutdown*) e armazenamento, e recuperação do estado de um componente antes e depois de suas alterações.

A CDL proposta para definição dos componentes DCUP tem sua sintaxe baseada na IDL - *Interface Definition Language* da OMG. Embora a CDL seja considerada uma extensão da IDL, está última não suporta a definição de componentes, como proposta pela CDL. Neste sentido, apesar da CDL ser considerada um subconjunto da IDL, arquivos gerados pela primeira não poderão ser reconhecidos pela segunda, em função da existência de definição de componentes.

```
Architecture DS_Architecture{
uses{
    Component agendas: Agenda [];
    Component schedulingManager: SchedulingManager;
    Connector agendaToSchedulingManager:
                        SecuredConnector[];
    Connector schedulingManagerToAgenda:
                        DefaultConnector[];
    Variable I: int;}
Config main{
    agendas[I].SchedulingManager connects
            agendaToSchedulingManager[I].Callee
    agendaToSchedulingManager[I].Caller connects
            schedulingManager.SchedulingManager
    schedulingManager.Agenda connects
            schedulingManagerToAgenda[I].Callee
    schedulingManagerToAgenda[I].Caller connects
            agenda[I].Agenda}
SemanticDescription{
    OCL{
        Self.timetable is Sequence of TimeSlice
        Invariants{
            Self.timetable→ForAll(t1,t2|t1<>t2 implies
                t1.starttime >=t2.endtime or
                t2.starttime>=t1.endtime)}
}
Component Dating_system is System{
    Structure {architecture DS_Architecture}}
```

Figura 14: Parte da Descrição de um sistema de agenda através da ABC/ADL

Esta CDL trabalha com elementos da IDL (tipos, constantes, interfaces e exceções) e componentes os quais consistem de *templates* (componente interface) e arquiteturas (especificação de tipo de componente concreto). Em (MENCL, 1998) é apresentada uma descrição detalhada de todos os elementos que existem para especificação de um componente na CDL, as restrições e variações de alguns desses elementos (e.g. binding), a sintaxe compatível e um exemplo de aplicação, baseada no modelo de componente DCUP, usando a sintaxe da CDL para sua descrição arquitetural.

A Figura 15 mostra um exemplo de arquitetura modelada na CDL. Na parte (a) da Figura 15 são apresentados as definições de interfaces dos componentes, bem como seus métodos associados as interfaces fornecidas e a definição das interfaces requeridas. Na parte (b), são definidos componentes templates, utilizando as interfaces definidas na parte (a) da mesma Figura. Na parte (c), é definida a hierarquia de componentes, os subcomponentes, ligações entre componentes, e especificação das versões utilizadas.

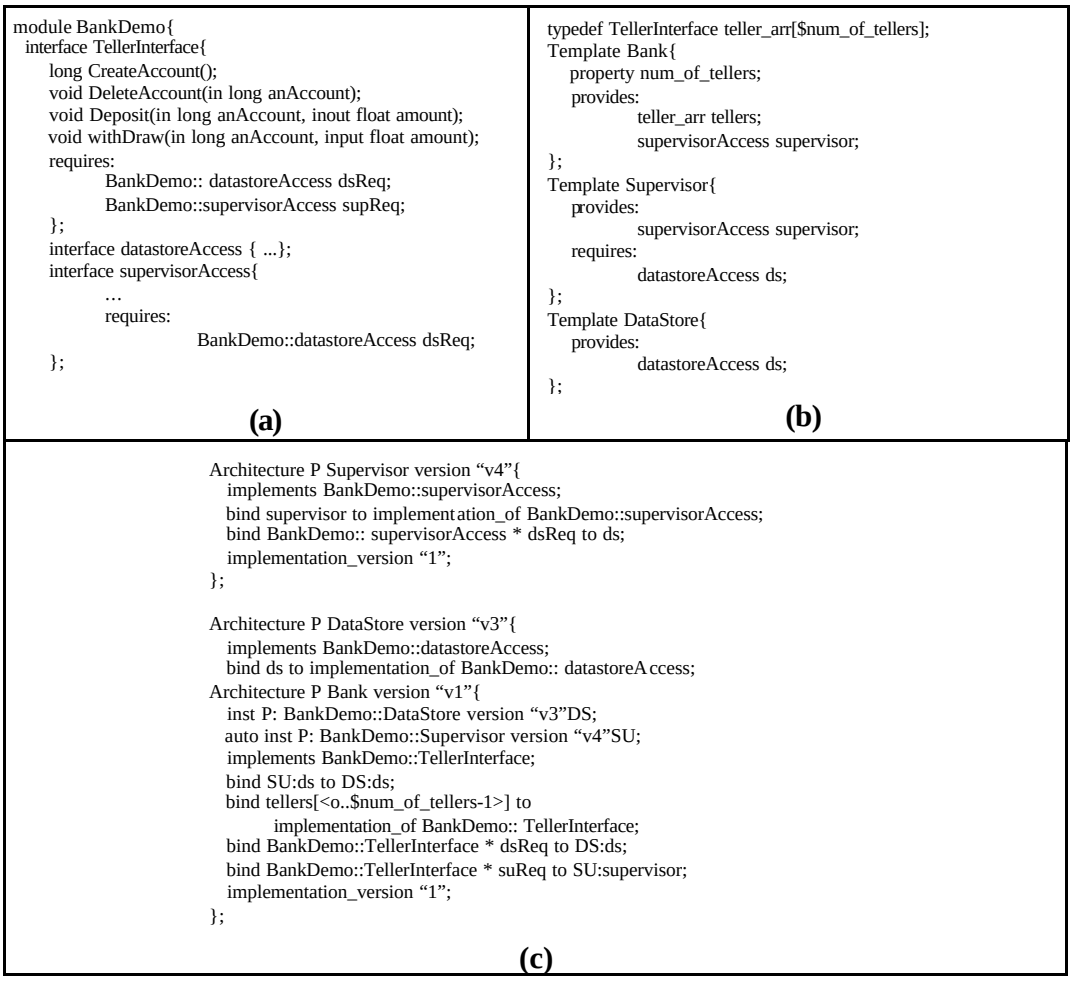

Figura 15: Trecho de uma descrição de sistema através da CDL

3.3.1.3 xADL

As ADLs apresentadas até o momento possuem um conjunto de notações particular, exigindo um esforço considerável da adaptação destas notações para uma arquitetura específica.

Neste sentido DASHOFY, HOEK e TAYLOR (2002) propuseram a xADL que utiliza os esquemas da XML (e**X**tended **M**arkup **L**anguage) para promover sua extensibilidade.

A XML permite a definição de construtores de uma linguagem e formas de estender e/ou modificar este construtores para adicionar e/ou modificar suas características. Arquivos no formato XML são documentos do tipo texto, compostos por um conjunto de *tags* (marcas) que definem o início e o final de segmentos do documento

XML. Estas *tags* na xADL visam representar os elementos arquiteturais a serem descritos a respeito da arquitetura de um sistema.

DASHOFY, HOEK e TAYLOR (2002) observaram que seria vantajoso o desenvolvimento de uma infra-estrutura para apoio a construção de ADLs. Através desta infra-estrutura, arquitetos podem criar, modificar e estender ADLs pela composição de características de modelagem.

Para dar suporte a criação de ADLs, a infra-estrutura utiliza uma meta linguagem modular. Esta meta linguagem define uma notação padrão extensível, onde novas características de uma ADL são encapsuladas em novos módulos reutilizáveis. São utilizados esquemas XML para definir um conjunto de construtores genéricos que são úteis na modelagem de arquiteturas de software. Estes construtores podem ser usados da forma como são propostos ou estendidos para suportar características adicionais de modelagem. A xADL define construtores básicos como componentes e conectores, mas não determina como eles devem se comportar ou como podem ser conectados uns aos outros. Estas características variam de uma ADL para outra e, portanto, devem ser especificadas através de extensões a esquemas XML da meta linguagem. Neste sentido, a xADL propõe construtores para descrição de arquiteturas genéricas, bem como características que podem ser usadas numa ADL derivada da xADL. Estes construtores são divididos em três grupos, conforme apresenta a Tabela 3.

Tabela 3: Esquema da xADL – traduzido de (DASHOFY, HOEK e TAYLOR 2002)

Propósito	**Esquemas XML (construtores)**	**Características**
Modelagem em Tempo de Projeto e Tempo de Execução	Instâncias	Componente executável, conector, ligações de instâncias; subarquiteturas, grupos gerais
	Estruturas e Tipos	Componentes de projeto, conectores, interfaces, ligações, subarquiteturas, grupos gerais
Mapeamento de Implementações	Implementação Abstrata	Espaço reservado para implementação de dados sobre componentes, conectores e interfaces.
	Implementação JAVA	Implementação concreta de dados para componentes Java, conectores e interfaces.
Gerência de evolução da Arquitetura e Suporte a Arquiteturas de Linha de Produto	Versões	Grafos de versões para componentes, conectores, interfaces e ligações
	Opções	Opções de componentes de projeto e tipos de conectores.
	Variantes	Variações de componente em nível de projeto, conectores, interfaces e ligações.
	Diferenças	Descrição de diferenças entre duas arquiteturas em tempo de projeto.

A Figura 16 mostra o relacionamento entre as ferramentas da infra-estrutura da xADL para construção de arquiteturas com foco em linha de produto.

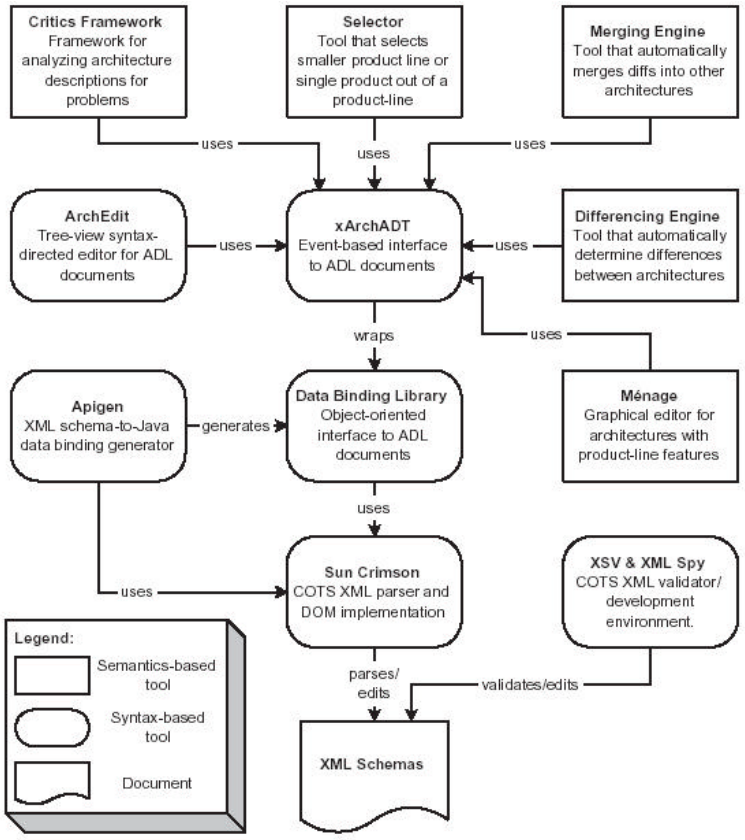

Figura 16: Ferramentas da Infra-estrutura xADL – extraído de (DASHOFY, HOEK e TAYLOR 2002)

Para avaliação da xADL foi obtido através da *web* o ambiente *Arch Studio* (HOEK, 2003). O ambiente que se propõe a especificar em alto nível arquiteturas de componentes baseadas em abordagem de linha de produto. Dentre as várias ferramentas existentes está a Ménage, uma ferramenta visual que permite componentes, interfaces providas e requeridas e conectores serem criados pelo projetista e organizados arquiteturalmente. A arquitetura gerada possui um arquivo XML correspondente a sua ADL, no qual são explorados os esquemas da Tabela 3 através de um conjunto de *tags* pré-definidos. A Figura 17 mostra um pequeno trecho de uma xADL gerada a partir de uma arquitetura definida para um sistema de *chat*. Nesta figura, podem ser observados os esquemas XML tipo e instância, utilizados para definição de componentes e interfaces.

```
<?xml version="1.0" encoding="UTF-8" ?>
- <instance:xArch …
  - <types:archStructure xsi:type="types:ArchStructure" types:id="ChatSystem">
      <types:description xsi:type="instance:Description">Chat System Demo Architecture</types:description>
    - <types:component xsi:type="types:Component" types:id="Server">
        <types:description xsi:type="instance:Description">Server Component</types:description>
      - <types:interface xsi:type="types:Interface" types:id="Server.IFACE_TOP">
          <types:description xsi:type="instance:Description">Server Component Top
            Interface</types:description>
          <types:direction xsi:type="instance:Direction">inout</types:direction>
          <types:type xsi:type="instance:XMLLink" xlink:type="simple" xlink:href="#C2TopType" />
        </types:interface>
      - <types:interface xsi:type="types:Interface" types:id="Server.IFACE_BOTTOM">
          <types:description xsi:type="instance:Description">Server Component Bottom
            Interface</types:description>
          <types:direction xsi:type="instance:Direction">inout</types:direction>
          <types:type xsi:type="instance:XMLLink" xlink:type="simple" xlink:href="#C2BottomType" />
        </types:interface>
        <types:type xsi:type="instance:XMLLink" xlink:type="simple" xlink:href="#Server_type" />
      </types:component>
    - <types:component xsi:type="types:Component" types:id="ChatClient1">
        <types:description xsi:type="instance:Description">Chat Client 1 Component</types:description>
      - <types:interface xsi:type="types:Interface" types:id="ChatClient1.IFACE_TOP">
          <types:description xsi:type="instance:Description">Chat Client 1 Component Top
            Interface</types:description>
          <types:direction xsi:type="instance:Direction">inout</types:direction>
          <types:type xsi:type="instance:XMLLink" xlink:type="simple" xlink:href="#C2TopType" />
        </types:interface>
      - <types:interface xsi:type="types:Interface" types:id="ChatClient1.IFACE_BOTTOM">
          <types:description xsi:type="instance:Description">Chat Client 1 Component Bottom
            Interface</types:description>
          <types:direction xsi:type="instance:Direction">inout</types:direction>
          <types:type xsi:type="instance:XMLLink" xlink:type="simple" xlink:href="#C2BottomType" />
        </types:interface>
        <types:type xsi:type="instance:XMLLink" xlink:type="simple" xlink:href="#Client_type" />
      </types:component>
```

Figura 17: Trecho de uma xADL para um sistema de *chat* (HOEK, 2003)

3.3.2 Considerações sobre ADLs

Enquanto que as ADLs não voltadas ao contexto de DBC não apresentavam uma forma padronizada de representar seus elementos e semântica envolvidos (conforme Figura 11), as ADL para DBC já apresentam um grau de conformidade maior, tendo em vista que elas estão voltadas a conceitos bem definidos como, por exemplo, componente e interface.

Quanto a ABC/ADL, observa-se que a mesma tem a preocupação em permitir que o próprio usuário defina seu estilo arquitetural para que este então possa ser empregado a um componente e, conseqüentemente, ao modelo de componentes ao qual pertence e que é especificado através da ADL.

Tanto para ABC/ADL quando para CDL não fica claro se o arquiteto de software pode especificar componentes de uma mesma arquitetura, os quais estão baseados em diferentes estilos arquiteturais. Uma arquitetura de componentes pode adotar um ou mais estilos arquiteturais, especialmente quando se considera não só os componentes de negócio, os quais foram explorados nos exemplos apresentados em (CHEN, 2002) e (MENCL, 1998), mas também componentes utilitários e de infra-estrutura conforme apresentado na Seção 2.2. No caso de ABC/ADL, os exemplos de componentes apresentados adotavam um único estilo arquitetural e, no caso da CDL, o autor parte do pressuposto que os componentes comunicam-se através de um estilo arquitetural hierárquico (componentes, subcomponentes). Como apresentado na Seção 3.2.1, estilos arquiteturais para componentes de negócio são baseados em tipos e baseados em instância e, para os demais componentes de uma arquitetura (utilitários, infra-estrutura), outros estilos arquiteturais genéricos poderiam ser adotados, de forma combinada ou não.

Com relação à sintaxe, a CDL parece tratar com maior precisão detalhes com relação a ligação (*binding*) de componentes, a especificação de suas interfaces, gerência de versões de componentes, e por outro lado, a ABC/ADL explora o uso da OCL para especificar restrições de tempo de acesso, utilização de componentes, segurança, etc.

Tanto numa ADL quando na outra não são especificadas ligações dos componentes gerados com os outros artefatos do processo de projeto que originaram tais componentes, interfaces e subcomponentes. Estes rastros poderiam de alguma forma ser especificados na ADL de maneira que a evolução e a manutenção da aplicação pudesse ser facilitada e, conseqüentemente, causar menores danos ao aplicativo já existente, proveniente da arquitetura gerada.

No que diz respeito a xADL a sua principal característica é a capacidade de estender os esquemas XML para suporte a outras características necessárias para uma determinada arquitetura. A xADL é uma descrição genérica, e o fato de utilizar um padrão como XML a torna mais flexível, ampliando suas possibilidades de utilização numa arquitetura a ser construída. Cabe portanto fazer um estudo mais detalhado desta ADL para obter informações do processo de extensão dos esquemas XML padrão da xADL, tendo em vista as necessidades específicas de uma atividade de projeto arquitetural.

3.4 Considerações Finais

O objetivo de analisar as propostas de estilos arquiteturais e ADL's, em especial aquelas que focam a tecnologia de componentes, é obter informações sobre o tipo de suporte oferecido por essas tecnologias numa atividade de projeto arquitetural, que é o foco da pesquisa proposta nesta monografia.

Neste sentido, o que se pode observar é que estilos arquiteturais, em especial aqueles voltados a componentes, auxiliam no processo de construção destes artefatos, sugerindo formas de acoplamento dos artefatos de análise para os artefatos de projeto. Numa atividade de projeto arquitetural, este tipo de apoio tende a facilitar a atividade do projetista de componentes, principalmente aqueles que representam o negócio da aplicação projetada. Quanto aos estilos arquiteturais genéricos, se observa a aplicabilidade de alguns deles para coordenação e comunicação dos diferentes tipos de componentes que podem ser fornecidos por uma arquitetura. Dos estilos arquiteturais genéricos, aquele que mais parece atender a estas expectativas é o conhecido estilo em camadas. Este estilo é comumente utilizado para expressar a arquitetura geral. Porém, internamente os componentes podem ser especificados por um conjunto de artefatos que por sua vez estão organizados por um estilo arquitetural genérico qualquer (e.g. *pipe and filter*).

Por outro lado, as ADLs parecem auxiliar fortemente na descrição da arquitetura de uma aplicação. Esta descrição pode apoiar a atividade do implementador na construção dos artefatos da arquitetura. No contexto do projeto arquitetural de componentes, as ADL's podem descrever os principais elementos arquiteturais de uma aplicação. Uma ADL pode ser utilizada para representar a semântica arquitetural dos componentes, independente do tipo de tecnologia no qual venham a ser implementados. Com este tipo de suporte, torna-se mais viável a construção de um projeto arquitetural tão independente quanto possível da tecnologia de implementação, ainda mais quando a ADL utiliza um padrão genérico como, por exemplo, a xADL.

Neste sentido, acredita-se que ADL's e estilos possam efetivamente apoiar o conjunto de alternativas de pesquisa para apoio ao projeto arquitetural baseado em componentes, apresentado no Capítulo 6.

4 Engenharia de Domínio

Neste Capítulo são apresentados e discutidos conceitos básicos relacionados a Engenharia de Domínio, etapas previstas e artefatos que são utilizados durante estas etapas.

4.1 Definições Básicas

A atividade de desenvolvimento de software pode ser melhor conduzida através de métodos, processos, dentre outras formas de orientação, obtendo como resultado um conjunto de artefatos gerados que atende às necessidades específicas de uma dada aplicação. No entanto, existem muitos domínios (e.g. ensino, hospitalar, contábil) que possuem muitas características comuns e que portanto, suas aplicações poderiam ser construídas a partir de um mesmo processo, conjunto de artefatos, e com isso, promovendo a reutilização dos conceitos e funcionalidades em comum.

Em Engenharia de Software existe uma área de pesquisa que se preocupa com esta questão, a qual é denominada de **Engenharia de Domínio** (ED). O objetivo da Engenharia de Domínio é possibilitar que características comuns e variáveis possam ser identificadas e modeladas com base num processo previamente definido. Este conjunto de artefatos gerados pela Engenharia de Domínio (e.g. componentes ou outros artefatos) pode ser instanciado para uma aplicação específica do domínio. A esta instanciação se dá o nome de **Engenharia de Aplicação** (EA). Neste sentido, a ED está preocupada em prover um conjunto de artefatos **para** reutilização, enquanto que a EA constrói aplicações **com** base na reutilização de artefatos gerados na ED. Alguns conceitos sobre ED e EA são apresentados a seguir:

"A Engenharia de Domínio busca uma criação sistemática de modelos de domínio e arquiteturas. A ED também dá suporte à engenharia de aplicação, a qual reusa estes modelos e arquiteturas para construir sistemas, enfatizando a reutilização" (SEI, 2003) .

"A Engenharia de Domínio é o processo de identificação e organização do conhecimento sobre uma classe de problemas, o domínio do problema, para suportar sua descrição e solução" (WERNER e BRAGA, 2000) (WERNER, 2003) .

"A intenção da Engenharia de Domínio é identificar, construir, catalogar e disseminar um conjunto de artefatos de software que tem aplicabilidade para software

existente e/ou a serem construídos, num domínio particular de aplicação" (WERNER, 2003) .

"A Engenharia de Aplicações desenvolve produtos de software baseado nos artefatos gerados pelo processo de Engenharia de Domínio". (SEI, 2003)

"A Engenharia de Aplicações se dedica ao estudo das melhores técnicas, processo e métodos para a produção de aplicações, no contexto da reutilização. Na EA, os componentes construídos na ED são utilizados para o desenvolvimento de um produto de software" (MILER, 2000) .

Durante a década de 80 e 90, começaram a surgir na comunidade de Engenharia de Software vários métodos de Engenharia de Domínio, dentre eles FODA (*Feature Oriented Domain Analysis*) (KANG *et al.*, 1993), FORM (*Feature Oriented Reuse Method*) (KANG, LEE e DONOHOE, 2002), FODAcom (VICI e ARGENTIERI, 1998) , RSEB (*Reuse-Driven Software Engineering Business*) (GRISS, FAVARO e D'ALESSANDRO, 2001) e ODM (*Organization Domain Modeling*) (SIMOS e ANTHONY, 1998) . Estes métodos de ED possuem propósitos comuns, ou seja, apresentar uma sistemática para modelagem de artefatos de mais alto nível no contexto de Engenharia de Domínio, promovendo a reutilização no contexto de EA. Embora cada um tenha suas devidas particularidades para obtenção e tratamento destes artefatos, existem alguns pontos em comum a respeito das atividades envolvidas no processo.

Em geral, a modelagem de um domínio atende a todas as atividades necessárias para a construção de artefatos que compreendem o núcleo de um software. Estas atividades incluem a identificação de um domínio e escopo, captura de variabilidades e similaridades de um domínio (**análise de domínio**), construção de projeto adaptável (**projeto do domínio**), ou seja, a especificação de uma arquitetura do domínio, e definição de mecanismos para tradução dos requisitos (resultados da análise e projeto do domínio) em sistemas criados a partir de componentes reutilizáveis (**implementação do domínio**). Os produtos destas atividades são modelos de análise, modelos de projeto, arquiteturas do domínio, linguagens específicas do domínio, geradores de código e código de componentes (SEI, 2003) .

A **análise de domínio** busca identificar, coletar e organizar informações relevantes do domínio, utilizando para tal o conhecimento existente do domínio e

técnicas para modelagem de informação. Estas técnicas de modelagem (e.g. FODA, FODAcom, FORM, RSEB, ODM) de informação permitirão determinar os limites que norteiam o domínio, as características comuns e variáveis de aplicações do domínio e gerar como resultados modelos para a representação de tais características.

Projeto de domínio é a etapa onde o desenvolvimento de modelos de projeto são construídos, com base nos modelos de análise, no conhecimento obtido por estudos a respeito de projeto reutilizável e arquiteturas genéricas. O projeto do domínio visa especificar a estrutura arquitetural a ser seguida pelas aplicações provenientes do domínio modelado, utilizando para isto especificações de projeto, padrões de projeto e arquitetural e estratégias para a construção/agrupamento de componentes (SEI, 2003) (BRAGA, 2000). Como resultado do projeto de domínio obtém-se modelos de projeto, os quais podem vir a ser, modelos de componentes projetados segundo um estilo arquitetural específico, modelos de classes, colaboração, seqüência, referentes à estrutura interna dos componentes e modelos de interação (e.g. colaboração, seqüência) entre componentes.

Por último, a atividade de **implementação do domínio** visa identificar os componentes reutilizáveis baseado na modelagem do domínio e na arquitetura genérica proposta na atividade de projeto. Estes componentes identificados vão formar uma biblioteca de componentes a ser construída durante esta atividade de implementação, obtendo-se como resultado os próprios componentes reutilizáveis, geradores de aplicações para o domínio, com base na arquitetura proposta.

Na Engenharia de Aplicações, as fases que norteiam o processo são análogas a ED. Como a EA baseia-se no processo de reutilização dos artefatos gerados na ED, as atividades que compreendem a modelagem de aplicações são mais específicas, atendendo não só características comuns ao domínio da aplicação, mas sobretudo as particularidades da própria aplicação.

Durante a Engenharia de Domínio espera-se o envolvimento de alguns profissionais, que segundo BRAGA (2000) são classificados como profissionais fonte (e.g. usuários de aplicações já desenvolvidas no domínio), produtores (e.g. analistas e projetistas do domínio que capturam informações obtidas pelas fontes) e consumidores (e.g. desenvolvedores de aplicações e usuários finais interessados no entendimento do domínio).

4.2 Artefatos de Análise de Domínio

Os métodos de Engenharia de Domínio propõem um conjunto de artefatos que promovem a estruturação de informações, as quais são contextualizadas no processo definido nos métodos. Os artefatos mais utilizados sob o ponto de vista de reutilização na EA estão: *feature*, caso de uso e tipos do negócio. Geralmente *features* determinam o ponto de corte das características do domínio a serem reutilizadas na EA, enquanto que e os demais especificam sob diferentes perspectivas as *features* do domínio. No contexto desta monografia serão brevemente discutidos e detalhados os artefatos *feature*, caso de uso e tipos, pois são os mais representativos no contexto de análise de domínio.

4.2.1 Feature

Features representam as capacidades/características do domínio. Tais abstrações do domínio são obtidas através de especialistas, usuários e sistemas já existentes, sendo que estas abstrações não existem isoladamente, ou seja, podem existir relacionamentos entre elas (e.g. associação, composição), os quais são modelados nos métodos de ED através de um modelo de *features*.

Um modelo de *features* descreve a teoria do domínio. Ele não inclui somente as características do domínio, mas também descreve como elas estão relacionadas estruturalmente (BRAGA, 2000). O modelo de *features* é um elemento chave para relacionar os conceitos de mais alto nível aos demais artefatos de análise e também para os artefatos de projeto, além de ser considerado o melhor caminho para efetuar os "pontos de corte" para a reutilização de parte de um domínio (um conjunto específico de *features* do domínio) para uma dada aplicação (BRAGA, 2000) (KANG *et al.*, 1993) (VICI e ARGENTIERI, 1998) (SIMOS e ANTHONY, 1998) (GRISS, FAVARO e D'ALESSANDRO, 2001). As *features* devem estar ligadas, através do conceito de rastreabilidade, aos demais artefatos que correspondem a um dado domínio.

4.2.1.1 Tipos de *Feature*

Alguns autores propõem diferentes tipos de classificações para *features*, com o intuito de distinguir o tipo de informação que ela representa. GRISS *et al.* (2001), KANG, LEE E DONOHOE (2002) (KANG *et al.*, 1993), MILER (2000) classificam *features* da seguinte forma:

- obrigatória: corresponde a características que fazem parte da infra-estrutura básica do domínio;
- opcional: corresponde à identificação de algumas características que podem não ser necessárias para alguns sistemas do domínio. As *features* opcionais correspondem a características secundárias do domínio se comparadas as *features* obrigatórias;
- variação: corresponde a formas alternativas de configurar *features* obrigatórias e/ou opcionais.

MILER (2000) propõe uma classificação para *feature* no contexto da infra-estrutura de reutilização baseada em modelos de domínio – Odyssey. Para o autor, as *features* podem ser classificadas como segue:

- *Features* Funcionais e Conceituais: a primeira categoria representa as funcionalidades do domínio e a segunda, os conceitos. Cada uma dessas *features* é categorizada como obrigatória, opcional ou variação, conforme classificação anterior. MILER (2000) propõem ainda que estas *features* funcionais e conceituais sejam também categorizadas como:
    - Organizacional: com intuito apenas de facilitar o entendimento ou organizar o domínio;
    - Entidade: para representar os atores do modelo;
    - Externa: para representar a ligação com outros domínios, podendo ser ou não refinadas pelo modelo;
    - Adicional: representam características adicionais, importantes para compreender o domínio;
    - Não-definida: características que já foram identificadas no domínio, mas que não foram específicas por outros artefatos do domínio (e.g. casos de uso, classes).
- *Features* de Projeto: o nível de abstração desta *feature* é mais baixo que as funcionais/conceituais. Estas *features* são classificadas como:
    - Operacional: derivadas das *features* funcionais/conceituais, representando seus aspectos tecnológicos;

    - Arquiteturais: representam aspectos essencialmente tecnológicos, e estão ligadas geralmente a padrões arquiteturais;
    - Incompletas: não estão definidas nem implementadas por completo, embora sejam necessárias no contexto de reutilização.
- *Features* Implementacionais: ligadas às fases de implementação do domínio e de uma aplicação, representam características associadas à codificação, dentre elas: compilação, dados, execução, etc.

LEE, KANG e LEE (2002) e KANG, LEE e DONOHOE (2002) propõem uma classificação para *features* as quais são organizadas em quatro camadas:

- Camada de Capacidades: caracteriza os serviços, operações e funcionalidades de uma dada aplicação ou domínio;
- Camada de Ambiente Operacional: as *features* dessa camada representam atributos de um ambiente que uma aplicação do domínio pode usar e operar, geralmente representando características de plataforma de hardware adotada (e.g. tipo de terminal);
- Camada de Tecnologia de Domínio: as *features* representam detalhes de implementação de mais baixo nível, específicos para o contexto de um domínio (e.g. técnicas cirúrgicas num domínio neurocirúrgico);
- Camada de Técnica de Implementação: as *features* representam detalhes de implementação de mais baixo nível, contudo de cunho mais genérico que as relativas a camada de tecnologia de domínio. (e.g. técnicas de implementação de algoritmos de busca).

4.2.1.2 Relacionamento de *Features*

Os modelos de *features* nos métodos de ED têm adaptado conceitos de relacionamento do modelo Entidade Relacionamento, e mais recentemente da UML, para representar relações entre *features*. Através das relações, podemos melhor avaliar o grau de acoplamento entre as *features* do domínio modelado, gerando árvores ou redes de *features*, e servindo como ponto de partida para futuras composições de artefatos (componentes gerados a partir das *features*).

KANG, LEE e DONOHOE (2002), KANG *et al.* (1993) e LEE, KANG e LEE (2002), propõem as relações *composed-of*, para representar composição entre *features*, *generalization*, para representar relação de herança, e *implemented-by*, para representar que *features* podem ser implementadas por outras *features*, que geralmente se encontram nas camadas de ambiente operacional, tecnologia de domínio e/ou de técnicas de implementação. Pode também ser observada uma característica adicional de *exclusividade*, onde *features* filhas não podem ser usadas ao mesmo tempo.

Em (MILER, 2000), são apresentados outros tipos de relacionamento para um modelo de *features*, que são: composição, onde uma *feature* é composta de várias outras, isto é, uma não existe sem a outra; agregação, onde uma *feature* representa o todo e suas partes,não existindo uma dependência entre elas; herança, análogo a proposta de *generalization* de KANG et al (1993) e LEE *et al.*(2002) e associação, para representar uma ligação simples entre *features*.

Por último, GRISS *et al.* (2001) propõem que num modelo de *features* estejam previstos os seguintes relacionamentos: composição, análogo a proposta de Kang et al e alternativa, também chamada de ponto de variação, este relacionamento é atribuído a uma *feature* que determina um fluxo alternativo para outras *features* (e.g. uma estação telefônica – *feature* alternativa, pode ser através de PABX ou linha individual).

As duas figuras que se seguem (Figura 18 e Figura 19) apresentam dois modelos de *features*, onde na primeira é ilustrado um modelo de *features* baseado na proposta de KANG, LEE e DONOHOE (2002) e na segunda um modelo de *features* segundo a proposta de MILER (2000). No modelo da primeira figura, se observa as *features* das diferentes camadas propostas pelo autor, a indicação de opcionalidade de algumas *features* e os diferentes tipos de relacionamento existente no domínio. No modelo da segunda figura, se observa somente as *features* conceituais/funcionais, as especificidades da categorização proposta pelo autor, bem como os relacionamentos previstos, ou seja, não estão contempladas as *features* de projeto e implementacionais previstas pelo autor.

4.2.1.3 Considerações sobre *Features*

Existem diferenças entre as propostas de conceituação de *features* e dos possíveis relacionamentos. Embora a proposta de MILER (2000) seja considerada uma extensão do

modelo de *features* do método FODA, a preocupação numa definição mais detalhada do conceito de *feature* tornou-o consideravelmente diferente da proposta original, e dos demais autores.

A proposta de *features* conceituais e funcionais no contexto do trabalho de MILER (2000) tem forte base no conceito de ontologias, o qual está ligado ao contexto de gerência de conhecimento. Ontologias representam o conhecimento de um domínio e estas são basicamente caracterizadas como ontologias conceituais e ontologias de tarefas, onde em geral a primeira corresponde às *features* conceituais, e a segunda às *features* funcionais (BRAGA, 2000). Conforme apresentado em MANGAN *et* al. (2001), ambas as técnicas (ontologias e *features*) foram propostas como uma forma de otimizar a construção de sistemas de computador. Modelos de *features* são usados, principalmente, pela comunidade de reutilização de software. Ontologias são aplicadas, principalmente, pela comunidade de inteligência artificial na perspectiva de modelagem de conhecimento.

Em KANG, LEE e DONOHOE (2002), LEE, KANG e LEE (2002), GRISS *et al.* (2001) não existe esta preocupação em categorizar as *features* que representam conceitos ou funcionalidade do domínio. Estas *features* não possuem muitos níveis de detalhamento quanto a sua representação visual e valor semântico, diferenciando-se somente pelo fato de poderem ser obrigatórias, opcionais ou variação. No entanto, os autores distinguem estas *features* ligadas especificamente ao domínio da aplicação, daquelas que de alguma forma dão suporte técnico, tecnológico e implementacional ao domínio.

Do ponto de vista de uso do conceito, as propostas de KANG, LEE e DONOHOE (2002), GRISS *et al.* (2001) e LEE, KANG e LEE (2002) são mais flexíveis, não exigindo um conhecimento tão detalhado do conceito de *feature* como o exigido em MILER (2000) (e.g. *feature* organizacional, não definida, adicional). Tal flexibilidade pode tornar a atividade de obtenção das *features* do domínio mais produtiva, no sentido de valorizar mais a busca, do que efetivamente a contextualização da informação obtida numa especificidade do conceito que ela contempla.

No que diz respeito ao relacionamento, observa-se que, na proposta de MILER (2000), houve também um maior detalhamento dos mesmos se comparado às outras abordagens. Observa-se, nesta proposta, uma tendência no uso de alguns conceitos de relacionamentos, explorados atualmente através da UML, como, por exemplo, agregação

e associação simples que não aparecem nas demais propostas. Acredita-se que, principalmente, a relação de associação simples é bastante producente, pois nem sempre *features* estão relacionadas por meio de composição, generalização ou implementação como propõem KANG, LEE e DONOHOE (2002), por exemplo.

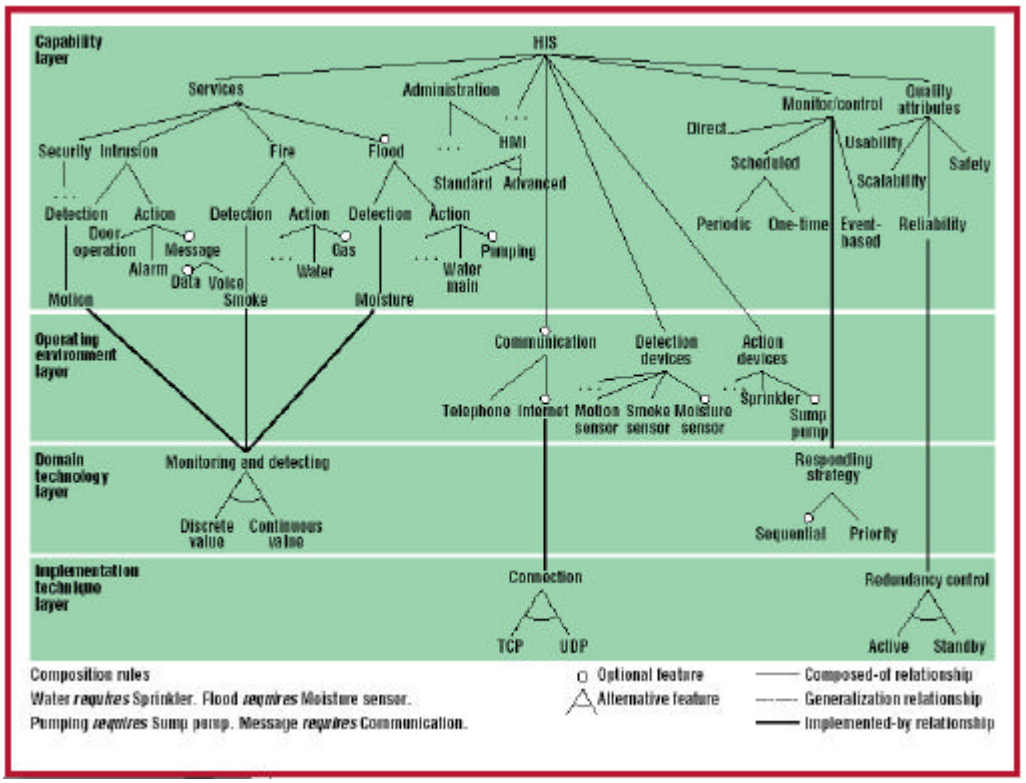

Figura 18: Modelo de *Features* sobre o Domínio de Sistema de Segurança apresentado em (KANG, LEE e DONOHOE, 2002)

O objetivo das regras de relacionamento é, como já sugere o nome, definir associações, hierarquias e dependências entre elementos e, no contexto de um modelo de *features*. Estas relações serão fator determinante não só para conduzir a construção dos demais artefatos da análise, projeto e implementação do domínio, como para a reutilização destes artefatos no contexto de EA. Neste sentido, quanto mais claras estiverem determinadas as relações existentes entre as *features*, mais produtiva será a atividade do engenheiro de domínio na geração de artefatos ligados a estas *features*. Uma *feature* pode derivar em muitos outros artefatos que de alguma forma devem

corresponder na demais fases do ciclo de vida, as relações estabelecidas no modelo de *features* do qual foram embasados.

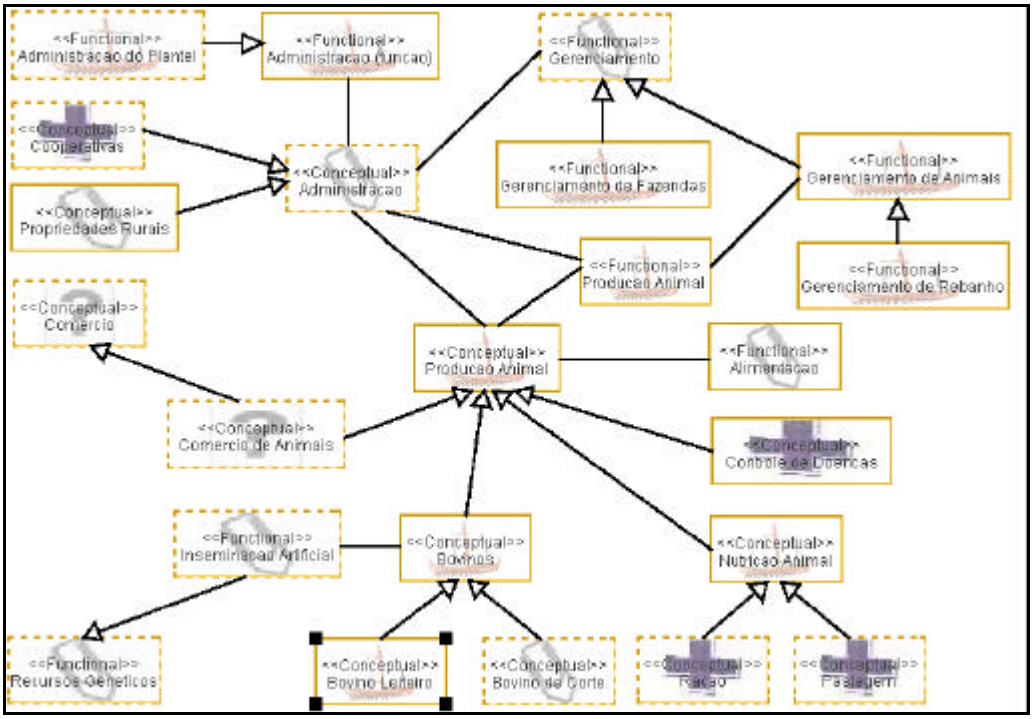

Figura 19: Modelo de *Features* para Domínio de Lactação no Odyssey

Embora a classificação correta de uma *feature* e seu relacionamento com outras *features* seja efetivamente importante, do ponto de vista de utilização percebe-se algumas dificuldades de identificar o que venha a ser uma *feature* do domínio e o que distingue uma *feature* de uma função, um objeto ou um aspecto, por exemplo. Neste sentido, LEE *et al*. (1999) afirmam que funções, objeto, aspectos são principalmente usados para especificar detalhes internos de um sistema. Por outro lado, as *features* visam identificar características de um domínio que são visíveis externamente em termos de similaridades e variabilidades e, a partir do entendimento destas características similares e variáveis, podem ser derivadas funções, objetos e outros artefatos de reutilização.

LEE *et al.* (1999) apresentam um conjunto de orientações para modelagem de *features* que podem vir ao encontro às necessidades de um Engenheiro de Domínio e que de algum forma atendem às preocupações citadas durante esta Seção. Estas orientações contemplam o processo de identificação de *features* de um domínio, categorização,

focando na proposta de KANG *et al.* (1993) (KANG, LEE e DONOHOE, 2002) (LEE, KANG e LEE, 2002), organização das *features*, explorando as regras de relacionamento, apoiando a construção do modelo de *features*, e orientações para refinamento de *features* em artefatos de projeto. Embora estas orientações tenham sido exploradas especificamente para o método FODA e suas extensões, nada impede que sejam também aplicadas para a identificação de *features* em outros métodos de ED.

4.2.2 Caso de Uso

Um outro artefato utilizado no contexto de análise de domínio é o caso de uso. Casos de Uso são artefatos originalmente utilizados no contexto de métodos de modelagem OO, com destaque a UML.

BOOCH *et al.* (2000) definem um caso de uso como uma forma de especificar o comportamento de um sistema ou de parte de um sistema, e serve como uma descrição de um conjunto de seqüência de ações. Salientam ainda que casos de uso podem ser aplicados para captar o comportamento pretendido do sistema que está sendo desenvolvido, sem que seja necessário especificar como esse comportamento é implementado.

No contexto de análise de domínio, casos de uso têm a intenção de apoiar a atividade de aquisição de conhecimento.

Em (BRAGA, 2000), este processo de aquisição de conhecimento começa pela descrição de cenários existentes, os quais são instanciados para templates de casos de uso previamente definidos pela autora. O *template* de caso de uso permite descrever, dentre outras informações, o escopo, pré-condições, condição de sucesso final, condição de finalização falha, atores primários e secundários para a condição inicializadora ao caso de uso e seqüência de passos do caso de uso. Estes casos de uso descritos devem, então, ser analisados pelo engenheiro de domínio, o qual abstrai as descrições e gera os chamados casos de uso de domínio. Os casos de uso de domínio possibilitam a identificação de atores envolvidos, conceitos e atividades que serão modelados por outros artefatos (e.g. *feature*). Segundo a autora, casos de uso de domínio devem ser mais abstratos que os casos de uso da UML. No entanto, pelo fato de casos de uso já representarem um nível de abstração alto, tal diferenciação parece bastante tênue. Quanto à notação, caso de uso de

domínio em (BRAGA, 2000) são descritos da mesma forma que os casos de uso da UML.

Em (VICI e ARGENTIERI, 1998), é apresentado o emprego de casos de uso de domínio no contexto de análise de requisitos do método FODACom, o qual sofreu alterações significativas no método original FODA. Casos de uso de domínio servem para representar o fluxo de eventos existentes entre atores do domínio. Atores neste contexto representam sistemas externos que interagem com o domínio sendo modelado, e atores internos, pertencentes ao domínio. O modelo de casos de uso de domínio, em (VICI e ARGENTIERI, 1998), estende a notação padrão UML, adicionando dois tipos de mecanismos para representar variabilidades nos fluxos de eventos, sendo eles: parametrização e ponto de extensão. A parametrização é adicionada na descrição do caso de uso e serve para representar certas características que variam no domínio, de um sistema para outro (atores do domínio). Segundo os autores o objetivo da parametrização é apoiar a validação de um caso de uso genérico no contexto do domínio. Pontos de Extensão são usados na descrição textual de um caso de uso e expressam a variação de comportamento de um sistema do domínio para outro (atores do domínio).

O método de Engenharia de Domínio RSEB também utiliza casos de uso para capturar conhecimento sobre um domínio. No entanto, ao contrário da maioria dos métodos de ED (e.g. FODA, FORM, ODM, FODAcom), RSEB não utiliza o conceito de *features*. GRISS *et al*. (2001) afirmam que o conceito de *feature* originou-se a partir do método FODA (KANG *et al.*, 1993) e no início da década de 90 casos de uso ainda não eram amplamente aceitos pela comunidade de orientação a objetos como uma forma de análise de requisitos, ao contrário do papel que hoje eles desempenham. No entanto, os mesmos autores afirmam que *features* e casos de uso possuem propósitos diferentes. Enquanto a primeiro tem por objetivo identificar características para reutilização futura e que são exploradas pelo engenheiro de domínio, o segundo pode ser aplicado para elicitar requisitos do ponto de vista do usuário de um sistema, os quais são identificados por um engenheiro de sistema.

Assim, os casos de uso podem ser aplicados de forma complementar às *features*. Enquanto casos de uso podem levantar e descrever requisitos do usuário a respeito das funcionalidades de um domínio, *features* podem ser aplicadas para organizar a análise de

requisitos em termos de similaridades e variabilidades, viabilizando uma futura reutilização.

Neste sentido, GRISS *et al*. (2001) propõem a integração de modelos de *features* a fase de análise de domínio do RSEB, predominada originalmente por modelos de casos de uso. A esta nova abordagem foi dado o nome de FeaturRSEB.

As duas figuras que se seguem (Figura 20 e Figura 21) apresentam exemplos de casos de uso de domínio no contexto do trabalho de BRAGA (2000) e VICI e ARGENTIERI (1998), respectivamente. Nota-se que em ambos exemplos, a notação UML parece ser padrão, embora no exemplo da abordagem de VICI e ARGENTIERI (1998) são usados estereótipos para representar parametrização e pontos de extensão.

4.2.2.1 Considerações sobre Casos de Uso

Casos de uso e modelos de casos de uso são bastantes conhecidos pela comunidade que desenvolve análise de sistemas OO. Casos de uso trabalham num nível de abstração que permite o levantamento de requisitos de uma aplicação e, no contexto de domínio, aqueles que envolvem um conjunto de aplicações. Embora seja empregado o termo "casos de uso de domínio", não fica clara a diferença de nível de abstração nos modelos de casos de uso de domínio, se comparado ao emprego que convencionalmente é dado a casos de uso no desenvolvimento de aplicações. No entanto, os aspectos de variabilidade que são, por exemplo, estereotipados como na proposta de VICI e ARGENTIERI (1998), estes sim podem determinar o que deva ser considerado em nível de domínio e o que poderá ser ou não adotado em nível de aplicações derivadas.

O emprego de casos de uso, complementar a outros artefatos da ED, pode melhorar o processo de especificação de requisitos de um domínio. Na Seção 4.2.2, foi citado a possibilidade de uso combinado de *features* e casos de uso para trabalhar com os requisitos de um domínio, cada um tratando estes requisitos sob diferentes pontos de vista. Se considerarmos a abordagem de categorização de *features* de MILER (2000), por exemplo, poderíamos considerar que *features* funcionais poderiam ser descritas através de casos de uso de domínio. Na Seção 5, serão discutidos alguns exemplos de processos de Engenharia de Domínio, onde estas relações complementares entre os artefatos poderão ser melhor contextualizadas.

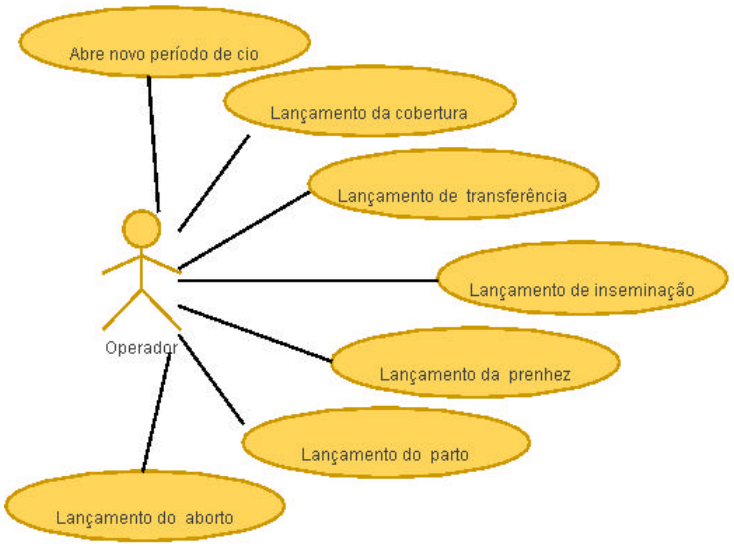

Figura 20: Modelo de Casos de Uso de Domínio no Odyssey

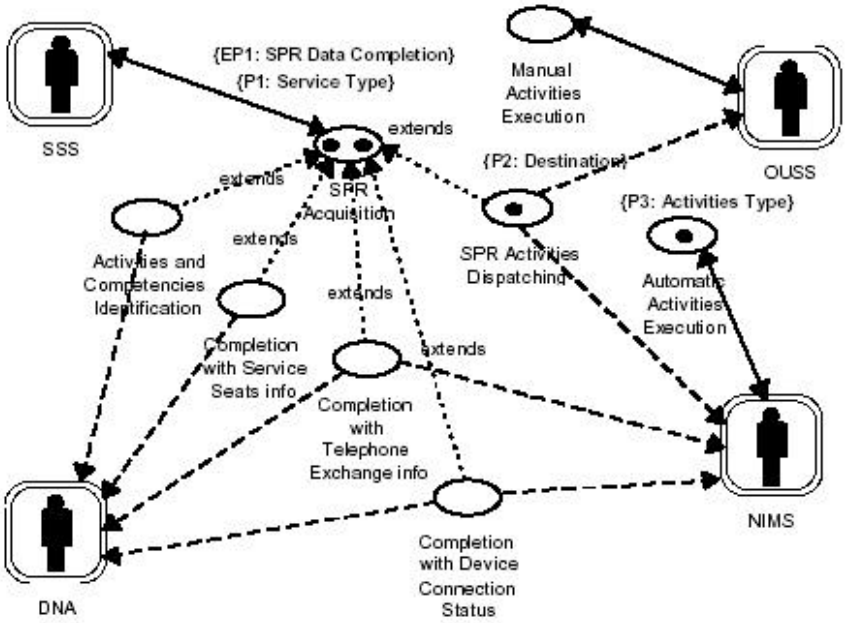

Figura 21: Modelo de Casos de Uso no FODAcom para o domínio de abastecimento de serviços para empresas de telecomunicações (VICI e ARGENTIERI, 1998)

4.2.3 Tipos do Negócio

Tipos do Negócio representam os aspectos estáticos de uma aplicação ou domínio, e a partir deles pode-se dar início à identificação dos possíveis artefatos

candidatos à persistência no contexto da atividade de projeto. Um modelo de tipos/classes de análise mostra como estes conceitos estão relacionados uns aos outros.

BROWN (2000) sugere que, inicialmente, este modelo de tipos seja gerado de forma análoga a mapas conceituais, ou seja, obter-se simplesmente conceitos e relacionamentos entre eles, com estereótipos associados aos relacionamentos. Num estágio mais aprofundado do modelo de tipos, o autor sugere que os mapas conceituais sejam refinados para receber informações como atributos dos tipos e cardinalidades para os relacionamentos. Neste caso, os mapas conceituais começam a sofrer alterações tomando forma semelhante aos modelos de entidade-relacionamento. De posse deste último refinamento, caberá ao engenheiro de domínio analisar os artefatos (tipos) gerados e novamente refiná-los para obter um conjunto de tipos do domínio melhor definidos.

Em BRAGA (2000), os tipos do negócio, juntamente com os casos de uso de domínio, permitem gerar modelos de colaboração. Estes modelos de colaboração definem as interações dos tipos participantes de um caso de uso, sugerindo futuros subsistemas/pacotes/componentes do domínio em função do relacionamento existente entre os tipos. No trabalho de BRAGA (2000), os tipos de negócio são obtidos através de *features* conceituais existentes no modelo de *features* de um domínio, as quais foram propostas no contexto do trabalho de MILER (2000). Para cada *feature* conceitual é criado um tipo de negócio equivalente. Em TEIXEIRA (2003), os tipos de negócio e os casos de uso são artefatos utilizados na geração de componentes de negócio e de processo, atendendo a fase de projeto de domínio, com base em estilos arquiteturais para componentes. Os trabalhos de BRAGA (2000), MILER (2000) e TEIXEIRA (2003) fazem parte do projeto Odyssey e que, portanto, são utilizados de forma combinada nesta infra-estrutura de reutilização.

O método FODA tem sua proposta originada no paradigma estruturado, no qual são disponibilizados modelos hierárquicos (modelo de *features*), modelos de entidade-relacionamento e diagrama de fluxo de dados. Como a proposta de um modelo de tipos de negócio é bastante semelhante aos modelos E-R, observa-se que está preocupação em capturar elementos estáticos do domínio é contemplada no método. No entanto, na extensão do FODA, método FORM, com uma proposta baseada no paradigma OO, a representação dos aspectos estáticos poderá ser representada através de diagramas de

tipos, classes ou por meio de componentes, embora tenham propostas distintas. No entanto, não existe informação precisa quanto ao uso de quais tipos de artefatos OO são utilizados no método. O mesmo acontece com o método RSEB. Este tem foco no paradigma estruturado, mas não ficam claros na literatura quais são os artefatos adotados. Em geral, a literatura explora o emprego dos modelos de *features* para captura de informações do domínio, técnicas de agrupamento de *features* para componentes, rastreabilidade entre artefatos (e.g. *features* para componentes).

4.2.3.1 Considerações sobre Tipos de Negócio

Embora este artefato não esteja explicitamente difundido nos métodos de ED citados anteriormente, as informações obtidas através de um modelo de tipos podem representar um degrau importante numa possível escala de diminuição de abstração de informações de um domínio, as quais são geralmente obtidas através de modelos de *features* e casos de uso. Outra vantagem vista no uso de um modelo de tipos é que ele consegue apresentar uma semântica mais robusta para representar relações e propor um conjunto inicial de atributos ligados aos tipos do negócio. Do ponto de vista de OO, um modelo de tipos pode ser visto como uma primeira tentativa de identificar classes de um sistema, as quais serão aprimoradas no contexto de projeto para atender as necessidades da aplicação, conforme previsto nos seus casos de uso.

Tendo em vista estas vantagem, este artefato também pode ser vantajoso na modelagem de domínios baseados no paradigma OO.

4.2.4 Outros Artefatos

Na Análise de Domínio pode ser encontrado o uso de outros tipos de artefatos como diagramas de contexto, estados, seqüência, colaboração, fluxos de dados- DFD e modelos E-R, os quais são igualmente importantes como os descritos anteriormente, mas que do ponto de vista da literatura existente, não foi dada a mesma ênfase atribuída a *features*, casos de uso e tipos.

Além de artefatos do tipo modelos, outros artefatos como documentação do domínio, formulários, sistemas legados, etc, são igualmente utilizados, principalmente na etapa de levantamento de requisitos, servindo como uma das fontes de informação para a geração dos artefatos e modelos acima discutidos.

4.3 Artefatos de Projeto de Domínio

Tendo em vista a fase de análise de domínio e seus artefatos e modelos gerados, no projeto de domínio, são utilizados outros artefatos que promovem a estruturação das soluções para o domínio. Em geral, os artefatos permitem o registro de informações de mais baixo nível como operações, restrições, integridade, acesso, etc. Neste sentido, os artefatos que mais se destacam em projeto de domínio são classe e componente. Pacote, também, é um artefato que tem sido empregado para representar subsistemas ou agrupamento de um conjunto de componentes e/ou classes de um domínio. Estes artefatos serão brevemente discutidos nas seções que se seguem, tendo em vista sua importância no projeto de domínios e aplicações, e o suporte por eles oferecidos para as fases subseqüentes. Salienta-se que tanto o artefato classe, quanto componente são utilizados no contexto de métodos de ED que estão focados no paradigma OO. Como o nosso trabalho de pesquisa busca apoiar mais efetivamente a atividade de projeto arquitetural em ED, estes artefatos serão novamente abordados no contexto da proposta de pesquisa, nas seções 5, 6.1.3 e no Capítulo 7.

4.3.1 Classes

O artefato classe em ED é utilizado da mesma maneira do que em métodos OO. Classes encapsulam dados e comportamento. Classes provavelmente se relacionam com outras classes com uma cardinalidade associada, a qual é expressa através de um modelo de classes. A dinâmica existente entre estas classes pode ser melhor compreendida através de diagramas de seqüência e/ou colaboração correspondentes.

Métodos como FORM e FeaturRSEB, utilizam modelos de classes para apoio a construção da arquitetura do domínio. Embora os métodos anteriormente citados não tenham foco no desenvolvimento de projeto de arquiteturas baseados em componentes, modelos de classes deveriam ser utilizados para definir a arquitetura interna de cada componente. No entanto, tal suposição não pode ser confirmada, pois a literatura a respeito dos métodos atribui maior foco a atividade de análise de domínio.

No contexto do Projeto Odyssey, inicialmente, classes representavam os artefatos de mais baixo nível de uma arquitetura de domínio, mas com a adição de componentes neste contexto (TEIXEIRA, 2003), modelos de classes devem passar a descrever a

arquitetura interna de cada componente existente na arquitetura em geral. Detalhes a respeito deste processo serão melhor abordados na Seção 6.

4.3.2 Componentes

O conceito de componente tinha ainda no início da década de 90 uma forte associação a código, ou seja, representava no processo de modelagem de um sistema, um conjunto de bibliotecas de software, componentes gráficos, dentre outros.

A área de Desenvolvimento Baseado em Componentes foi se tornando mais difundida na academia e na indústria ao longo dos anos 90. Baseado neste fato e também pela necessidade de reutilização não só de código, mas de artefatos de projeto, análise, etc, é que componentes foram sendo empregados em maior escala na fase de projeto de sistemas.

Componentes no contexto de projeto, são aplicados para encapsular dados e comportamentos que possuem um grau de relacionamento (e.g. herança). Neste sentido, o propósito de um componente pode vir a ser análogo ao propósito de uma classe. No entanto, componentes podem ter diferentes níveis de granularidade, onde aqueles de maior nível poderão internamente possuir um conjunto de classes, ou seja, um modelo de classes interno. Para que as informações encapsuladas possam ser acessadas, um componente deve prover um conjunto de assinaturas que definem a forma como artefatos externos fazem acesso a ele (interfaces fornecidas) e também determinam a forma como ele próprio utiliza outros componentes para o desenvolvimento de seus comportamentos previstos (interfaces requeridas).

Alguns métodos de ED utilizam componentes como artefatos de projeto. Em TEIXEIRA (2003), é proposta uma sistemática de apoio a geração de componentes de negócio e de processo, a partir de artefatos de análise, a saber, casos de uso e tipos de negócio. Esta sistemática está contextualizada no processo de ED e EA proposto por BRAGA (2000) e MILER (2000), respectivamente, o qual será brevemente apresentado na Seção 5.1. No seu trabalho, TEIXEIRA (2003) propôs que componentes de negócio fossem gerados a partir de dois estilos arquiteturais (baseados em tipos e baseados em instâncias), previamente discutidos na Seção 3.2.1. Os componentes gerados atendem a um conjunto de heurísticas baseadas nas regras de relacionamento entre os tipos do negócio.

Outro método de ED que utiliza componentes como artefatos de projeto é o FORM (KANG, LEE e DONOHOE, 2002). Neste método, pode-se identificar dois tipos de componentes: conceitual e concreto. O projeto no FORM começa pela definição da arquitetura dos componentes de mais alto nível, sendo possível através destes componentes especificar dados e controle de dependências entre componentes. Geralmente componentes conceituais correspondem a subsistemas existentes no domínio. Os componentes concretos são resultado do refinamento da arquitetura de componentes conceituais, os quais já possuem especificação utilizando padrão UML. O refinamento dos componentes poderá ser apoiado por técnicas de geração de código, encapsulamento, parametrização, uso de *frameworks*, padrões, *templates*, etc.

4.3.3 Pacotes

Por último, o artefato pacote é utilizado para representar subsistemas de uma arquitetura e, através de diagramas de pacotes, são expressas as interações entre subsistemas. Pacotes internamente podem ser compostos por qualquer artefato do domínio que faz parte da semântica que ele representa.

A partir do final da década de 90, o uso de diagramas de pacotes tem caído em desuso principalmente para aplicações baseadas em componentes. Tal fato se deve a basicamente dois motivos: 1) emprego mais orientado à tecnologia de componentes e, portanto, arquiteturas passarem a ser expressas através de diagramas de componentes e, 2) a semântica fraca do ponto de vista arquitetural que um diagrama de pacotes tende a expressar, não só no contexto de ED, mas também para sistemas sem perspectiva de reutilização. Dos métodos de ED, somente a proposta original de FODA utiliza tal artefato.

Embora pacotes tenham apresentado problemas quanto à representação semântica, ainda existem propostas de uso, principalmente para aplicações baseadas no paradigma estruturado. Também no paradigma OO, alguma utilização é atribuída ao artefato pacote. Por exemplo, na especificação da UML 2.0, pacotes são utilizados para empacotamento de componentes lógicos (e.g. componentes de negócio, processo) e componentes físicos (e.g. componentes EJB, CORBA, etc.) (UML, 2003) .

4.3.4 Outros artefatos: seqüência e colaboração

Assim como na fase de análise, outros tipos de artefatos podem ser adotados mas que na literatura não foi dado ênfase a sua utilização, comparada aos artefatos classe, componente e pacote. Também em projeto podem vir a ser utilizados diagramas de seqüência e colaboração para expressar a interação entre componentes e classes de um domínio. No entanto, o nível de abstração de diagramas de seqüência e colaboração na fase de projeto seja consideravelmente mais baixo comparado ao uso destes num contexto de análise. Em projeto, as interações podem ter assinaturas baseadas em métodos de classes, interfaces dos componentes, referência a atributos, dentre outros dados encapsulados por classes e/ou componentes de projeto.

4.3.5 Considerações Finais sobre Artefatos de Projeto em ED

O propósito dos artefatos de projeto é representar de forma mais concreta as soluções de um domínio, tendo como base as informações provenientes dos artefatos de análise. No processo de ED apresentado em BRAGA (2000), é proposto o uso de diagramas de classes associado a cada caso de uso do domínio durante a fase de análise e, no contexto de projeto, a construção da arquitetura do domínio baseado em componentes e interfaces. No entanto, a forma como o diagrama de classes está sendo utilizado neste contexto não acrescenta nenhuma informação que possa vir a ser aproveitada como base para a criação da arquitetura de componentes proposta pelo autor.

Quanto aos outros métodos de ED, como já citado anteriormente, existem poucos relatos sobre como ocorre a fase de projeto num determinado domínio e, conseqüentemente, como são utilizados os artefatos relacionados a esta etapa da ED.

Com o estudo dos métodos de ED existentes e com a experiência na utilização da infra-estrutura de reutilização Odyssey, acredita-se que uma forma mais apropriada de utilização destes artefatos no projeto de ED seria através do uso de componentes para representar serviços e dados do domínio. Estes componentes devem estar em consonância com o conceito de componente proposto por SAMETINGER (1997), abordado no Capítulo 2, ou seja, um elemento arquitetural que represente uma ou um conjunto de características afins de um determinado domínio. A partir deste elemento arquitetural que um componente pode representar no projeto de ED, especificações a seu respeito devem ser desenvolvidas. Neste sentido, tais especificações poderiam ser construídas a partir de

modelos de classes. Esta forma de utilização destes artefatos de projeto será adotada na pesquisa que está sendo proposta nesta monografia, com apoio de um conjunto de orientações para viabilizar esta atividade. Uma proposta de alternativas de pesquisa para apoio ao projeto arquitetural, baseado nestes artefatos, será abordada na Seção 6.

4.4 Considerações Finais

Neste capítulo foram analisados os principais artefatos que contemplam as fases de análise e projeto em Engenharia de Domínio. Esta análise pôde destacar a importância destes artefatos para a representação dos diferentes níveis de abstração de um domínio modelado, bem como as particularidades atribuídas a cada artefato no contexto dos métodos de ED, até que o mesmo represente um artefato de implementação, numa linguagem de programação específica.

No entanto, a ED não acontece simplesmente pelo uso do conjunto de artefatos apresentados anteriormente, mas sim através de um processo coordenado que conduza a sua utilização, com o objetivo de se obter melhores resultados da Engenharia de Domínio. Este processo coordenado tende a sugerir as melhores alternativas para a construção da Engenharia de Aplicações.

Como citado ao longo deste Capítulo, foram propostos alguns métodos de ED (e.g. FODA, FORM, FeaturRSEB, Odyssey-DE) para orientar a atividade de modelagem do domínio. No entanto, os processos sugeridos apresentam algumas lacunas, principalmente com relação às atividades que envolvem a fase de projeto do domínio.

Neste sentido, nos próximos dois capítulos desta monografia são feitas análises a respeito de processos para ED, e do apoio ao projeto arquitetural baseado em componente existentes na literatura, respectivamente. Ao final de cada Capítulo é apresentado uma proposta inicial de alternativas para a solução dos problemas identificados nas abordagens existentes, que são o objetivo principal desta pesquisa.

No Capítulo 5, são analisadas as abordagens de processo de ED. Nesta análise, são identificadas as principais lacunas observadas nos processos de ED existentes, principalmente no que diz respeito ao apoio para a construção da arquitetura do domínio. Tendo em vista os problemas detectados, ao final do Capítulo é apresentado o processo ***CBD-Arch-ED***, proposto no contexto desta pesquisa. Este processo atende a todas as fases do processo de ED, mas com foco no detalhamento das atividades que

compreendem a fase de projeto do domínio, na qual as soluções propostas na literatura foram consideradas insatisfatórias.

O Capítulo 6 mostra o apoio oferecido ao projeto arquitetural de componentes no contexto de ED, e de métodos de DBC em geral. Durante as Seções deste Capítulo são apresentadas as falhas identificadas. As orientações existentes são bastante abstratas, tornando-se insuficientes para apoio efetivo na construção de uma arquitetura de componentes. Neste sentido, ao final do Capítulo 6, é apresentado um conjunto de alternativas para apoio ao projeto arquitetural baseado em componentes, proposto no contexto desta pesquisa, com base do processo ***CBD-Arch-ED***, descrito no Capítulo 5. Estas alternativas de pesquisa tem por objetivo gerar um arcabouço que possa, de forma concreta, apoiar o projetista na construção da arquitetura do domínio, com foco na tecnologia de componentes.

5 Processos em Engenharia de Domínio

Neste Capítulo, são apresentados os principais processos de ED existentes na literatura. Tendo em vista os problemas identificados, seja através da análise da literatura ou pelo uso efetivo dos processos, é apresentada uma proposta de processo de ED denominado ***CBD-Arch-DE***, com foco em projeto arquitetural baseado em componentes.

5.1 Processos existentes para suporte a ED

O método de ED mais conhecido e considerado o precursor dos demais métodos é o FODA. O processo do método FODA começa pela análise de contexto, onde são avaliados o escopo do domínio, suas relações com elementos externos e a variabilidade existente entre estas relações. Tais informações são modeladas através de diagramas de estrutura e fluxos de dados e, o resultado na análise de contexto é apresentado num diagrama de contextos, representando escopo e limites do domínio.

A segunda etapa é a modelagem do domínio, na qual problemas de similaridades e diferenças que norteiam o domínio são analisadas e, para isto, são gerados modelos que representam os diferentes aspectos destes problemas. A modelagem do domínio envolve três atividades: análise de *features*, modelagem de entidades e seus relacionamentos e análise funcional. Na atividade de análise de *features* orientações para identificação, categorização e validação de *features* e modelos de *features* são propostas, atendendo aos três conceitos básicos do método (generalização/especialização, agregação/decomposição, parametrização). Na atividade de modelagem de entidades/relacionamentos (E-R), modelos E-R devem capturar e definir o conhecimento do domínio que é essencial para a implementação de aplicações no domínio. A atividade de análise funcional identifica similaridades e diferenças funcionais de aplicações de um domínio. Ela abstrai e estrutura funções comuns no modelo do qual modelos funcionais específicos para uma aplicação podem ser instanciados e derivados com adaptações apropriadas (SEI, 2003). Modelos de *features* e E-R são usados para orientar o desenvolvimento do modelo funcional. *Features* obrigatórias e entidades formam a base para derivar as abstrações funcionais de um modelo. *Features* alternativas e opcionais serão contempladas durante o refinamento do modelo. Outras características que causam

diferenças funcionais são definidas e refinadas através de parametrização. Nesta atividade, diagramas de fluxos de dados e de estados são bastante utilizados para a representação das funcionalidades do domínio.

A terceira e última etapa é a construção da arquitetura do domínio que se propõe a fornecer uma solução de software para os problemas definidos na fase anterior. O modelo arquitetural sugerido pelo FODA é de alto nível, onde são identificados processos concorrentes e módulos comuns do domínio, alocando as *features*, funções e dados definidos na modelagem do domínio em termos de processos e módulos da arquitetura. Para promover a reutilização, o método sugere o uso de uma arquitetura em camadas para que impactos técnicos e trocas de requisitos do modelo possam ser melhor localizados e tratados**.** Estas camadas sugeridas tende a organizar a arquitetura em quatro camadas, onde cada uma contempla um conjunto de artefatos de projeto que satisfazem as categorias sugeridas pelo método em relação as suas *features*, conforme apresentado na Seção 4.2.1.

A partir do FODA, outros métodos de ED foram derivados. Um exemplo é o método FODAcom, que se baseia, em grande parte, na definição de processos e artefatos definidos por FODA. FODAcom adiciona a modelagem de casos de uso como o primeiro veículo para captura de requisitos do domínio, a partir da modelagem do contexto. Neste método, modelos de casos de uso e modelos de *features* não têm relação de subordinação, mas sim se complementam uns aos outros na obtenção de informações do domínio. A Figura 22 mostra a estrutura do processo de ED em FODAcom.

FORM é um outro método baseado em FODA. FORM estende as idéias de FODA, incorporando uma perspectiva de *marketing* e explorando as questões de análise e projeto a partir dessa perspectiva. É proposto um plano de marketing e produto que pode auxiliar a impulsionar o desenvolvimento das atividades de ED. O método FORM tem sido apresentado pelos autores como uma proposta focando em linha de produto, que é um conceito bastante semelhante à engenharia de domínio, mas que no âmbito comercial tem sido melhor aceita.

Em BRAGA (2000), foi proposto um processo de ED denominado Odyssey-DE (*Domain Engineering*), o qual tem o propósito de unir os aspectos de reutilização e entendimento do domínio que são providos pelos processos de ED existentes (e.g.

FODA, RSEB) e o detalhamento do desenvolvimento de componentes, provido pelos processos de DBC.

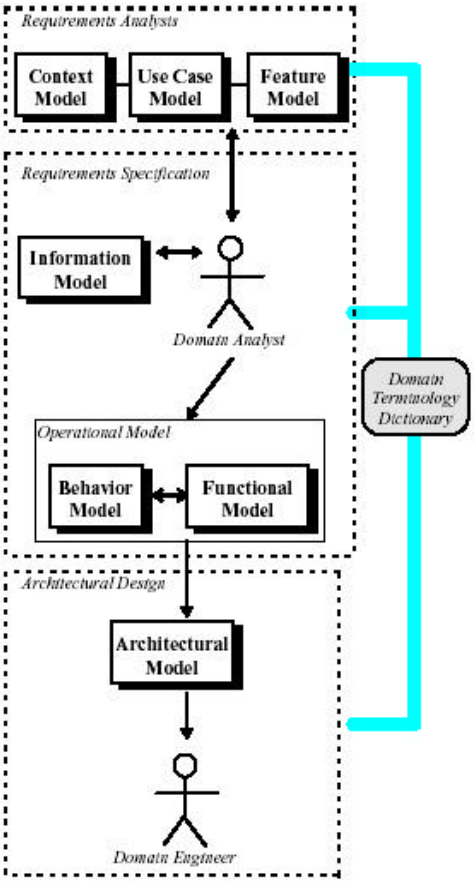

Figura 22: Método FODAcom (extraído de (VICI *et al*, 1998))

Conforme apresenta a Figura 23, o início do processo começa com a etapa de planejamento do mesmo, onde estudos de viabilidade do domínios são desenvolvidos. O planejamento de domínio deve prover um conjunto de critérios que permitam ao gerente do domínio fazer estimativas razoáveis em termos de recursos a serem utilizados, cronograma e custos (BRAGA, 2000). A análise de riscos avalia as ameaças que podem existir no planejamento efetuado (e.g, atrasos em cronogramas, aumento de custos). Na etapa de análise são definidos os conceitos do domínio, suas diferenças e similaridades. Para tal, são criados diagramas de contextos, *features*, casos de uso de domínio e, para cada caso de uso, são especificados modelos OO como, por exemplos, modelos de classes e seqüências. Nesta etapa, parte da proposta de *features* de MILER (2000) foi utilizada, não contemplando somente as *features* de projeto e implementacionais. Em TEIXEIRA

(2003), foi adicionado a este processo o diagrama de tipos de negócio e reestruturação de casos de uso, com o objetivo de dar melhor suporte a DBC nesta fase. Na fase de projeto, são identificados e projetados os elementos importantes do domínio, analisados na etapa anterior, adicionando características arquiteturais aos mesmos, com objetivo de gerar componentes de projeto. Nesta atividade, modelos de componentes, modelos arquiteturais, especificação de interfaces deverão ser construídos. Por último, a etapa de implementação do domínio, prevê que componentes sejam implementados e também componentes legados sejam avaliados para possível utilização no domínio.

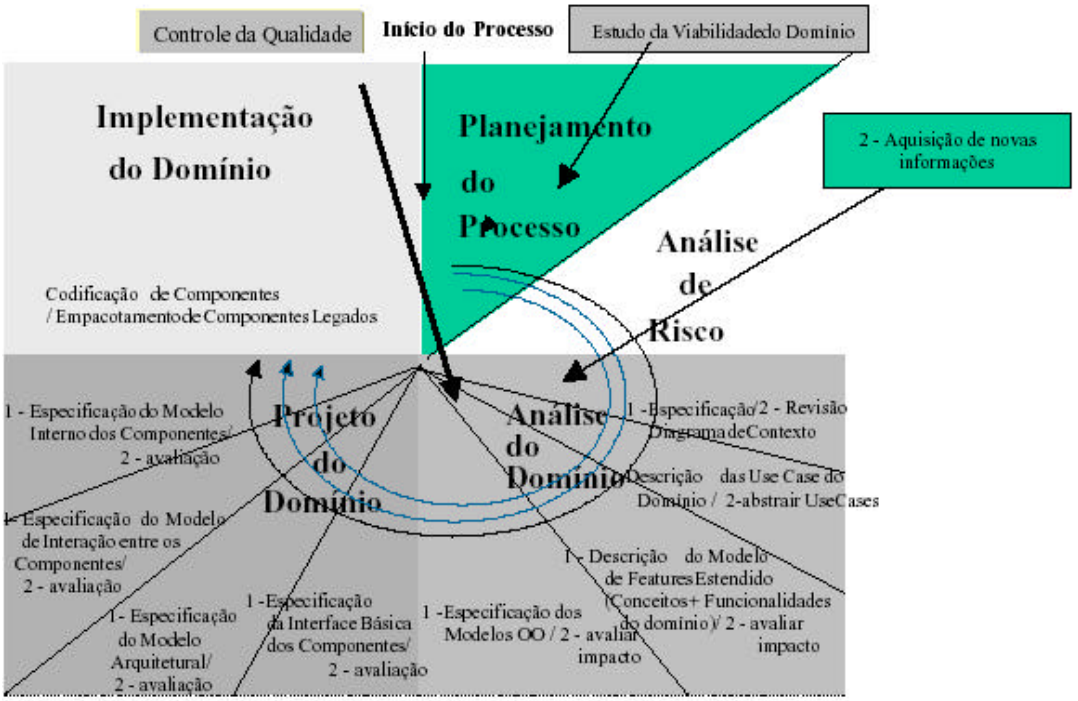

Figura 23: Método de ED Odyssey-DE (extraído de BRAGA (2000))

MILER (2000) propõe um processo para Engenharia de Aplicações que é análogo ao de ED proposto por BRAGA (2000). No entanto, para reutilizar os artefatos do domínio, o engenheiro de aplicações deve identificar, a partir do modelo de *features* do domínio, quais serão aquelas reutilizáveis no contexto da aplicação. Depois de selecionadas as *features* a serem reutilizadas, as próprias *features* e os demais artefatos relacionados a elas serão também reutilizados. Os aspectos relacionados a reutilização da ED neste processo de EA estão melhor discutidos em MILER (2000).

5.1.1 Considerações sobre os Processos de ED

Embora a Engenharia de Domínio possua algumas fases bem definidas, é importante destacar a existência de métodos que conduzam o processo de construção de artefatos, apoiando suas relações, restrições, e a forma como o conhecimento do domínio, expresso através destes artefatos, pode ser disponibilizado aos seus reutilizadores.

Apesar de existir uma tendência em vincular o método FODA como ponto de partida para a ED, os métodos dele desencadeados apresentam propostas para atender, ou a algumas lacunas existentes do método original (e.g. Odyssey-DE, para apoio a DBC), ou para adaptar-se a contexto específicos (e.g. FODAcom, para atender as necessidades de uma empresa de Telecomunicações Italiana).

Esta tendência se deve em grande parte a forma como o método FODA suporta o processo de levantamento dos requisitos do domínio, o qual se dá em essência por meio de *features*. Estas abstrações de alto nível são o marco referencial do método FODA e, nas propostas de métodos nele baseadas, é unânime o uso de modelo de *features* como forma de expressar as similaridades e variabilidades do domínio.

Métodos com o RSEB, que surgiu depois de FODA e que propôs a adoção de casos de uso para obter informações do domínio, com o tempo, seus autores propuseram uma versão do método com o uso de *features*, denominado FeaturRSEB. Como já comentado anteriormente, no FeaturRSEB, casos de uso e *features* possuem papel complementar uns aos outros, sem relação de subordinação.

Uma característica comum aos métodos de ED é o foco atribuído à atividade de análise definida no processo de ED. As publicações existentes atribuem bastante foco na forma como contextos são definidos, e sobretudo na identificação das *features* do domínio e aspectos como similaridade, variabilidades, tipo de relação, etc, expressos através de seus modelos. Não é dado maior destaque aos outros modelos que fazem parte da análise de domínio, provavelmente pelo fato de existir um foco mais dirigido na criação de rastros da fase de análise para o projeto através das *features* do domínio. Os rastros têm por objetivo determinar as relações existentes entre artefatos que tratam de problemas comuns ao domínio, os quais representam estes problemas em diferentes níveis de abstração (e.g. *feature* e um componente de projeto), ou também numa visão diferente dos problemas num nível de abstração semelhante (e.g. *feature* e caso de uso).

Outro fator que impede uma melhor avaliação dos métodos de ED para apoio às atividades de projeto é a distância existente entre as *features* do domínio, e o conjunto de artefatos de projeto (e.g. classes, componentes). Existe uma lacuna identificada nos métodos e que se reflete no conjunto de artefatos e seus respectivos rastros para um determinado domínio. A forma como os processos de ED são conduzidos para chegar até os artefatos de projeto gerados não é claramente descrita.

O foco atribuído à fase de análise e em especial aos modelos de *features* do domínio também pode ser justificado pelo fato de *features* serem os principais artefatos considerados para a reutilização do domínio em futuras aplicações decorrentes. Por exemplo, na infra-estrutura de reutilização Odyssey, todos os artefatos de análise e projeto do domínio modelado têm rastro para as suas *features* correspondentes. Neste sentido, quando um Engenheiro de Software deseja especificar uma aplicação a partir de um domínio já modelado no Odyssey, este deverá selecionar quais são os contextos e as *features* do domínio que serão reutilizados na sua aplicação. A partir desta escolha, o Engenheiro de Software terá disponível todos os artefatos que estão relacionados às *features* escolhidas (e.g, casos de uso, tipos do negócio, classes, componentes), inclusive outras *features* as quais aquelas escolhidas tenham alguma relação de dependência, detectado durante o processo de reutilização apoiado pela infra-estrutura.

Odyssey-DE é uma proposta de método que preenche algumas lacunas existentes em métodos de ED e em especial ao método FODA. Em Odyssey-DE, a etapa de planejamento do domínio é proposta para que somente após um estudo de viabilidade é que se dê início a fase de análise de domínio. Segundo BRAGA (2000), nem todos os domínios precisam participar de um processo de ED para que depois possam ser efetuadas as EA decorrentes. Para domínios com características peculiares e com processos com alto nível de complexidade, sugere-se o uso de ED, ou seja, suporte ao entendimento do domínio e construção de artefatos reutilizáveis. Ao contrário, onde as soluções de software são pequenas e de fácil entendimento, sugere-se o uso direto de EA. Apesar desta orientação, nada impede que seja construída a modelagem de domínio para tais soluções de software, sendo esta uma decisão da equipe envolvida.

Outra lacuna preenchida por Odyssey-DE é o suporte a DBC, com o foco em projeto baseado em componentes. Embora BRAGA (2000) apresente um conjunto de

atividades que compõem a fase de projeto baseado no processo Odyssey-DE, alguns problemas foram identificados com o uso da infra-estrutura de reutilização Odyssey, os quais serão discutidos no Capítulo 6. Os outros métodos como FODA, FORM e RSEB, quando descrevem a fase de projeto de ED, apresentam indícios de suporte a construção de componentes. No entanto, tal constatação fica prejudicada pela falta de foco a esta fase da ED na literatura disponível.

No que se refere às soluções de software para apoio aos métodos de DBC, Odyssey-DE parece ser o único método que tem suporte a maioria das atividades de modelagem de ED. Entretanto, a fase de projeto no Odyssey ainda possui algumas lacunas as quais devem ser atendidas no contexto desta proposta de tese. Tais lacunas são discutidas na Seção 6. Quanto aos outros métodos, FORM possui uma ferramenta de modelagem denominada *FORM Case Tool*, a qual dá apoio a modelagem de *features*. Os demais métodos basicamente utilizam ferramentas de modelagem OO como Rational Rose, adotando mecanismos de extensão e estereótipos para representar os artefatos sugeridos.

Em função das restrições ferramentais, o apoio ao projeto arquitetural de domínio é praticamente inexistente. O Odyssey permite criar uma arquitetura de componentes, mas com suporte reduzido e voltado somente a representação de componente e interfaces requeridas e fornecidas. Necessidades de apoio ao projeto arquitetural como os propostos nas Seções 5.2 e 6.2, não são providos pelos métodos de ED estudados. A Tabela 4 mostra um visão geral dos aspectos discutidos acima a respeito dos métodos de ED. Para os itens onde não foi possível fazer uma avaliação precisa pela falta de informação na literatura existente, convencionou-se a sigla NA (Não Avaliado).

Tabela 4: Principais Características para Comparação de Métodos de ED

Método /Característica	Odyssey-DE	FODA	FORM	FODAcom	FeaturRSEB
Uso de *Features*	Sim. Padrão FODA estendido	Sim	Sim. Padrão FODA	Sim. Padrão Próprio	Sim. Padrão FODA estendido
Foco	Análise	Análise	Análise	Análise	Análise
Apoio a DBC	Em parte	Não	NA	NA	NA
Apoio ao Projeto de ED	Em parte	NA	Em parte	NA	NA
Ferramental para Projeto Arquitetural	Em parte	NA	NA	NA	NA

5.2 Processo de DBC no Contexto de ED – CBD-Arch-DE

Tendo em vista os problemas destacados na Seção 5.1.1 com relação ao suporte ao projeto arquitetural dos métodos de ED, em especial no uso da tecnologia de componentes, e também aqueles destacados na Seção 2.4, referente ao suporte oferecido pelos métodos de DBC, é proposto nesta monografia o processo ***CBD-Arch-DE***.

Conforme dito anteriormente, embora BRAGA (2000) tenha proposto um processo de ED para componentes, observou-se que o resultado deste trabalho obteve maiores resultados voltados à fase de análise de domínio, como em geral acontece com os métodos de ED, e o conjunto de atividades previstas no projeto, não foram efetuadas por completo. Além disso, a forma como o projeto arquitetural de componentes foi proposto não permite uma especificação mais detalhada dos componentes e interfaces, envolvendo a arquitetura que cada componente pode conter internamente, e a forma como seus elementos internos, componentes e/ou classes, se comunicam entre si e com componentes externos que fazem parte do domínio modelado.

No processo proposto, o qual denominamos de ***CBD-Arch-DE***, são apresentadas as atividades de todas as fases de ED previstas, dando ênfase às questões arquiteturais que apresentam problemas nas outras soluções de processo para ED com DBC, e sobretudo contemplando a visão macro proposta por BROWN (2000), conforme discutida na Seção 2.2.

Na infra-estrutura de reutilização Odyssey, existe uma ferramenta para modelagem e acompanhamento de processos, denominada Charon (MURTA, 2002), a qual foi utilizada para modelar ***CBD-Arch-DE***. A máquina de processo Charon prevê que atividades sejam modeladas através de processos primitivos (p) e compostos (c). Os processos compostos agrupam um conjunto de processos primitivos e estes últimos detalham a atividade que vão desenvolver, o tempo gasto, quais serão os artefatos consumidos, os papéis das pessoas que farão parte da atividade e quais serão os artefatos produzidos. Cada processo primitivo deve ser devidamente descrito para guiar as pessoas que o executarão para o seu desenvolvimento correto, conforme previsto na modelagem do processo.

O processo ***CBD-Arch-DE*** inicia com a atividade de planejamento do domínio. Tal atividade deve ser desenvolvida de forma análoga à proposta do processo Odyssey-DE. Conforme afirma BRAGA (2000), a etapa de planejamento do domínio tem por objetivo prover um conjunto de critérios que permitam ao gerente do domínio (especialista e engenheiro) fazer estimativas razoáveis em termos de recursos a serem utilizados, custos e cronograma. Para tal, são previstas atividades como estudo de viabilidade do domínio, estimativa de recursos requeridos para a realização da engenharia de domínio e estimativa dos custos do projeto. No contexto deste processo, é também prevista a atividade de análise de riscos, onde as ameaças ao planejamento podem ser identificadas, bem como soluções para tentar minimizar atrasos de cronograma, aumento de custos, etc.

Concluída a atividade de planejamento e verificada a viabilidade de execução da Engenharia de Domínio, dá-se início às atividades de análise, projeto e implementação do domínio, de forma interativa, ou seja, são previstos retornos às fases anteriores para refinamento do domínio. A Figura 24 mostra o processo ***CBD-Arch-DE*** definido na Charon e a Tabela 5 descreve brevemente seus processos primitivos e compostos.

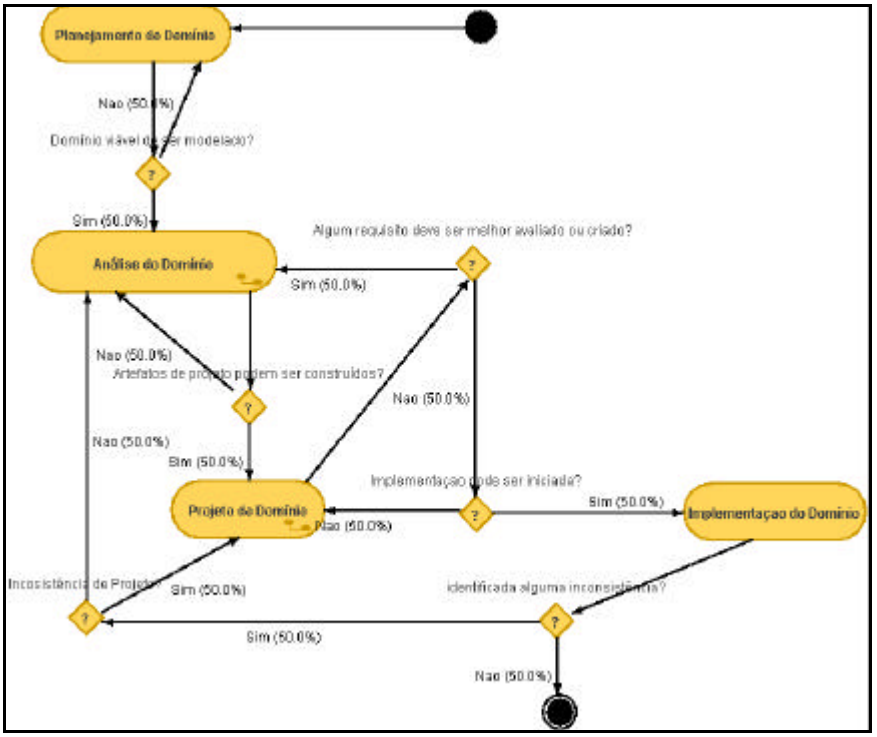

Figura 24: Processo ***CBD-Arch-DE*** definido na Charon

Tabela 5: Processos Principais do ***CBD-Arch-DE***

Processo	Tipo P /C	Descrição Geral
1. ***CBD-Arch-DE***	C	Compreende os processos 2, 3, 4 e 5 enumerados nesta tabela.
2. Planejamento do Domínio	P	Efetua estimativa de custos, recursos e cronograma, além da viabilidade de modelagem do domínio.
3. Análise do Domínio	C	Compreende os processos 6, 7, 8 e 9 da Tabela 6.
4. Projeto do Domínio	C	Compreende os processos 10, 11, 12, 13, 14, 15 e 16 da Tabela 7.
5. Implementação do Domínio	C	Implementação dos componentes gerados pelo projeto, usando uma tecnologia de componentes.

A atividade de Análise de Domínio compreende um processo composto na Charon. Este processo composto começa pela atividade de identificação dos contextos. Esta consiste na definição dos principais conceitos do domínio em termos de sub-áreas (contextos) existentes. O engenheiro de domínio, juntamente com o especialista, deve identificar quais são os diferentes contextos do domínio sendo modelado, bem como as relações existentes entre os mesmos. A partir dessa atividade, o especialista do domínio determinará quais contextos serão atendidos na modelagem do domínio subseqüente.

A atividade posterior é a identificação das *features* do domínio. Nesta atividade são criadas as *features* do domínio e os modelos de *features* para representar as relações existentes entre as mesmas. Nesta proposta de processo, é adotada a abordagem de *features* proposta por MILER (2000), no que se refere à adoção das *features* conceituais e funcionais, e a abordagem de KANG, LEE e DONOHOE (2002) para representar *features* relativas a ambiente operacional, tecnologias do domínio e técnicas implementacionais. Tais decisões se devem ao fato de que, no caso da abordagem de MILER (2000), a mesma encontra-se implementada na infra-estrutura de reutilização Odyssey, onde esta proposta de tese será efetivamente desenvolvida. No caso da proposta de KANG *et al.* (1993), o objetivo é dar melhor apoio a criação futura de componentes de suporte e de apoio à infra-estrutura do domínio, os quais não estão diretamente ligados ao negócio do domínio, mas que de alguma forma fazem parte de sua arquitetura.

Tal atividade deve ser desenvolvida por especialistas e engenheiros do domínio, que irão obter as *features* do domínio através dos contextos já identificados na atividade anterior, e também através de entrevistas com usuários, análise de documentos e sistemas

legados. Como resultado desta atividade se obtém modelos de *features*, os quais devem refletir as reais características existentes e necessárias para o domínio.

Após a identificação das *features*, duas atividades podem ocorrer em paralelo e de forma complementar.

Uma atividade é a definição dos tipos de negócio, ou seja, os artefatos de análise que representam aspectos estáticos do domínio, geralmente candidatos à persistência no contexto da atividade de projeto. Geralmente, os tipos de negócio estão ligados a uma *feature* do tipo conceitual, apresentando inicialmente um mapeamento um para um, isto é, um tipo de negócio para uma *feature*, mas, derivações sobre os tipos de negócio podem vir a ter vários tipos de negócio associados a uma *feature*. *Features* e tipos de negócio representam diferentes níveis de abstração, uma vez que o último prevê a identificação de seus atributos, ou seja, uma forma de tornar o artefato menos abstrato como se propõem as *features*.

A outra atividade é a definição dos casos de uso associados às *features*. Estes casos de uso podem estar relacionados a várias *features*, e também a um conjunto de tipos de negócio. Casos de uso podem descrever situações de criação, consulta e outras operações possíveis de serem feitas sobre tipos de negócio e seus atributos. À medida que tipos de negócio são criados, casos de uso destes tipos podem vir a ser definidos. Por outro lado, a definição de casos de uso pode dar início ao processo de criação de tipos de negócio ainda não construídos. Em função deste processo iterativo, é que estas atividades podem vir a ocorrer em paralelo e de forma cooperativa, como mostra a Figura 25. Quando ambas atividades forem concluídas, é que a atividade de análise de domínio será dada como encerrada. No entanto, especialista, engenheiro e projetista de domínio podem retornar para a fase de análise, caso seja necessário, conforme representado na Figura 24. A Tabela 6 mostra uma breve descrição dos processos primitivos do processo composto análise de domínio.

A atividade de projeto de domínio envolve a criação, composição e geração de arquitetura de componentes e suas interfaces, como mostra a Figura 26. A atividade de criação de componentes, como mostra a Figura 27, prevê que sejam gerados diagramas de seqüência, baseado nos tipos de negócio e descrição dos casos de uso, com o intuito de identificar possibilidades de criação de componentes. A proposta de geração de

componentes a partir de estilos arquiteturais baseados em tipos e baseados em instâncias, apresentada por TEIXEIRA (2003), poderia ser aplicada neste contexto. No entanto, outras *features* do domínio (e.g., *features* do ambiente operacional, tecnologia do domínio e técnicas implementacionais) não possuem a mesma sistemática de geração de seus componentes, e portanto, a criação de componentes para estas *features* pode ser avaliada através de um diagrama de seqüência, com instâncias destas *features* e de outros artefatos que interagem num caso de uso previsto na análise de domínio.

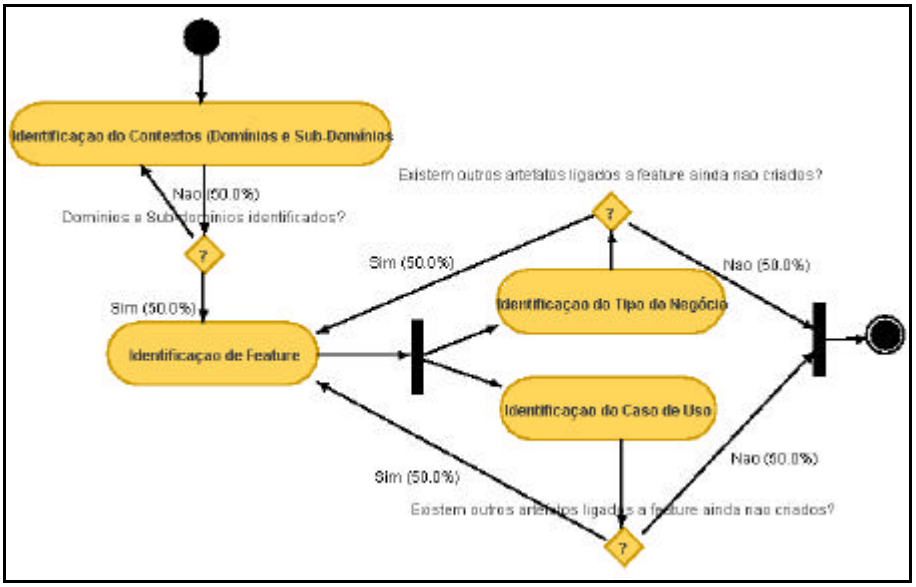

Figura 25: Processo de Análise de Domínio em ***CBD-Arch-DE***

Tabela 6: Processos da Atividade de Análise de Domínio no ***CBD-Arch-DE***

Processo	Tipo P /C	Descrição Geral
6. Identificação de Contextos	P	Identifica os diferentes contextos de um determinado domínio e quais daqueles serão contemplados na modelagem.
7. Identificação de *Features*	P	Levantamento das características do domínio, as quais representam elementos conceituais, funcionais e de apoio ao domínio.
8. Identificação de Tipos de Negócio	P	Representam as características estáticas do domínio, candidatas a persistência e que estão associadas a uma *feature* do domínio
9. Identificação de Casos de Uso	P	Representam as funcionalidades do domínio, ou seja, situações de utilização dos tipos do negócio. Casos de uso podem estar associados a vários tipos de negócio e *features*.

Independente do processo adotado para a geração de componentes e interfaces, internamente, devem ser especificados o conjunto de classes que os implementam, seus métodos, atributos e relacionamentos. Este modelo de classes, gerado para cada componente e suas interfaces, pode fazer uso de suporte já existente a padrões de projeto, proposto por Dantas *et al.* (DANTAS *ET AL.*, 2002) e suporte a instanciação de padrões arquiteturais, proposto por XAVIER (2001), ambos no contexto da infra-estrutura de reutilização Odyssey.

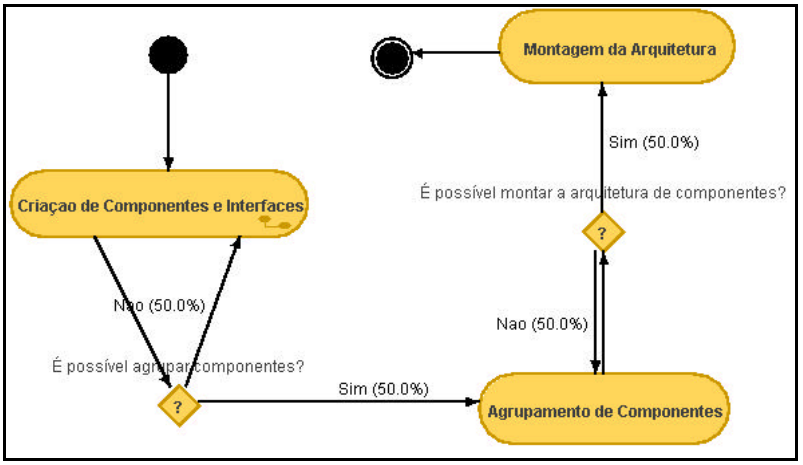

Figura 26: Projeto de Domínio no Processo ***CBD-Arch-DE***

Para avaliar as interações entre as classes de um componente, são gerados diagramas de seqüências, os quais também apontam referências as interfaces providas e requeridas pelo componente aos demais existentes no domínio. A Figura 27 mostra as atividades que envolvem a criação de um componente e suas interfaces, no contexto de projeto no processo ***CBD-Arch-DE*.**

Após a criação de componentes e interfaces, deve ser dado início a atividade de agrupamento de componentes. Tendo em vista o alto grau de acoplamento e coesão existente num conjunto de componentes, os quais internamente possuem uma estrutura de classes fortemente ligada as classes de outro componente, sugere-se que uma avaliação sobre a possibilidade de agrupamento destes componentes possa ser conduzida pelo projetista. Justifica-se o emprego de agrupamento de componentes devido a grande quantidade de interações que ocorrem num grupo seleto, possibilitando dessa forma uma representação única da microarquitetura formada pelo agrupamento, ou seja, através de

uma única interface provida e fornecida, este agrupamento poderá comunicar-se com os demais componentes e agrupamentos de componentes existentes numa arquitetura. Caberá à interface que representa este agrupamento de componentes, comunicar-se com as interfaces mais internas para coordenar as atividades atribuídas a cada componente interno. Esta preocupação com relação à comunicação de interfaces é aplicável, tanto para aquelas fornecidas pelo componente, quanto para as requeridas.

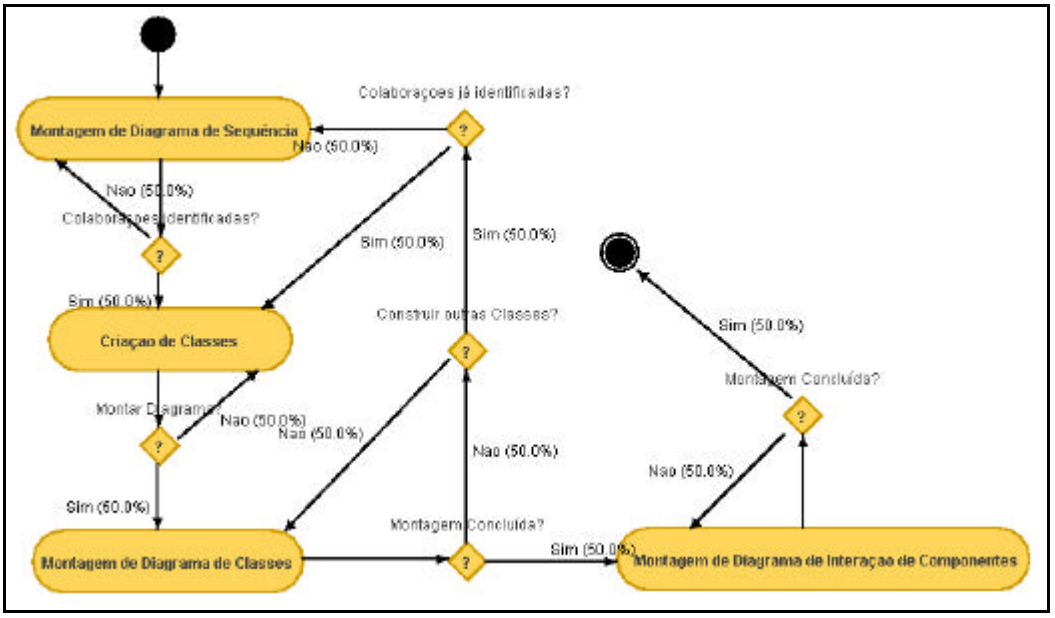

Figura 27: Criação de Componente e Interface no Projeto de Domínio em ***CBD-Arch-DE***

Algumas técnicas de agrupamento de componentes foram sugeridas em TEIXEIRA (2003) para gerar componentes de negócio a partir de seus tipos, conforme apresentado nas seções 3.2.1.1 e 3.2.1.2. Por exemplo, no caso de dois tipos de negócio terem uma relação de generalização/especialização, nesta proposta é gerado um único componente com duas interfaces providas, uma para cada tipo que compõe o componente de negócio, bem como os métodos padrão existentes em cada interface específica.

No entanto, tais técnicas podem vir a ser aplicadas para avaliar a possibilidade de agrupamento em nível de componente, ou seja, um componente de mais alto nível representando um grupo de componentes que possuem alto grau de acoplamento. Estas sugestões de agrupamento e de uso de técnicas de apoio são discutidas na Seção 6.2.

Com o agrupamento de componentes concluído, já existem algumas tendências na estrutura da arquitetura de componentes do domínio, as quais devem ser refinadas pelo

projetista na atividade de montagem da arquitetura. Nesta atividade o uso de estilos arquiteturais poder ser viável para organizar a arquitetura de componentes do domínio, tendo a possibilidade de melhor atender a requisitos não funcionais como desempenho, concorrência, etc. A Figura 26 representa o processo iterativo das atividades de criação de componentes e interfaces, agrupamento de componentes e montagem de arquitetura. A Tabela 7 mostra todos os processos primitivos e compostos da Charon para a atividade de projeto de domínio no ***CBD-Arch-DE***.

Tabela 7: Processos da Atividade de Projeto do Domínio no ***CBD-Arch-DE***

Processo	Tipo P /C	Descrição Geral
10. Criação de Componentes	C	Compreende os processos 11, 12, 13 e 14 desta tabela
11. Montagem de Diagrama de Seqüência	P	Representação das interações entre classes de análise em geral, apoiando a construção de classes de componentes
12. Criação de Classes	P	Criação das classes de componentes, com atributos, métodos e relacionamento com as demais classes
13. Montagem do Diagrama de Classes	P	Representação da dinâmica existente entre as classes e a interação que uma classe de um componente deve ter uma classe de outro componente
14. Montagem do Diagrama de Interação do Componente	P	Instanciação das classes do componente para avaliação de sua interação, sugerido pelos casos de uso da análise
15. Agrupamento dos Componentes	P	Avaliação das possibilidades de agrupamento de componentes, gerando microarquiteturas internas
16. Montagem da Arquitetura	P	Montagem efetuada, com orientações baseadas em estilos arquiteturaIs, possibilitando a geração de uma descrição arquitetural padronizada.

Por último, a atividade de implementação do domínio, na qual componentes de terceiros poderão ser utilizados para a implementação de alguns componentes do domínio, ou então implementa-los a partir do zero, numa linguagem que dê suporte ao desenvolvimento baseado em componentes. Como é esperado que a arquitetura gerada na fase anterior esteja devidamente descrita através de uma linguagem de descrição arquitetural, possivelmente a xADL, espera-se que tal descrição possa apoiar a implementação do domínio, sendo os desenvolvedores responsáveis em seguir a proposta arquitetural sugerida. A xADL é uma forte candidata pelo fato de apresentar mecanismos de extensão que não foram detectados em outras ADL's para DBC (e.g. CDL, ABC/ADL).

5.3 Considerações Finais

Neste Capítulo foram analisadas as principais abordagens de processo de ED obtidas na literatura, identificados seus pontos falhos e, em conseqüência desta análise, é apresentada uma proposta inicial de um processo de ED – ***CBD-Arch-DE***.

O ***CDB-Arch-DE*** apresenta as seguintes contribuições se comparado aos demais processos de ED:

- Propõe um detalhamento das atividades que compreendem a etapa de projeto de domínio;
- Está disponível para qualquer equipe de Engenharia de Domínio através de um ambiente de reutilização (Odyssey – SDE) já existente e de domínio público;
- Utiliza uma infra-estrutura para a gerência e instanciação de processos como a máquina Charon (MURTA, 2003), na qual o processo ***CDB-Arch-DE*** pode ser avaliado na prática e, em conseqüência, os resultados obtidos com a sua utilização num domínio qualquer, e
- Explora tecnologias ligadas a DBC para suporte a representação de arquiteturas de componentes (estilos e linguagens de descrição arquitetural).

Para avaliar se o processo efetivamente atende às expectativas de apoio esperadas, está sendo prevista uma atividade de experimentação, a qual é brevemente discutida na Seção 7.3.

Conforme relatado ao longo deste Capítulo e na Seção 4.4, a atividade de projeto associada ao processo ***CDB-Arch-DE*** é a principal contra-partida desta pesquisa para as áreas de ED e DBC. Tendo em vista sua importância, o Capítulo 6 analisa o que a literatura tem apresentado em termos de apoio ao projeto arquitetural baseado em componentes e, ao final, as questões a serem atendidas nesta proposta de tese em função dos problemas identificados nestas áreas.

6 Projeto Arquitetural baseado em Componentes na Engenharia de Domínio

Neste capítulo são discutidas as soluções já existentes para apoio ao projeto arquitetural, os aspectos positivos e negativos do suporte oferecido e, ao final, é apresentado um conjunto de alternativas de pesquisa para a construção efetiva de uma proposta de projeto arquitetural baseada em componentes, relatada brevemente na Seção 5.2, no processo ***CDB-Arch-DE***.

6.1 Propostas de Apoio ao Projeto Arquitetural

O projeto arquitetural de um sistema tem como objetivo representar a semântica existente entre seus artefatos, a qual será levada em consideração na atividade de implementação. Este processo arquitetural consiste de um processo de conversão de um conjunto de requisitos em uma arquitetura de software que preencha, ou, no mínimo, facilite o preenchimento dos requisitos (BOSCH, 2000). O projetista de um sistema pode desenvolver o projeto arquitetural explorando o uso de estilos arquiteturais, padrões arquiteturais e de projeto, *frameworks*, linguagens de descrição arquitetural, etc, de forma coordenada, com o intuito de melhor descrever a semântica existente no sistema, com uma representação clara dos artefatos envolvidos, suas responsabilidades com relação ao conjunto de dados que está gerenciando e a forma como ocorre a troca de informações entre os artefatos.

Durante a análise da literatura referente a DBC e a Engenharia de Domínio, foi possível observar que as propostas existentes para apoio ao projeto arquitetural baseado em componentes são provenientes de alguns **métodos de DBC,** (e.g. Kobra, UML Components) e de **métodos de ED** (e.g. Odyssey-DE) com foco na tecnologia de componentes. Nesta análise, também foi possível observar duas vertentes conceituais nas quais as propostas existentes permeiam.

A primeira é que as propostas de apoio ao projeto arquitetural estão contextualizadas num processo de ED, mas que o projeto arquitetural em si é baseado em algum método de DBC específico. Por exemplo, o Odyssey-DE é uma proposta de método de ED, que propõe algumas atividades para projeto arquitetural de componentes,

com forte base no método UML Components para a tarefa de especificação de requisitos e geração de componentes de negócio e de processo. No entanto UML Components não oferece apoio efetivo para a atividade de projeto arquitetural de componentes.

A segunda vertente é com relação ao conceito de Linha de Produto (LP) que tem aparecido desde o início dos anos 90, onde suas propostas de pesquisa têm dado maior destaque a atividade de projeto arquitetural se comparado aos métodos de ED analisados até o momento. Desde então, comparações são feitas com relação a LP e ED, e métodos com foco em DBC para LP têm sido propostos na literatura (ATKINSON *et al.*, 2002) (BOSCH, 2000).

Muito se tem escutado a respeito do conceito de desenvolvimento baseado em linha de produtos, o qual tem por objetivo disponibilizar um conjunto de artefatos que fazem parte de uma linha de produto, e que por sua vez podem ser reutilizados na construção de vários produtos que pertençam àquela linha.

A introdução deste conceito tem gerado algumas discussões na comunidade de Engenharia de Software a respeito do que o conceito de LP tem de diferente da ED (POULIN, 1997). Até o momento, poucas justificativas e não muito convincentes foram apresentadas. Uma delas é que a linha de produto tem um foco na área industrial, enquanto a engenharia de domínio um foco mais acadêmico. No entanto, esta é uma questão puramente de marketing, não justificando as diferenças conceituais e metodológicas que tentam impor nas duas propostas. Outra justifica é com relação ao nível de abstração, onde se diz que na ED são gerados artefatos de mais alto nível do que aqueles gerados para linha de produto. Tal justificativa não pode ser avaliada, uma vez que até o momento não foi apresentado na literatura um conjunto de artefatos de um mesmo domínio gerados em ED e também em Linha de Produto, possibilitando que tais comparações pudessem ser avaliadas.

Não é objetivo nesta Seção discutir as diferenças e/ou semelhanças conceituais entre linha de produto e engenharia de domínio, mas identificar que as propostas existentes para projeto arquitetural em LP, também podem ser aplicadas em ED.

Nas subseções que se seguem serão descritas e analisadas três propostas de apoio ao projeto arquitetural que efetivamente contribuíram para a proposta apresentada na Seção 6.2, sendo elas:

1) A atividade de projeto proposta no processo Odyssey-DE, como representante de uma abordagem de ED com foco em DBC (BRAGA, 2000);
2) A proposta de método para projeto de arquiteturas de software para linha de produto (BOSCH, 2000);
3) O método de DBC Kobra com foco em linha de produto (ATKINSON *et al.*, 2002) (ATKINSON *et al.*, 2001), o qual se propõe a melhor especificar o projeto arquitetural dos componentes, se comparado aos demais estudados (e.g. UML Componentes, Catalysis).

6.1.1 Projeto Arquitetural de Componentes em ED – Odyssey-DE

Pode-se observar no Capítulo anterior que várias propostas de processo descrevem algumas orientações ao projeto arquitetural. Grande parte dos processos dão maior ênfase à forma como os artefatos de análise serão utilizados no contexto de projeto de domínio, para que possa então ser construída sua arquitetura. Tal levantamento é de extrema importância para o suporte aos requisitos analisados. No entanto, a construção efetiva da arquitetura do domínio e o suporte que pode ser oferecido para sua construção, também, devem ser igualmente atendidos para que o resultado da etapa de projeto possa efetivamente atender às expectativas da implementação.

Dentre os processos de ED analisados no Capítulo anterior, o Odyssey-DE é o que se propõe a melhor detalhar a atividade de projeto arquitetural do domínio. Os demais (FODA, FORM, FODAcom, FeatuRSEB) apresentam linhas gerais do que venha a ser desenvolvido no projeto arquitetural de componentes do domínio. No caso do FODA, como ele tem origem no paradigma estruturado, o significado do conceito de componente atribuído pelos autores não é o mesmo previsto pela área de DBC. Em FODA, componente representa um subsistema do domínio, um conjunto de procedimentos e de aplicativos gerados durante a ED e obtidos por terceiros.

BRAGA (2000) propõe a atividade de projeto do domínio no Odyssey-DE centrada na definição arquitetural do domínio, e no detalhamento dos modelos internos dos componentes, baseado nos artefatos gerados na fase de análise. Para tal, a autora define as seguintes fases:

Fase 1: definição dos componentes e suas interfaces: esta fase baseia-se na análise dos modelos de casos de uso e nos tipos de negócio. Os tipos de negócio vão representar os elementos estáticos do projeto arquitetural, os quais serão expressos por meio de componentes. Os casos de uso vão basicamente determinar as relações entre os componentes a serem gerados (a partir dos tipos de negócio), as quais serão providas através de interfaces. A análise dos casos de uso vai apoiar a geração tanto das interfaces fornecidas por um componente, como as requeridas, levando em conta a necessidade de composição entre componentes. A identificação dos componentes e suas interfaces possibilitam a geração de um modelo de componentes que representa a definição dos serviços disponibilizados ao meio externo e não os serviços internos de cada componente. Este modelo gerado dá início a segunda fase.

Fase 2: definição de um modelo arquitetural geral de colaboração entre os componentes. Nesta fase, além da interação entre os componentes do domínio, questões arquiteturais como persistência, processamento distribuído, etc, devem ser levadas em consideração. A estas questões é atribuído o nome de interfaces arquiteturais que determinam o conjunto de serviços necessários para a implementação de um dado estilo arquitetural (e.g. camadas). Estas interfaces arquiteturais são muitas vezes induzidas pela própria forma de interação entre os componentes do domínio, isto é, recomendado para aplicações geradas/existentes naquele domínio. Neste sentido, o estilo arquitetural de uma arquitetura é muito mais influenciado pela aplicação dependente de domínio do que pelo próprio domínio. Além do estilo arquitetural, nesta fase devem ser identificados os componentes que dão suporte à arquitetura dos componentes de domínio. Estes componentes de suporte podem dar apoio a questões como comunicação distribuída, banco de dados centralizado, transação, etc, e podem ser construídos pela equipe de implementação ou adquiridos de terceiros.

Fase 3: definição do projeto interno dos componentes. Nesta fase, todos os modelos de análise relacionados a um dado componente são refinados, mais precisamente os tipos que participam do componente, seus atributos e métodos, e a forma de interação entre os tipos já existentes ou criados no próprio componente. Para melhor modelar estes tipos internos de um componente, é sugerido a consulta a uma base de padrões de projeto. Como os tipos gerados na análise podem participar de mais de um componente de

projeto, desempenhando diferentes papéis, podendo dar origem a mais de um componente de projeto, é necessário que o modelo de colaboração de componentes arquiteturais represente os tipos internos ao componente que participam da sua interface. Este modelo de colaboração é representado em duas visões, onde a primeira mostra a interação entre componentes e a segunda a interação interna dos tipos que compõem um componente. Como resultado desta fase, são gerados modelos de interfaces dos componentes, modelo arquitetural de colaboração entre componentes do domínio, modelos de classes e diagrama de estados de cada componente do domínio e um modelo arquitetural com duas visões.

6.1.1.1 Considerações sobre a Proposta de Projeto Arquitetural no Odyssey-DE

O apoio ao projeto arquitetural em ED proposto por BRAGA (2000) atinge em parte seu objetivo. Dentre as fases citadas acima somente a primeira efetivamente é apoiada pelo ambiente de reutilização Odyssey, dentro do qual o apoio é oferecido. Conforme citado anteriormente, o modelo de componentes gerado possibilita a geração de componentes e suas interfaces, não permitindo que a especificação interna desses componentes possa ser executada, conforme previsto na fase 3.

Outro aspecto é com relação ao suporte oferecido para geração dos componentes, disponibilizado no trabalho de TEIXEIRA (2003) no contexto do Odyssey-DE. Apesar de serem utilizadas regras de acoplamento dos tipos de negócio para geração de componentes de negócio, em casos de domínios muito grandes, a quantidade de componentes gerados tende a ser elevado, incluindo também componentes de processo, de suporte e infra-estrutura. A proposta de projeto arquitetural de BRAGA (2000) não apresenta soluções para agrupamento de componentes, construção de microarquiteturas, etc., visando o tratamento destes problemas. Neste sentido, a arquitetura de componentes do domínio poderá vir a ser de difícil compreensão, gerência e sobretudo, complicada a avaliação no uso de estilos arquiteturais genéricos (SHAW e GARLAN, 1996) para apoio a organização da arquitetura de componentes do domínio.

Se observa também que não existe uma preocupação com relação a forma como o projeto interno dos componentes proposto na Fase 3 é conduzido. Esta tarefa pode gerar uma série de inconsistências no modelo (e.g. referente a acoplamento de classes,

repetição de classes em diferentes acoplamentos, como requisitos de um componente estarão expressos em termos de classes, etc.) as quais não são tratadas nesta proposta.

Por último, se observa também que a proposta de BRAGA (2000) não disponibiliza suporte arquitetural para viabilizar a implementação da arquitetura de componentes gerada. Este suporte arquitetural poderia vir através de uma ADL. A partir de sua especificação, a arquitetura poderia ser implementada por alguma tecnologia de componentes (e.g. EJB, COM).

Conforme apresentado nos Capítulos 4 e 5, o foco do trabalho de BRAGA (2000) é realmente atribuído a fase de análise de domínio. O suporte ao projeto arquitetural baseado em componentes previsto no processo Odyssey-DE foi desenvolvido no contexto do trabalho TEIXEIRA (2003). Neste sentido, algumas orientações previstas no processo Odyssey-DE foram somente em parte atendidas (Fase 1).

6.1.2 Projeto Arquitetural em Linha de Produto (BOSCH, 2000)

Em (BOSCH, 2000), é apresentada uma abordagem de projeto arquitetural de software para linha de produto, o qual se propõe a também contemplar uma avaliação explícita ao projeto de atributos de qualidade, ou seja, requisitos não-funcionais. Como em todos os métodos, o projeto arquitetural começa pela especificação dos requisitos, o qual é utilizado como fonte para o projeto arquitetural baseado em funcionalidades. Este projeto arquitetural gerado é considerado uma primeira versão da arquitetura, a qual, após sua primeira versão, sofrerá transformações com o intuito de atender os atributos de qualidade. A cada atributo de qualidade adicionado a arquitetura, uma transformação da mesma é gerada, ou seja, uma nova versão. Este processo ocorre repetidamente até que todos os atributos de qualidade identificados na especificação de requisitos sejam atendidos, ou até que o projetista perceba que não existe uma solução exeqüível. A Figura 28 mostra uma visão geral sobre como ocorre o projeto da arquitetura na proposta de BOSCH (2000), contemplando requisitos funcionais e não funcionais. As atividades das elipses são brevemente discutidas a seguir.

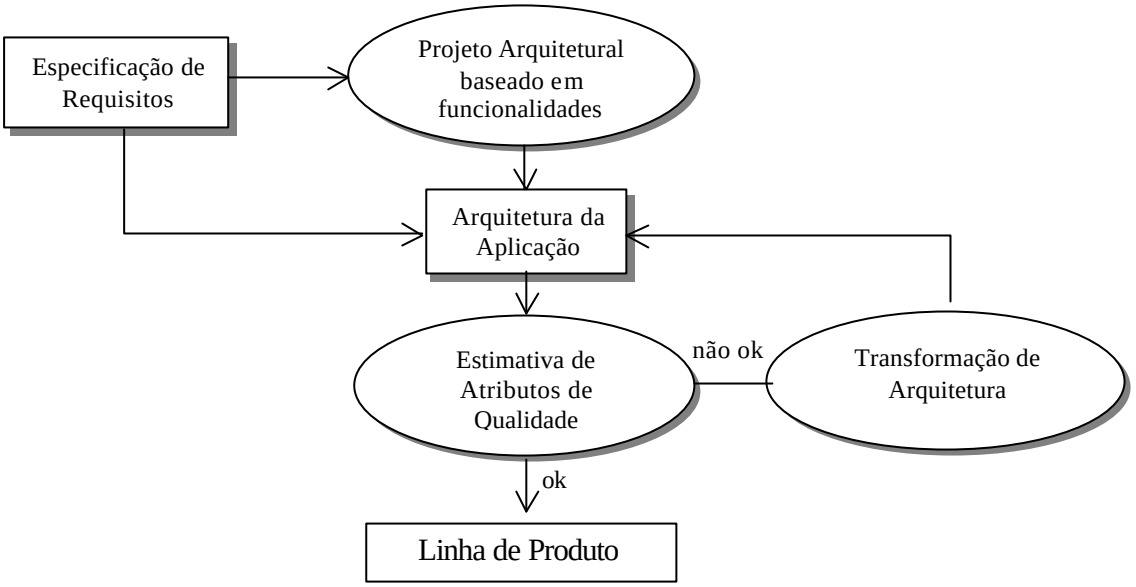

Figura 28: Método de Projeto Arquitetural para Linha de Produto - extraído de (BOSCH, 2000)

O projeto arquitetural baseado em funcionalidades compreende quatro passos: o primeiro tem por objetivo identificar o contexto do sistema sobre o qual o projeto será executado, as interfaces deste sistema com entidades externas e a forma como ocorre esta interação.

O segundo passo consiste na identificação do que ele denomina *archetypes*. *Archetypes* representam as entidades arquiteturais de um sistema, abstrações principais que geralmente são consideradas estáveis. As atividades que envolvem a identificação dos *archetypes* são:

- Identificação dos *archetypes* candidatos;
- Seleção de pequenos conjuntos de *archetypes* dentre os candidatos com intuito de identificar a exclusão de alguns e agrupamentos de outros que estão bastante acoplados;
- Identificação e seleção de relações entre *archetypes*.

No processo posterior de transformação arquitetural, existe uma forte tendência em construir estruturas adicionais sobre os *archetypes* para atender aos requisitos de qualidade adicionados a arquitetura.

BOSCH (2000) cita que o processo de identificação destas entidades arquiteturais está relacionado aos métodos de análise de domínio (KANG *et al.*, 1993), mas que

algumas diferenças podem ser identificadas. A primeira diferença refere-se à forma como as abstrações são capturadas. Embora *archetypes* possam ser modelos como objetos do domínio (e.g. *features*), a identificação destes *archetypes* não é feita imediatamente num domínio de aplicação. Ao invés disso, eles são resultado de um processo criativo no qual, após análises sucessivas de várias entidades de um domínio, as propriedades mais relevantes são consideradas entidades arquiteturais, e modelos são gerados para representar a interação entre os *archetypes*. A segunda diferença entre projeto arquitetural no contexto de linha de produto e análise de domínio é que a arquitetura de um sistema geralmente cobre múltiplos domínios.

O terceiro passo compreende a decomposição do sistema baseada em seus componentes principais e a identificação do relacionamento entre eles. A decomposição de um sistema em componentes não precisa ser num único nível, podendo contemplar dois ou mais níveis recursivos de decomposição para partes críticas do sistema. Para apoiar o processo de identificação e seleção de componentes, algumas atividades são sugeridas como:

- Identificação das interfaces de cada componente de um sistema;
- O domínio ao qual o componente está associado;
- As camadas de abstração que podem existir para os componentes identificados;
- Identificação das entidades do domínio ao qual o componente pertence;
- Instanciação dos *archetypes*.

O último e quarto passo do projeto arquitetural baseado em funcionalidades é a descrição das instâncias do sistema, usando os *archetypes* e as interfaces do sistema.

A arquitetura obtida a partir das atividades que envolvem o projeto arquitetural baseado em funcionalidade será o ponto de partida para a avaliação dos atributos de qualidade, os quais também foram obtidos na etapa de especificação de requisitos. São sugeridas quatro abordagens para a avaliação de atributos de qualidade:

- Avaliação baseada em cenários, onde um conjunto de cenários é proposto para que se possa concretizar o significado do atributo de qualidade na arquitetura já existente.

- Simulação, onde uma seqüência de ações pode ser executada para avaliar atributos de qualidade. Neste caso, os principais componentes de uma arquitetura são implementados e outros componentes são simulados, resultando num sistema executável. Para alguns atributos de qualidade (e.g. desempenho), a simulação complementa a avaliação baseada em cenários.
- Modelagem matemática, que é considerada como uma alternativa à simulação. Modelos matemáticos podem ser usados para avaliar atributos de qualidade operacional, onde por exemplo, para uma determinada simulação, possam ser estimados os requisitos computacionais necessários para a execução de diferentes seqüências numa arquitetura.
- Avaliação baseada em experiência, onde engenheiros de software com suas experiências prévias podem contribuir na avaliação das melhores alternativas de tratamento de atributos de qualidade. Apesar de parecer uma avaliação subjetiva, baseada em intuição e experiências, tais sugestões não devem ser subestimadas.

Com o resultado do processo de avaliação dos atributos de qualidade, pode-se proceder transformações arquiteturais, onde as trocas efetuadas na arquitetura gerada como resultado da atividade de projeto arquitetural baseado em funcionalidades serão efetivamente executadas, contemplando os atributos de qualidade da arquitetura construída. Este processo pode ocorrer de forma iterativa, ou seja, duas ou mais transformações podem ser necessárias para atender completamente aos atributos de qualidade da arquitetura. Para que o processo de transformação ocorra de forma gradativa, cinco categorias de transformação são propostas, sendo elas:

1) a imposição de um estilo arquitetural;
2) imposição de padrões arquiteturais;
3) a aplicação de padrões de projeto;
4) a conversão de requisitos de qualidade em soluções funcionais, estendendo a arquitetura com funcionalidades não específicas do domínio mas usadas para atender melhor a estas funcionalidades e,

5) o uso do princípio de dividir para conquistar onde os atributos de qualidade são distribuídos em subsistemas e componentes que fazem parte do sistema.

6.1.2.1 Considerações sobre a Proposta de Projeto Arquitetural em Linha de Produto (BOSCH, 2000)

Esta abordagem de projeto arquitetural apresenta uma característica diferente da proposta do Odyssey-DE pelo fato de apresentar preocupações no tratamento de requisitos não funcionais em linha de produto. Requisitos não funcionais são suportados numa arquitetura de linha de produto através de um processo iterativo de transformação arquitetural, como descrito anteriormente.

Embora tenha apresentado uma preocupação que não tinha sido explicitamente contemplada nas propostas de processo e apoio arquitetural nos métodos de ED, a descrição do projeto arquitetural é ainda de muito alto nível. O projeto arquitetural, como sugerido pelo autor, não pode se restringir a somente identificar interfaces, componentes e suas respectivas camadas. O que se espera de um projeto arquitetural é um detalhamento desta identificação e sobretudo a semântica atribuída a estes elementos arquiteturais. A literatura que explora estudos de caso desta proposta de projeto arquitetural também ilustra um projeto arquitetural de alto nível, sem qualquer tipo de apoio da UML para a especificação dos artefatos gerados. No entanto, do ponto de vista de reutilização e sobretudo para apoio a implementação, a especificação de cada elemento arquitetural é imprescindível.

No que tange o uso de estilos arquiteturais e padrões de projeto, o autor sugere uma imposição no uso destas abordagens. A forma como o autor impõe o uso destas tecnologias não é tratado na literatura disponível, dificultando uma avaliação mais precisa quanto ao uso destas abordagens na proposta de projeto arquitetural. Ainda com relação a estas tecnologias, o autor sugere que as mesmas sejam levadas em consideração na etapa de transformação arquitetural, momento em que são adicionados os requisitos não funcionais da linha de produto. No entanto, tais abordagens devem também ser consideradas para apoio ao projeto dos componentes que atendem aos requisitos funcionais da linha de produto, apoiando na melhor especificação dos *archetypes* que, conforme mencionado anteriormente, é um ponto fraco da abordagem arquitetural.

Por último, esta proposta até o momento não possui suporte ferramental para que possa ser experimentada. Os estudos de caso explorados na literatura não mencionam este tipo de suporte. Sem um ferramental de apoio a construção dos artefatos sugeridos e, sobretudo, um processo que oriente a execução da proposta de projeto arquitetural, torna-se difícil identificar as vantagens que venham a existir nesta abordagem. O esforço no uso desta abordagem em ambientes genéricos para apoio a modelagem de componentes (e.g. Rational Rose) é muito grande e sobretudo incompatível para representação de artefatos com propostas específicas (e.g. *archetypes*).

6.1.3 Projeto Arquitetural no Método de DBC Kobra

Dentre os métodos de DBC apresentados na Seção 2.3, o Kobra é aquele que mais detalha o suporte ao projeto arquitetural de seus componentes, com orientações a respeito de suas especificações internas. Tendo em vista esta preocupação, é que será dada ênfase à avaliação do Kobra no contexto desta Seção, sobretudo as orientações atribuídas para a construção de componentes.

Conforme apresentado na Seção 2.3.3, o elemento principal do Kobra é o que ele denomina de *komponent*. Este *komponent* é construído através de um processo de especificação e realização, contemplando a engenharia do *framework* (linha de produto), e reutilização dos artefatos derivados do *komponent*, através da engenharia da aplicação (produto instanciado).

O método Kobra tem uma preocupação forte com a questão da arquitetura interna de seus *komponents*. Cada *komponent* especificado é composto de:

- diagramas da UML para representação da sua visão estrutural (e.g. diagrama de classes, objetos), comportamental (e.g. diagrama de estados);
- documentos textuais descrevendo modelos de decisão[4] referente à execução interna do *komponent*;
- especificação funcional que descreve os efeitos externamente visíveis por outros *komponents* da linha de produto.

[4] Modelo de Decisão – descreve as características de um komponent para os diferentes membros de uma linha de produto. Identifica pontos de variação e soluções correspondentes em termos de efeitos que uma solução pode gerar na especificação e realização do *komponent*.

ATKINSON *et al.* (2002) apresentam um conjunto de atividades usadas para desenvolver a especificação dos artefatos acima. Para cada tipo de diagrama, são apresentadas tais orientações.

No que se refere ao diagrama de classes de um *komponent*, os autores sugerem, em linhas gerais, os seguintes passos:

- uso de estereótipos para identificar *komponents* existentes no diagrama de classes;
- adicionar atributos lógicos para todas as classes;
- adicionar associações essenciais entre as classes;
- comparar modelos funcionais com modelos comportamentais disponíveis;

Para a modelagem funcional, a qual compreende um conjunto de especificações de operações para cada operação de um *komponent*, são recomendados dentre outros, os seguintes passos:

- identificar operações do *komponent* para examinar todas as mensagens enviadas para outros *komponents*;
- para cada operação identificar:
    - tipos de parâmetros e tipos de retorno;
    - efeitos esperados pela operação;
    - formular resultados e cláusulas a serem atendidas;
    - etc.
- para cada operação, criar regras para envio, retorno, leitura e troca de informações.

O mesmo tipo de orientação é atribuído para os demais modelos que atendem tanto a especificação quanto à realização de *komponents*.

Com o uso destas orientações, os autores afirmam que os *komponents* gerados atenderão a quatro princípios básicos esperados de uma arquitetura de componentes Kobra, sendo eles:

- Princípio da uniformidade, onde conceitos devem ser manipulados e representados de forma semelhante;
- Principio da localidade, onde é sugerido o uso de subsistemas para organizar a arquitetura de *komponents*;
- Princípio da parcimônia, no qual parte da premissa que os diagramas devem conter o mínimo de informações necessárias para atender as idéias requeridas;
- Por último, princípio de encapsulamento, o qual sugere que um *komponent* contemple a sua especificação, realização e implementação.

6.1.3.1 Considerações sobre o Apoio ao Projeto Arquitetural de Kobra

Dentre as propostas de apoio ao projeto arquitetural de componentes, o método de DBC Kobra para Linha de Produtos é o único que apresenta um conjunto de orientações que norteiam o processo de especificação e realização de komponents. Embora tais orientações sejam interessantes, não se tem certeza que as mesmas irão convergir para as premissas de componentes Kobra apresentadas acima.

Kobra propõe um conjunto de diagramas que compreendem a especificação e realização de um *komponent*. No entanto, não fica claro como ocorre o processo de construção destes diagramas, quando um tipo dá fomento a construção do outro. O conjunto de informações registradas em especificações textuais, como modelos de decisão, não são de fácil compreensão e, em conseqüência, não se sabe como as informações ali registradas estão sendo consideradas nos demais diagramas que compõem um *komponent*.

Observa-se uma forte ênfase do método na construção dos *komponents*. A arquitetura geral da linha de produto modelada fica desprovida de orientações pontuais que norteiam a ligação de *komponents*, com suporte apropriado (e.g. padrões arquiteturais, estilos arquiteturais).

Por último, em nenhum momento foi apresentado pelo método o uso de linguagens de descrição arquiteturais para a arquitetura de *komponents* gerada pelo método Kobra. Em geral, o método dá pouca ênfase a eventuais contribuições no uso de tecnologias como padrões, *frameworks* e estilos arquiteturais para suporte a especificação de seus componentes.

6.2 Proposta de Projeto Arquitetural baseado em Componentes: alternativas de pesquisa

As propostas analisadas na Seção 6.1 pecam em geral por dois grandes motivos: ou por efetivamente não darem suporte a grande parte do projeto arquitetural de componentes, ou porque as propostas de especificação para a construção de arquiteturas de componentes são descritas de forma muito genérica, tornando difícil sua utilização.

Esta proposta de apoio ao projeto arquitetural de componentes tem por objetivo minimizar alguns problemas ainda existentes, seja nos métodos de DBC, seja nas propostas de apoio arquitetural para ED ou LP. Tais problemas são:

- Falta de clareza com relação à especificação detalhada de componentes, bem como o uso de orientações que apóiem este processo;
- Falta de clareza quanto ao uso de tecnologias de padrões de projeto, padrões arquiteturais, estilos arquiteturais, no contexto da definição de componentes do domínio e na arquitetura de componentes como um todo;
- Falta de orientação no processo de organização de uma arquitetura de componentes em termos de microarquiteturas[1], as quais efetivamente possam representar sub-domínios específicos.
- Não é apresentada em quaisquer das abordagens estudadas o uso de uma linguagem de descrição arquitetural, a qual possa servir de suporte ao processo de implementação da arquitetura.
- Em geral, falta clareza quanto ao processo que conduz a construção da arquitetura de componentes.

Os exemplos de emprego dos métodos de DBC disponíveis na literatura apresentam problemas do tipo: o domínio que foi explorado na fase de definição do contexto é diferente daquele usado na definição da arquitetural; o processo de refinamento dos elementos arquiteturais que resultam na arquitetura de uma aplicação não é claro; não existem diretrizes claras para apoiar este processo de refinamento, ou seja, é necessária uma percepção voltada às experiências do projetista ao invés de uma proposta de solução arquitetural proveniente do método em si, e há uma falta de ambientes que dêem suporte aos métodos para melhor avalia-los, sob todos os aspectos, e

sobretudo quanto a rastreabilidade entre os artefatos de diferentes níveis de abstração e apoio a reutilização.

Quanto a Engenharia de Domínio, observa-se pelas análises efetuadas nos Capítulos 4 e 5 e na Seção 6.1.1, um foco muito grande no apoio a fase de análise de domínio. Alguns métodos propõem um conjunto de atividade para o projeto de domínio mas que na literatura é praticamente inexplorada, pois os exemplos se restringem, em grande maioria, em mostrar componentes gerados a partir de um artefato de análise, sem tratar as questões arquiteturais do componente e dele em relação aos demais previstos na arquitetura. No entanto, boa parte da literatura de ED restringe-se a mostrar o processo envolvido especificamente na análise, uso de seus artefatos, o que representam, o tratamento que pode ser atribuído a cada um, dentre outras informações.

Assim, em nossa proposta de projeto arquitetural, pretende-se:

- fazer uso dos benefícios que a ED já possui consolidado, sobretudo para as fases iniciais da modelagem de um domínio, com intuito de obter informações para a construção e especificação de componentes;
- fazer uso dos benefícios que a tecnologia de DBC provê no que diz respeito aos métodos de DBC, as abordagens de arquiteturas para componentes, e os princípios sobre o qual um artefato componente está baseado;
- refinar o apoio ao projeto arquitetural de componentes existente na literatura, sugerindo heurísticas para apoio ao projeto.
- adotar esta proposta focada em projeto arquitetural no contexto de uma infra-estrutura de reutilização (e.g. Odyssey), viabilizando a avaliação desta proposta em caráter prático.

Durante a Seção 5.2, foi apresentado o processo ***CBD-Arch-DE*** e na Figura 26 o detalhamento da atividade de projeto de domínio, que é o foco desta proposta de tese. Estas atividades, no contexto da descrição do processo, estão apresentadas de forma genérica. No entanto, todas elas têm um grau de complexidade menor ou maior, o qual poderá ser melhor conduzido pelo projetista de domínio se este utilizar algumas técnicas que dêem suporte as suas atividades. Nas seções subseqüentes, pretende-se discutir para

cada atividade definida no projeto do domínio em ***CBD-Arch-ED***, algumas propostas já existentes na literatura para sua solução, como poderiam ser utilizadas no contexto de projeto arquitetural para componentes em ED, e que tipo de apoio já é oferecido na infra-estrutura Odyssey, onde esta proposta de pesquisa possivelmente será implantada.

6.2.1 Criação de Componentes e Interfaces

6.2.1.1 Contexto da Atividade

A atividade de criação de componentes e suas interfaces parece ser simples, uma vez que, de forma minimalista, podemos pensar que se trata somente de criar o componente e suas interfaces providas e requeridas e conectá-lo aos demais componentes de acordo com os requisitos da aplicação. No entanto, tal atividade não pode ser vista de maneira tão simplificada, pois cada componente requer uma especificação para atender seus requisitos e que envolve, a princípio:

1. identificação das classes que internamente contemplam o componente, seus atributos e operações;
2. identificação do relacionamento que uma classe interna a um componente tem com outra classe do próprio componente;
3. identificação das necessidades que um componente possui, provenientes de outros componentes, para execução de suas responsabilidades, isto é, seus métodos;
4. construção das interfaces providas e requeridas do componente.

6.2.1.2 Soluções Existentes para Criação de Componentes e Interfaces

Existem propostas para a geração de componentes apoiadas em princípios de DBC (BROWN, 2000) (BISCOE *et al.*, 2002) (LÜER e ROSENBLUM, 2001) (LEE *et al.*, 1999) (TEIXEIRA, 2003). Como a proposta de projeto arquitetural que se pretende construir será implantada na infra-estrutura de reutilização Odyssey, foi necessário avaliar o apoio já oferecido e melhorar as lacunas existentes em projeto arquitetural que já foram identificadas nas Seções posteriores dentro desse Capítulo. Na infra-estrutura Odyssey, existe apoio a criação de componentes de negócio e de processo, gerados a partir dos tipos de negócio e casos de uso, respectivamente. Embora os componentes

sejam oferecidos ao projetista, não existe uma especificação detalhada e é neste sentido que esta proposta pretende preencher esta lacuna.

6.2.1.3 Soluções Consideradas para esta Pesquisa

A especificação tanto dos componentes de negócio como para os de processo deve contemplar as quatro atividades enumeradas na Seção 6.2.1.1. Para isso, é sugerido o uso de modelos para orientar esta identificação, e algumas possíveis técnicas de apoio a projeto OO, aplicadas aqui para a especificação de componentes.

No que se refere a modelos, é proposto o uso de diagramas de seqüência, os quais instanciam os tipos de negócio envolvidos em cada caso de uso e sugere métodos ou informações de interação entre os tipos. Foi escolhido o diagrama de seqüência em função da sua proposta interativa, onde se pode avaliar de forma mais detalhada as relações existentes entre os elementos instanciados. Este diagrama de seqüência na fase de projeto, deverá fazer um maior detalhamento de operações específicas ao contexto do domínio, com intuito de estender o conjunto de operações básicas que já podem ser especificadas nos casos de uso. O diagrama de seqüência como um todo irá refinar o componente de processo já gerado. Tendo como base um componente de negócio gerado a partir de um tipo de negócio, o diagrama de seqüência será utilizado como veículo para:

- identificar as classes internas de um componente de negócio, derivadas de uma ou mais instâncias do diagrama de seqüência;
- refinar o conjunto de atributos definidos na fase de análise e que no projeto devem ser melhor detalhados;
- criar os métodos de classe, baseado naqueles utilizados para a sua instância.

O uso combinado da proposta de TEIXEIRA (2003) no que se refere à descrição de passos de casos de uso através de tipos de negócio, com o uso dos diagramas de seqüência para cada caso de uso gerado, proposto no processo ***CBD-Arch-DE***, poderá mais efetivamente apoiar na identificação do conjunto de classes que compreende um componente, além de sugerir métodos de cada classe e interações entre elas.

Com os devidos refinamentos, o resultado da combinação das duas propostas até o momento auxilia somente as atividades de criação de classes de um componente, do

relacionamento interno destas e a construção de suas interfaces providas e requeridas. Por outro lado, provavelmente, existem muitas inconsistências, dentre as possíveis:

1) Existem classes que pertencem a mais de um componente?
2) Caso existam como tratar esta questão, ou seja, é possível dois ou mais componentes terem internamente uma mesma classe com papéis diferentes?
3) Caso seja possível, por quais dos componentes os dados desta classe será persistido?
4) Quando um método de classe precisa utilizar os serviços provenientes de uma classe que pertence a um outro componente, como viabilizar tal interação?
5) Como será a política de geração de interfaces providas e requeridas? Será através de uma única de cada tipo (requerida ou provida) ou poderão ser geradas tantas quantas o projetista julgar necessário?

Para melhor atender a estas inconsistências e outras que venham a aparecer, algumas áreas de pesquisa e aplicação em projeto orientado a objetos merecem ser avaliadas. Até o momento foram identificadas:

1. O uso de padrões de projeto, como por exemplo àqueles propostos por GAMMA *et al* (1994), para a estruturação das classes internas de um componente. Padrões são uma tecnologia onde experiências recorrentes são documentadas e validadas pela comunidade de padrões e que têm sido empregadas com resultados satisfatórios;
2. O uso de métricas para projeto OO, como as apresentadas em (ANDRADE, 1998) (CHO *et al.*, 1998) (YACOUB, AMMAR e ROBINSON, 1999) , onde propostas para a avaliação de acoplamento estático e dinâmico de objetos, profundidade da hierarquia entre as classes, tamanho das classes, coesão, dentre outras, foram analisadas e utilizadas para contextos específicos. Acredita-se que com o uso de métricas OO poder-se-ia minimizar ou solucionar grande parte dos problemas identificados anteriormente, principalmente para aqueles referentes ao acoplamento de objetos que são mais claramente identificados no contexto de especificação de componentes.

3. Como a UML 2.0 dá suporte a especificação de componentes e de que forma esta proposta pode se adequar a esta notação padrão (UML, 2003) .

Após a avaliação rigorosa destas e de outras soluções para projeto OO para o contexto de especificação de componentes, pretende-se organizar todas as questões levantadas a respeito de criação de componentes e de suas interfaces, e disponibilizar este conhecimento ao projetista do domínio através de um conjunto de heurísticas.

Estas heurísticas deverão orientar, inicialmente, a especificação de componentes de negócio e de processo os quais já apresentam um grau de complexidade elevado a ser minimizado pelo uso correto das orientações. Embora não se possa afirmar com precisão, é provável que este conjunto de heurísticas possa ser generalizado para apoio à construção dos demais componentes (e.g. utilitários e de infra-estrutura) de uma arquitetura (BROWN, 2000). Espera-se que este conjunto de **heurísticas para especificação de componentes de negócio e de processo** a ser proposto possa melhorar a orientação de projetistas na atividade de construção do modelo interno de classes de um componente, indicando propostas concretas para a solução dos problemas até então enumerados nesta Seção.

Uma forma de avaliar se a especificação dos componentes foi bem projetada é através da construção de um diagrama de interação sugerido pelo processo ***CBD-Arch-DE*** para os componentes gerados. Para isso, será necessário um refinamento dos casos de uso da análise para o projeto, uma vez que a descrição de seus passos será feita na fase atual, através das interfaces dos componentes que viabilizam o uso das operações existentes nos métodos de classes internas de cada componente. A partir desses casos de uso de projeto, é que serão criados os diagramas de interação (seqüência ou colaboração), um para cada caso de uso. Além de mostrar pura e simplesmente interações entre componentes, através de suas interfaces providas e requeridas, pretende-se que estes diagramas gerem informações ao projetista do domínio para a próxima atividade a ser desenvolvida, isto é, o agrupamento de componentes.

6.2.2 Agrupamento de Componentes

6.2.2.1 Contexto da Atividade

Através da análise dos diagramas de interação gerados ao final da atividade de criação de componentes, o projetista pode observar a existência de grupos de componentes que interagem com freqüência. Esta freqüência de interação pode ser por vários motivos, dentre os quais se destacam:

- **Os componentes cooperam para a solução de um mesmo problema**. Se pegarmos o exemplo de componentes de negócio *conta* e *cliente* de um domínio bancário, uma conta tem que ser de algum cliente do banco e, portanto, todas as operações efetuadas sobre as interfaces do componente conta, em algum momento, terão que requerer informações sobre cliente. O lado contrário também poderá ser válido.
- **Componente de processo tem maior predominância de interações com um determinado componente de negócio**. Por exemplo, também no domínio bancário, um componente de processo para gerência de conta. Este componente de processo, provavelmente, tem uma quantidade de operações que predominam a interação com o componente de negócio conta do que com outros que pertence ao mesmo domínio.

Com o intuito de apoiar a atividade de estruturação da arquitetura de componentes do domínio e também de identificar as áreas de concentração de um domínio refletidas no projeto arquitetural, é que se propõe no contexto desta pesquisa o agrupamento de componentes. Este grupo de componentes deve ter objetivos que se complementam e também devem refletir um conjunto de requisitos do domínio que estão relacionados com esta freqüência.

6.2.2.2 Soluções Existentes para utilização nesta Pesquisa

Neste momento, está em estudo o tipo de artefato gerado na arquitetura em função da atividade de agrupamento. No entanto, existem algumas perspectivas de representação que serão avaliadas, dentre elas:

- **Alternativa 1**: Se dois ou mais componentes forem agrupados, um destes terá uma relação de composição com os outros. Por exemplo, os componentes C1, C2 e C3 da Figura 29. Observe a estrutura original e o resultado do agrupamento segundo esta alternativa.

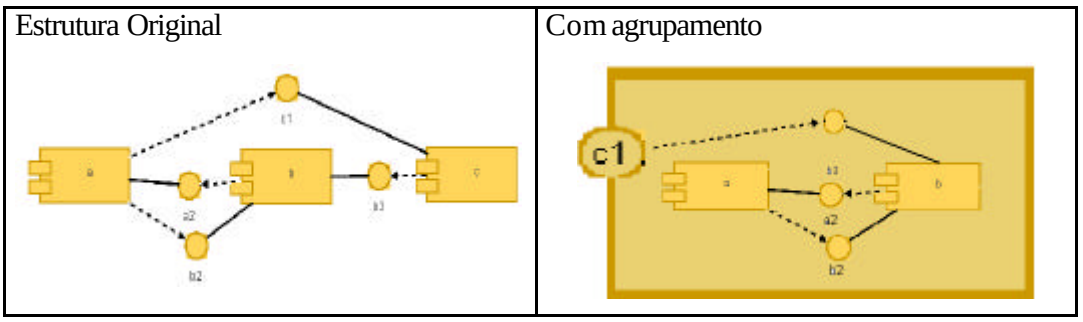

Figura 29: Alternativa 1 de agrupamento de componentes

- **Alternativa 2**: Propor um artefato do tipo microarquitetura que então seja composto pelos componentes agrupados. Por exemplo, os mesmos componentes C1, C2 e C3. Observe a estrutura original e o resultado do agrupamento segundo esta alternativa, conforme Figura 30.

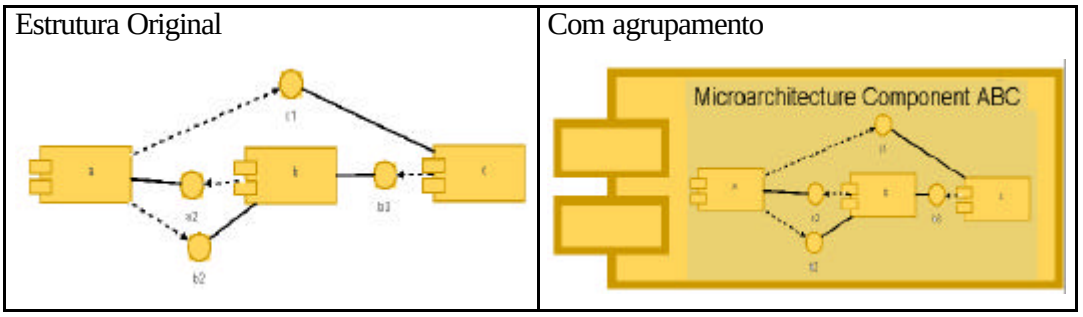

Figura 30: Alternativa 2 de agrupamento de componentes

A **Alternativa 1** é baseada na proposta de especificação de arquiteturas de componentes do ambiente *Arch Studio* (HOEK, 2003), que é um ambiente desenvolvido pelo grupo de Engenharia de Software na Universidade da Califórnia, o qual se propõe a especificar em alto nível arquiteturas de componentes baseado na abordagem de linha de produto. Várias ferramentas sobre versionamento de componentes, comparação entre

arquiteturas para a avaliação de suas diferenças, etc, foram desenvolvidas pelo grupo, e as que interessam mais diretamente para este trabalho são a xADL, como já apresentada na Seção 3.3.1.3 e no contexto desta Seção, a ferramenta *Ménage*. Esta ferramenta permite que componentes, interfaces providas e requeridas e conectores sejam criados pelo projetista. Este, então, primeiramente associa os componentes as suas interfaces requeridas e fornecidas. Após, ele poderá fazer composição de componentes, onde um dos componentes será composto por outros. Os conectores poderão ser empregados para situações onde os componentes precisam se comunicar através de suas interfaces, mas isso não foi projetado inicialmente. Então, este conector serve como um adaptador para viabilizar a comunicação dos componentes. A Figura 31, mostra um exemplo de um componente de sistema de entretenimento, sendo composto de vários outros componentes, com interfaces requeridas e fornecidas, bem como conectores.

Esta proposta de composição de componentes sugere que um componente represente uma abstração maior daqueles que o compõem, e este tipo de percepção nem sempre é trivial de se obter, em especial num contexto de Engenharia de Domínio. No entanto, caso esta alternativa seja adotada, algumas situações deveriam ser analisadas para a composição correta dos componentes, entre elas:

- Como fica representada a especificação do componente que agrupa os demais? Cada um mantém a sua especificação interna, cabendo àquele que agrupa especificar formalmente que ele está agrupando componentes e que suas interfaces requeridas e fornecidas devem ter esta informação para que o restante da arquitetura reconheça este agrupamento?
- Como se dá a especificação de conectores para prover a comunicação entre componentes que possuem interfaces que não haviam previsto troca de mensagens?
- E como seria orientado este processo de agrupamento?

Estas e outras respostas devem ser objetos desta pesquisa.

Por outro lado, tem-se a **alternativa 2**, onde é sugerida a criação de um outro componente, que então agrupa internamente todos os componentes que são freqüentemente interativos, o qual poderia receber um estereótipo de <<microarchitecture

component>>, por exemplo. Este componente gerado teria que de alguma forma expressar aos outros que não fazem parte da sua arquitetura interna, a respeito da existência de seus componentes. Isso deveria ser através de suas interfaces requeridas e fornecidas, que provavelmente sejam compostas pelas interfaces dos componentes agrupados.

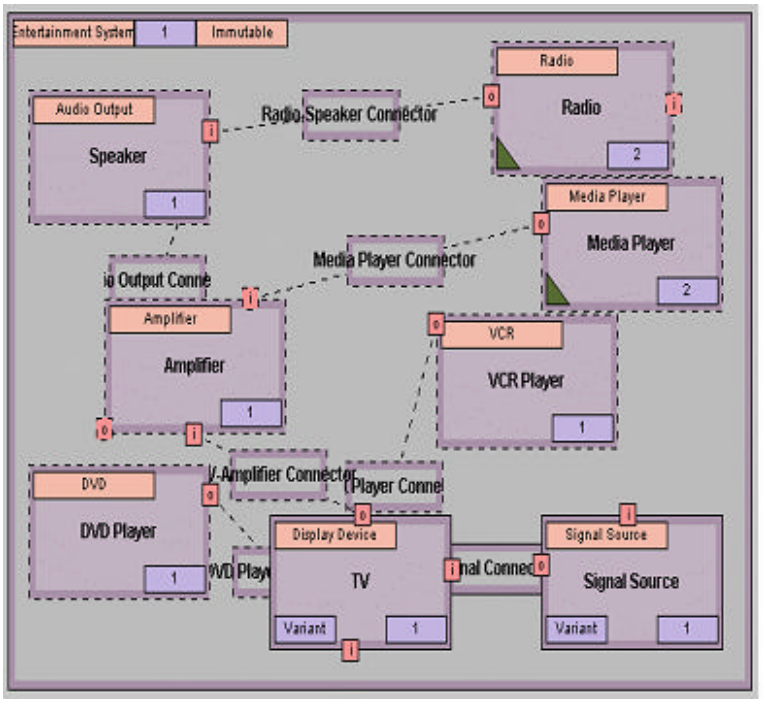

Figura 31: Exemplo de Agrupamento de Componentes no Ménage

Assim como no processo de criação da especificação de componentes, foram sugeridos propostas para apoio ao projeto OO, no caso de agrupamento de componentes poderiam também se utilizar das técnicas para apoio ao DBC que contribuíssem de alguma forma no resultado da composição de componentes das alternativas 1 e 2.

- Para os casos onde o agrupamento de componentes requer ajustes para viabilizar a comunicação entre eles, técnicas de adaptação existentes na literatura poderiam ser consideradas (BOSCH, 1999) (GANESAN e SENGUPTA, 2001) (HEINEMAN e OHLENBUSCH, 2003), e aplicadas sobre conectores. Neste sentido, pretende-se analisar algumas das técnicas que têm este propósito e de que forma poderiam ser contempladas nesta pesquisa;

- Será analisado o uso de padrões arquiteturais para componentes (SCHMIDT, 2000), com o intuito de verificar se algum tipo de suporte poderia ser dado por estes padrões ao agrupamento de componentes.
- Avaliar a especificação da UML 2.0 para identificar que suporte existe ao agrupamento de componentes em termos de artefatos, estereótipos, interfaces, etc (UML, 2003) .

Além dessas técnicas, pretende-se dar suporte ao agrupamento não só de componentes de negócio e de processo, mas também utilitários e de infra-estrutura (BROWN, 2000). No entanto, não se tem uma idéia muito clara da forma como serão disponibilizados estes componentes na arquitetura de um domínio.

Assim como no processo de criação de componentes, pretende-se, ao final das avaliações acima indicadas e após a escolha de uma alternativa de agrupamento, propor um conjunto de heurísticas que orientem o projetista no agrupamento de componente. Estas **heurísticas para agrupamento de componentes** visam apoiar o projetista na atividade de agrupamento em si, quando é válido ou não agrupar componente e, a partir disso, orientá-lo na utilização das técnicas de DBC e, com isso, assegurar uma microarquitetura que atenda os requisitos do domínio.

6.2.3 Montagem da Arquitetura

6.2.3.1 Contexto da Atividade

Uma vez que o agrupamento de componentes tenha ocorrido, a atividade de montagem da arquitetura tende a aproveitar as microarquiteturas geradas e preocupar-se com a arquitetura do domínio como um todo. Tal atividade pode parecer facilitada pelo agrupamento, no entanto, não é trivial. Todos os componentes de negócio, processo, de suporte e de infra-estrutura devem estar representados através de algum estilo arquitetural.

6.2.3.2 Soluções Existentes para utilização nesta Pesquisa

É comum encontrar na literatura, propostas de arquiteturas de componentes baseadas no estilo arquitetural de camadas (SHAW e GARLAN, 1996) (BROWN,

2000) (COPE e MATTHEWS, 2000). Algumas delas com o uso combinado dos estilos arquiteturais para a geração de componente de negócio e de processo (TEIXEIRA, 2003).

No estudo desenvolvido em BLOIS (2003b), foram brevemente analisados outros estilos arquiteturas não específicos para o contexto de componentes. Observou-se que o emprego de algum estilo arquitetural numa arquitetura de componentes não é explícito, nem nos modelos que a representam, nem nas especificações existentes através de alguma ADL. Por exemplo, uma arquitetura de componentes gerada a partir da ferramenta Ménage gera, além do modelo, uma descrição arquitetural através da linguagem xADL. Na xADL de uma arquitetura, pode ser encontrada uma *tag* informando que aquela arquitetura está construída com base naquele estilo arquitetural. No entanto, o restante da especificação da arquitetura não revela qualquer tipo de particularidade que indique restrições ou orientações provenientes do estilo. O que se pode encontrar são informações a respeito dos componentes, interfaces providas e requeridas, conectores, composição de componentes, versões existentes, etc, ou seja, especificações que independem da aplicação de um ou outro estilo arquitetural.

Neste sentido, existe uma tendência no uso do estilo arquitetural em camadas nesta arquitetura para representar os diferentes tipos de suporte oferecidos pelos componentes que compreendem o domínio. Outro ponto positivo no uso do estilo em camadas é o fato de que a representação de *features* também em diferentes camadas, adotada no processo ***CBD-Arch-DE,*** pode facilitar e tornar mais claro o mapeamento para as diferentes camadas de componentes do domínio. Por exemplo, uma *feature* da camada de tecnologia do domínio e/ou ambiente operacional, como proposto por KANG *et al*. (1993), pode vir a ser mapeada para um ou mais componente de suporte definido em (BROWN, 2000).

Independente do tipo de estilo arquitetural futuramente escolhido para a representação da arquitetura geral do domínio, pretende-se que a especificação desta arquitetura seja feita também através de uma ADL.

Baseado nas três ADLs para componentes analisadas na Seção 3.3.1, existe uma forte tendência no uso da xADL. Esta tendência se deve ao fato de que a mesma utiliza o padrão XML para representação arquitetural, propondo um conjunto de *tags* para

representar basicamente componentes, interfaces, conectores e composições de componentes.

Embora o XML gerado pela xADL seja bastante completo, neste trabalho seria interessante estender esta ADL para que se possa também representar as especificações internas dos componentes sugeridas nesta proposta de tese. Inicialmente, espera-se que estas extensões contemplem um conjunto de *tags* adicionais para representar classes, seus atributos e operações. No entanto, uma análise mais criteriosa precisa ser desenvolvida. O objetivo de se apresentar uma especificação arquitetural completa através de uma ADL é viabilizar a tarefa posterior de implementação do domínio. Nesta, o desenvolvedor do domínio poderá fazer uso desta especificação, feita numa linguagem que hoje tem forte tendência a ser padronizada, e implementar numa tecnologia com suporte a componentes.

Ainda anterior à etapa de implementação, pretende-se avaliar a possibilidade de simulação dos casos de uso do domínio com os componentes gerados. O objetivo da simulação é detectar se a solução proposta atende ou não as expectativas sobre o produto a partir de uma situação fictícia. No contexto desta tese, a simulação poderia avaliar se os componentes, da forma como estão arquiteturalmente organizados, estão correspondendo as necessidades do domínio. Embora seja uma atividade importante, este não é o foco da pesquisa, mas sim uma característica para melhor avaliar os resultados produzidos pelo apoio oferecido ao projeto arquitetural de componentes, até então conduzido.

6.3 Considerações Finais

Neste Capítulo foram apresentadas propostas de apoio ao projeto arquitetural baseado em componentes, das quais pode-se avaliar o quão restrito é o apoio a construção de arquiteturas de componentes, em especial no contexto de Engenharia de Domínio. Foram identificados vários problemas com relação ao suporte oferecido e, a partir disso, uma proposta inicial de projeto arquitetural baseada em componentes é apresentada no contexto desta monografia (Seção 6.2). A proposta de projeto arquitetural a ser obtida pela análise das alternativas aqui relatadas tem apoio de um processo (***CDB-Arch-DE***), o qual foi descrito na Seção 5.2. O objetivo desta proposta de projeto arquitetural é melhorar o suporte oferecido pelos métodos de DBC na especificação de componentes e arquiteturas no contexto de ED, em especial quanto às orientações que devem permear

esta atividade de projeto e a adoção de tecnologias para melhor especificar o resultado obtido pela arquitetura desenvolvida.

No próximo Capítulo resumimos as motivações iniciais desta pesquisa, seus objetivos, e apresentada a forma como está sendo conduzida até o momento, bem como uma previsão de atividades futuras e contribuições esperadas.

7 Proposta de Tese: Evolução da Pesquisa

Este Capítulo resume os fatores que motivaram esta proposta de pesquisa, seus objetivos, os trabalhos já desenvolvidos e em desenvolvimento, um cronograma de atividades e os resultados esperados.

7.1 Motivação e Objetivos da Pesquisa

Conforme apresentado na Seção 1.1, esta proposta de trabalho foi motivada por um conjunto de pesquisas desenvolvido junto ao PPGCC-FACIN-PUCRS desde o ano de 2000. As atividades desenvolvidas na PUCRS, tornaram possível a construção do Plano de Pesquisa para Doutorado na COPPE-Sistemas, o qual foca o apoio ao projeto arquitetural baseado em componentes, no contexto de aplicações de aprendizagem cooperativa.

Foi desenvolvida uma avaliação da Infra-Estrutura de Reutilização Odyssey com o objetivo de observar o apoio oferecido pela abordagem de ED (ED) existente no Odyssey para a proposta de projeto arquitetural em CSCL. Nesta avaliação foi possível observar que o apoio a fase de análise de domínio era o ponto forte da infra-estrutura. Por outro lado, pouco suporte era oferecido as atividades de projeto baseado em componentes, principalmente no que diz respeito a construção de uma arquitetura de componentes.

Em função da proposta inicial de pesquisa e da avaliação efetuada na Infra-estrutura Odyssey é que foi então proposta a questão de pesquisa a ser respondida ao longo deste trabalho, conforme apresentado na Seção 1.2: ***Como pode ser melhorado o projeto arquitetural baseado em componentes, no contexto de Engenharia de Domínio?***

Para responder a esta questão de pesquisa foram propostos alguns objetivos, descritos também na Seção 1.2, sendo eles:

1. Definir um processo de Engenharia de Domínio (ED), com detalhamento das atividades que compreendem a construção do projeto arquitetural de componentes do domínio, o qual encontra-se parcialmente apresentado nesta monografia;

2. Propor um conjunto de heurísticas para apoiar a especificação de componentes na atividade de projeto de Engenharia de Domínio;
3. Propor um conjunto de heurísticas para a construção de microarquiteturas de componentes na atividade de projeto de Engenharia de Domínio;
4. Utilizar abordagens para a especificação de arquiteturas de componentes tais, como Estilos Arquiteturais e Linguagens de Descrição Arquitetural;
5. Adotar esta proposta de projeto arquitetural numa infra-estrutura de reutilização já existente (e.g. Odyssey-SDE);
6. Avaliar a proposta de pesquisa através de experimentos desenvolvidos na infra-estrutura.

A forma como se pretende atingir os quatro primeiros objetivos já foi apresentada nas Seções 5.2 e 6.2. O primeiro objetivo desta pesquisa já se encontra possui uma versão inicial implementada na Infra-estrutura Odyssey, sendo esta experiência relatada na Seção 7.2.1. A Seção 7.2 visa descrever em linhas gerais a forma como esta proposta de tese está sendo inserida no contexto da infra-estrutura Odyssey. A Seção 7.3 visa apresentar algumas alternativas de experimentação que neste momento são percebidas para avaliar esta proposta de tese. Na Seção 7.4, é apresentada a ordem cronológica das atividades que fazem parte desta proposta de pesquisa. Por último, na Seção 7.5, são enumerados alguns resultados esperados pela pesquisa em desenvolvimento.

7.2 Infra-Estrutura de Reutilização Odyssey: suporte existente, sugestões de adaptação e implementações necessárias

Conforme registrado anteriormente, esta proposta de projeto arquitetural de componentes para ED, a qual encontra-se em fase de identificação das alternativas de apoio, está sendo submetida no contexto do grupo de reutilização de software da COPPE-UFRJ, onde existe uma infra-estrutura denominada Odyssey que provê uma série de ferramentas para apoio a ED e EA.

Do ponto de vista de suporte ao processo, a infra-estrutura Odyssey possui a ferramenta Charon (MURTA,2002) que se trata de uma máquina para instanciação e

acompanhamento de processos. Ajustes a respeito do uso da máquina de processo no Odyssey são relatados na Seção 7.2.1.

Quanto ao processo ***CBD-Arch-DE***, as atividades previstas na fase de análise são atualmente apoiadas pela infra-estrutura Odyssey. Os artefatos contexto, tipo de negócio e caso de uso já existem, bem como os seus respectivos diagramas. No entanto, foram feitas algumas extensões para uso de *features*, como a apresentada na Seção 7.2.1.2, com o intuito de melhor apoiar a construção da arquitetura de componentes.

Para a atividade de projeto arquitetural baseado em componentes, a infra-estrutura Odyssey disponibiliza os seguintes recursos:

- Estilos Arquiteturais para DBC para apoio a criação de componentes de negócio e de processo (TEIXEIRA, 2003);
- Artefatos Classe, Componente e Interface;
- Diagramas de classe, seqüência, componente;
- Suporte a instanciação de padrões (DANTAS *et al.*, 2002) (XAVIER, 2001).

Com o uso efetivo, ao longo de 2003, da infra-estrutura Odyssey, e tendo em vista o suporte esperado para esta proposta de projeto arquitetural, alguns problemas foram identificados e relatados a equipe do projeto Odyssey. Dentre aqueles que estão sendo resolvidos, estão:

- Mecanismos de rastros entre os diferentes artefatos do domínio, desde a fase de análise até o projeto. Foi feito um estudo detalhado dos rastros necessários, os quais estão em processo de implementação;
- A associação de classes e seus diagramas com seus respectivos componentes. Até então, o diagrama de classes no Odyssey não era vinculado a um componente. Como o objetivo de uma abordagem de DBC é especificar componente através de classes internas, sugeriu-se que o artefato classe estivesse associado ao componente que está especificando;
- Associar o artefato componente a sua *feature* correspondente. Na experiência com a infra-estrutura Odyssey, observou-se que o artefato de mais baixo nível de abstração associado a uma *feature* era a(s) classe(s).

No entanto, tendo como base uma abordagem de DBC, deve ser previsto a rastreabilidade de *features* com seus componentes correspondentes e vice-versa. Os componentes, nesta abordagem, representam os artefatos de mais baixo nível da arquitetura. Vinculando os componentes às *features*, indiretamente estão sendo vinculadas suas classes internas.

Os problemas relatados acima não fazem parte da solução desta proposta de tese. No entanto, a solução dos mesmos visa resolver inconsistências provenientes de muitas pesquisas efetuadas em paralelo na infra-estrutura Odyssey e que, somente ao utilizar o ambiente, foi possível reconhecê-las e sugerir soluções.

7.2.1 Implementações já desenvolvidas desta Proposta de Tese

7.2.1.1 Processo *CBD-Arch-DE*

A proposta da Charon é que, ao se criar um novo domínio, um novo processo possa ser modelado e instanciado pelo Engenheiro de Domínio. No entanto, o fato se ter que criar um processo para todo o novo domínio poderá causar desistência no uso de uma máquina de processo. Por exemplo, para todo o domínio no Odyssey que fosse adotar o processo ***CBD-Arch-DE***, todos as figuras que representam as atividades do processo, colocadas na Seção 5.2 deveriam ser construídas, bem como os artefatos, usuários e *scripts* associados a cada processo primitivo. Isto faz com que o Engenheiro de Domínio venha a desistir de utilizar a Charon, e portanto, desistir de adotar um processo.

Esta parece ser uma característica negativa e, para solucionar este problema, foi proposta uma implementação do processo ***CBD-Arch-DE***, utilizando a Charon. Esta implementação foi desenvolvida de maneira que, quando um Engenheiro de Domínio desejar a criação de um novo domínio, este poderá já fazer a opção por utilizar o processo ***CBD-Arch-DE***. Desta forma, a possibilidade de utilização de um processo, como o proposto nesta tese, torna-se mais ampliado. Com esta abordagem não fica de responsabilidade do Engenheiro de Domínio implementar o processo, uma vez que ele já existente, e que sua implantação poderia ser facilitada.

Assim como foi feito para o ***CBD-Arch-DE***, outros processos consolidados poderiam ser implementados (e.g. Odyssey-DE). Neste caso, o Engenheiro de Domínio

poderia ter, ao criar um domínio, um conjunto de opções de processo que poderia vir a adotar (e.g ***CBD-Arch-DE***, Odyssey-DE, etc.). Esta implementação não invalida a solução existente da Charon, somente acrescenta. Continua existindo a possibilidade do Engenheiro de Domínio criar os seus próprios processos e estes serem monitorados por agentes de simulação.

7.2.1.2 Extensão do Modelo de *Features*

Outro ajuste efetuado na infra-estrutura Odyssey é quanto ao modelo de *features* existente. Este modelo atualmente permite criar apenas as *features* conceituais e funcionais (MILER, 2000), que correspondem a primeira camada da proposta de KANG et al. (1993). Neste sentido, estão sendo disponibilizadas funcionalidades para criação das features das camadas de ambiente operacional, tecnologia de domínio e técnicas de implementação, apresentadas na Seção 4.2.1, as quais serão mapeadas para a arquitetura de componentes através das camadas de suporte e infra-estrutura. Por questões de espaço visual de um diagrama de *features* completo (Figura 32), cada camada poderá também estar representada numa visão diferente (e.g. visão de ambiente operacional com as features da sua camada), além do que cada *feature* conceitual/funcional poderá expressar suas relações com as novas *features* através da aba *technologies* adicionada ao padrão de descrição, conforme apresenta a Figura 33.

O objetivo de adicionar estas novas *features* na infra-estrutura Odyssey é propor, desde o início do processo, uma forma de representação de características tecnológicas que fazem parte do domínio modelado. Com a extensão do modelo de *features* do Odyssey, será possível obter também a especificação de requisitos referentes a questões tecnológicas do domínio e, que por sua vez, serão utilizadas para geração dos componentes de suporte e de infra-estrutura previsto numa arquitetura de componentes de um domínio (BROWN, 2000).

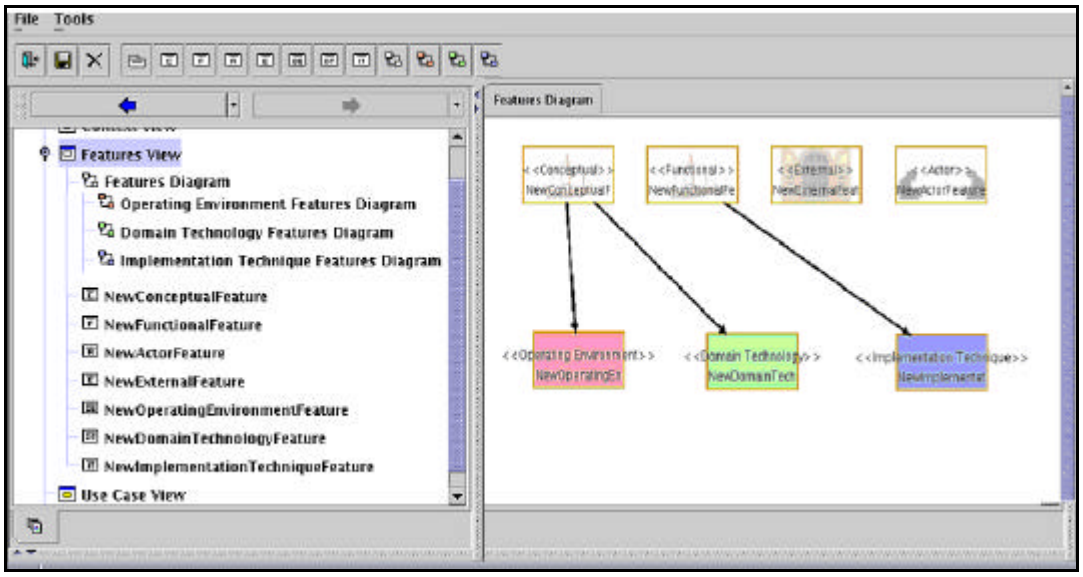

Figura 32: Modelo de *Features* do Odyssey Estendido

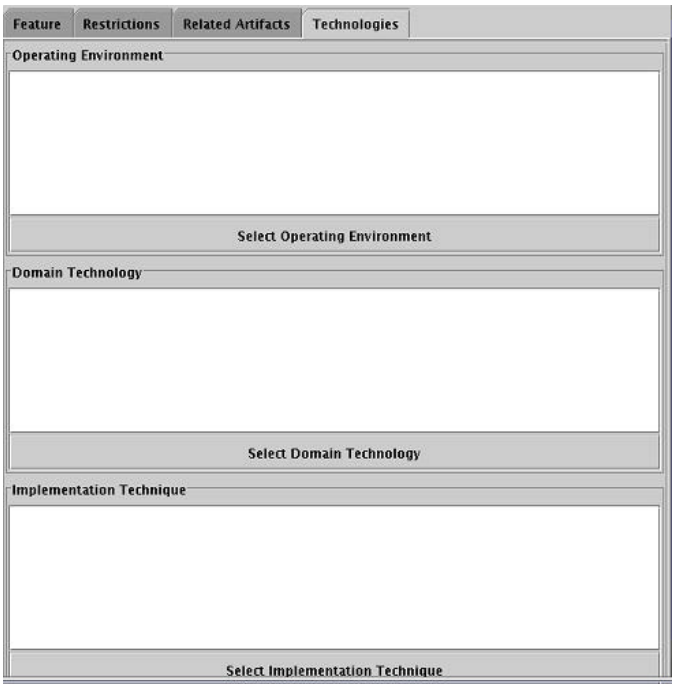

Figura 33: Relação entre *features* expressa através da aba *Technologies*

7.3 Experimentação na Proposta de Tese

Para avaliar se os objetivos desta pesquisa foram atendidos, pretende-se efetuar atividades de experimentação. Estas atividades de experimentação deverão avaliar:

- O apoio oferecido pelo processo ***CDB-ARCH-DE*** na atividade de um projetista do domínio durante a construção de arquitetura de componentes;

- O grau de apoio oferecido ao projetista de um domínio através do conjunto de heurísticas propostas para a especificação de componentes;
- O grau de apoio oferecido ao projetista de um domínio através do conjunto de heurísticas para agrupamento de componentes;

Em função da complexidade envolvida nas questões acima citadas, provavelmente seja necessária a avaliação através de experimentos separados. Para cada experimento será realizado um plano, seguindo o padrão sugerido em WOHLIN *et al.* (2000) e TRAVASSOS, GUROV e AMARAL (2002).

Como mencionado nas Seções 1.1 e 7.1, pretende-se desenvolver experimentos com pessoas envolvidas em atividades de projeto de domínios de aprendizagem cooperativa. Cada pessoa poderá participar de todas as experimentações desenvolvidas para as três perspectivas de experimentação identificadas acima.

Para comparar se a atividade de projeto arquitetural de componentes é melhor suportada através do conjunto de funcionalidades proposta nesta tese, no contexto da infra-estrutura de reutilização Odyssey, é necessário que o conjunto de pessoas que participarão dos experimentos seja dividido em dois grupos. Um grupo que tenha o desafio de fazer o projeto de um domínio de CSCL, utilizando a infra-estrutura Odyssey com o suporte desta proposta de tese já oferecido. O outro grupo, utilizando o processo ***CDB-Arch-DE*** e a proposta de projeto arquitetural concluída, sem suporte tecnológico.

Para estas atividades, além do plano de pesquisa, deverão ser confeccionados documentos para apoio a especificação de requisitos, além de questionários para as avaliações sugeridas.

Dos três tipos de experimentação discutidos em WOHLIN *et al.* (2000) e TRAVASSOS, GUROV e AMARAL (2002), acredita-se que o estudo de caso seja o mais adequado para esta proposta de avaliação. Ao longo desta pesquisa, serão melhor definidas as informações a serem analisadas durante a experimentação, a forma como os experimentos serão conduzidos e, sobretudo, o tipo de experimento a ser adotado.

7.4 Cronograma

Tendo em vista as necessidades de pesquisa enumeradas nas seções anteriores, e o trabalho já desenvolvido desde o ingresso no Programa de Engenharia de Sistemas e Computação, a Tabela 8 organiza cronologicamente as atividades já desenvolvidas e esperadas para o cumprimento desta proposta de tese, detalhadas após a mesma.

Tabela 8: Cronograma de Atividades

Atividade/Período	2003					2004						2005
	3/4	5/6	7/8	9/10	11/12	1/2	3/4	5/6	7/8	9/10	11/12	1/2
1. Cumprimento dos créditos	xx	xx	xx	xxx	xxxx							
2. Avaliação da infra-estrutura Odyssey	x	xx	xx	xxx								
3. Estudo de Estilos e Linguagens de descrição arquitetural para DBC			xx	xxx								
4. Definição Processo ***CBD-Arch-DE***				xx	xx	xx						
5. Redação e Defesa do Exame de Qualificação					xxxx							
6. Publicações						xx					xxxx	
7. Estudo de técnicas para especificação de componentes					xx	xx						
8. Estudo de técnicas para apoio ao agrupamento de componentes							xx	xx				
9. Construção das diretrizes para apoio ao projeto arquitetural						xx	x	xx	xx			
10. Implementação na infra-estrutura Odyssey								xx	xx	xxx	xxx	
11. Experimento para avaliação da proposta											xxx	
12. Redação e Defesa da Tese										xx	xxxx	xxx

1. Cumprimento dos créditos

Foram cumpridos os quatro créditos obrigatórios exigidos para o nível de Doutorado, além de uma disciplina opcional, cursada em caráter de ouvinte. A aluna obteve aprovação A em todas as disciplinas em que o grau já está publicado (3 delas).

2. Avaliação da infra-estrutura Odyssey

Conforme citado anteriormente, a avaliação da infra-estrutura Odyssey ocorreu em decorrência de um levantamento de requisitos para o domínio de CSCL, no contexto da disciplina de Tópicos Especiais em Engenharia de Software I, resultando, dentre outras coisas, uma das motivações desta proposta de tese.

3. Estudo de Estilos e Linguagens de descrição arquitetural para DBC

Esta atividade foi desenvolvida em função na necessidade de obter-se informações a respeito das abordagens de suporte a organização e especificação de arquiteturas de software, em especiais aquelas voltadas à tecnologia de componentes.

4. Definição do Processo *CBD-Arch-DE*

A definição deste processo ocorreu em função dos estudos desenvolvidos sobre Engenharia de Domínio e, em especial, as propostas de processos apresentadas neste documento nas Seções 4 e 5.

5. **Redação e Defesa do Exame de Qualificação**

Esta atividade envolve a redação deste documento e a defesa da proposta perante a banca examinadora.

6. **Publicações**

Até o momento não foram efetuadas publicações diretamente relacionadas a esta pesquisa. No entanto, foram publicados artigos que fizeram parte da motivação desta proposta, sendo elas: (KBECKER e BLOIS, 2001) e (KBECKER e BLOIS, 2002). A primeira é uma publicação veiculada pela IEEE e a segunda por Lecture Notes. Publicações a respeito desta proposta estão sendo previstas a partir da defesa do Exame de qualificação.

7. **Aprofundar o estudo de técnicas para a especificação de componentes**

Este estudo visa obter um levantamento das orientações e do suporte oferecido na literatura para a especificação interna de componentes. Os métodos de DBC (e.g. UML Components, Kobra, Catalysis) serão considerados o ponto de partida deste estudo, bem como os estudos sobre métricas de acoplamento para o projeto OO.

8. **Aprofundar o Estudo de técnicas para apoio ao agrupamento de componentes**

Esta atividade visa identificar as possibilidades para agrupamento de componentes, respeitando o grau de colaboração entre os mesmos. Métodos de DBC

serão analisados sob esta ótica. As métricas para projeto OO, que tem maior foco no projeto de classes serão avaliadas para identificar a possibilidade de estender seus princípios para o agrupamento de componentes.

9. **Elaboração das diretrizes para apoio ao projeto arquitetural**

Esta atividade está baseada nos resultados obtidos das duas anteriores. Com base no conjunto de heurísticas para apoio a especificação e o agrupamento de componentes, pretende-se tornar a atividade do projetista na ED mais orientada. Esta orientação não torna a atividade do projetista mais fácil, mas o ajuda a evitar problemas de projeto comumente ocorridos por falta desse tipo de apoio.

10. **Implementação na Infra-Estrutura Odyssey**

Algumas atividades previstas nesta tese já foram implementadas no Odyssey (Processo ***CBD-Arch-DE*** e Extensão do Modelo de *Features*). As heurísticas e o conjunto de artefatos que venham a ser propostos em função das heurísticas, deverão ser implementados na infra-estrutura de reutilização Odyssey. Com esta implementação, será possível, posteriormente, avaliar se a proposta de tese atinge seus objetivos.

11. **Experimento para Avaliação da Proposta**

Pretende-se desenvolver experimentos para avaliação do processo e do suporte arquitetural. Para tal serão desenvolvidos planos de experimentação baseado nas abordagens apresentadas em WOHLIN *et al.* (2000) e TRAVASSOS, GUROV e AMARAL (2002).

12. Redação e Defesa da Tese

O texto da tese será composto dos resultados obtidos das atividades posteriores e, ao final do período, a defesa perante a banca examinadora.

7.5 Resultados Esperados

Com o desenvolvimento de todas as atividades previstas nesta pesquisa, e em especial com o resultado obtido nas atividades de experimentação, espera-se obter alguns resultados promissores a respeito:

1) Do suporte oferecido pelo processo ***CBD-Arch-DE*** em especial para a construção de arquiteturas de componentes;

2) Da dinâmica oferecida pelo conjunto de heurísticas, tanto na especificação quanto no agrupamento de componentes, permitindo que a arquitetura em si represente da melhor forma possível o domínio ao qual ela se retrata;
3) Da utilização de uma ADL para descrever de forma clara a arquitetura de componentes de um domínio, viabilizando a atividade de implementação através de uma tecnologia de componentes que reconheça esta descrição;
4) Da contribuição que a proposta de tese vem a agregar no contexto da infra-estrutura de reutilização Odyssey;

Neste sentido, espera-se que, de forma combinada, os objetivos da pesquisa possam dar melhor apoio ao projeto arquitetural de componentes, comparado às propostas analisadas durante esta monografia, as quais apresentam limitações, conforme discutidos anteriormente e que, sobretudo, seja uma real contribuição para a Engenharia de Software.

8 Referências:

ANDRADE, R., 1998, **Aplicação Simultânea de Princípios, Diretrizes e Métricas para Redução de Complexidade de Projetos OO**. Tese de Mestrado, COPPE-UFRJ, 1998.

ATKINSON, C.; APECH, B.; REINHOLD, J.; SANDER, T., 2001, **Developing and applying component-based model-driven architectures in KobrA**. Enterprise Distributed Object Computing Conference, 2001. EDOC '01. Proceedings. Fifth IEEE International, 4-7 Sept. 2001. 212 –223pp.

ATKINSON, C.; BAYER, J.; BUNSE, C.; KAMSTIES, E.; LAITENBERGER, O.; LAQUA, R.; MUTHIG, D.; PAECH, B.; WÜST, J.; ZETTEL, J., 2002, **Component-based Product Line Engineering with UML.** Addison-Wesley, 505p.

BECKER, K.; BLOIS. A., 2002. **A Component-based Architecture to Support Collaborative Application Design**. In: 8th International Workshop on Groupware, Lecture Notes in Computer Science. La Serena, Chile.

BECKER, K.; BLOIS, A., 2001. **Considerations on the application of object-oriented reuse technology to the Computer-Supported Cooperative Learning Domain**. In: SEVENTH INTERNATIONAL WORKSHOP ON GROUPWARE - CRIWG 2001, 2001, Darmstadt. IEEE Computer Society Press.

BLOIS, A., 2003a, **Framework de Requisitos para CSCL**. Relatório da disciplina de Tópicos Especiais em Eangenharia de Software 1. COPPE-Sistemas, UFRJ, Orientadora: Cláudia M.L. Werner. Maio. 2003.

BLOIS, A., 2003b, **Estilos Arquiteturais e Linguagens de Descrição Arquiteturais para DBC**. Relatório da disciplina de Reutilização de Software. COPPE-Sistemas, UFRJ, Orientadora: Cláudia M.L. Werner. Set. 2003.

BLOIS, A., 2000. **FrameWorks orientados a objetos: estudos de caso**. Monografia apresentada na disciplina de Trabalho Individual II: PPGCC-PUCRS. Orientadora: Karin Becker. Set. 2000.

BLOIS, A., 2001, **Padrões e frameworks: tecnologias de reutilização para aplicações orientadas a objetos**. Monografia referente à disciplina de Trabalho Individual III do PPGCC-PUCRS. Orientadora: Karin Becker. Maio. 2000.

BOOCH; G.; RUMBAUGH, J.; JACOBSON, I., 2000, **UML: Guia do Usuário**. Rio de Janeiro: Campus, 2000. 472p.

BOSCH, J., 1999, **Superimposition: a component adaptation technique**. Information and Software Tecnology. Elsevier. N 41, 1999, 257-273pp.

BOSCH, J., 2000, **Design and use of software architectures: adopting and evolving a produc-line approach**. Addision-Wesley, 529p.

BROWN, A; WALLNAU, K., 1998, The Current State of CBSE, **IEEE Software**, setembro/outubro, 1998, 37-46pp.

BROWN, A., 2000, **Large-Scale Component-Base Development**. Prentice Hall Books. 2000. 286p.

BRAGA, R., 2000, **Busca e Recuperação de Componentes em Ambientes de Reutilização de Software**. Tese de Doutorado. COPPE Sistemas, 2000, 265p.

CHEESMAN, J.; DANIELS, J., 2001, **UML Components: A simple process for specifying component-based software**. Addison-Wesley,208p.

CHEN, F. et al., 2002, **An architecture-based approach for component-oriented development**. 26th Annual International Computer Software and Applications Conference **,** August 26 - 29, Oxford, England. 2002.

CHO,E.; KIM, C.; KIM, S.;RHEW, S., 1998, **Static and Dynamic Metrics for Effective Object Clustering**. Software Engineering Conference. 1998, Asia, 1998.

COPE, M.; MATTHEWS, H., 2000, **A Reference Architecture for Component Based Development**. Ratio Group. http://www.ratio.co.uk.. Extraído em 11/08/2000.

DANTAS,A.R.; VERONESE, G.; CORREA, A.; XAVIER, J; WERNER, C., 2002, **Suporte a Padrões no Projeto de Software**. Caderno de Ferramentas do XVI Simpósio Brasileiro de Engenharia de Software (SBES'2002), Gramado, Rio Grande do Sul, Outubro, 2002.

DASHOFY, E.; HOEK, A.; TAYLOR, R., 2002, **An Infrastructure for the Rapid Development of XML-based Architecture Description Languages**. Software

Engineering, 2002. ICSE 2002. Proceedings of the 24rd International Conference on , 19-25 May 2002.

DI NITTO, E.;ROSENBLUM, D., 1999, **Exploiting ADLs to Specify Architectural Styles induced by Middleware Infrastructures**. Software Engineering, 1999. Proceedings of the 1999 International Conference on , 16-22 May 1999, 13 –22pp.

D'SOUZA,D; WILLS, A., 1999, **Objects, components and frameworks with UML : the catalysis approach**. Addison-Wesley, 1999. 785p.

EMMERICH, W.; ELLMER, E.; FIEGLEIN, H, 2001, **TIGRA – An architectural style for enterprise application integration**. Software Engineering, 2001. ICSE 2001. Proceedings of the 23rd International Conference on , 12-19 May 2001.

FAYAD, M., 1997, **Object-oriented application frameworks**. CACM, v.0, n. 10,. Oct. 1997, 32-38pp.

FIELDING, R., 2000, **Architectural Styles and the Design of Network-based software architecture.** Doctor Dissertation, University of California,2000.

GAMMA, Erich, *et al*., 1994, **Design Patterns: Elements of Reusable Object-Oriented Software**. Reading, MA: Addison-Wesley, 1994.

GANESAN,R.; SENGUPTA, S., 2001, **O2BC: a techinique for the design of compoent-based application**. Technology of Object-Oriented Languages and Systems, 2001. TOOLS 39. 39th International Conference and Exhibition on , 29 July-3 Aug. 2001 46-55pp.

GRISS, M.; FAVARO, J.; D'ALESSANDRO, M., 1998, **Integrating Feature Modeling with the RSEB**. Software Reuse, 1998. Proceedings. Fifth International Conference on, 2-5 June 1998. 76 –85pp.

HEINEMAN, G.; OHLENBUSCH, H., 2001, **An Evaluation of Component Adaptation Technique**. URL: www.cs.wpi.edu/~heineman/classes/cs562/pdf/ho99.pdf. Extraído em: 10/08/2003.

HOEK, A. 2003. Arch Studio 3. URL: http://www.isr.uci.edu/projects/archstudio. Extraído em 01/11/2003.

HERZUM, P; SIMS, O., 2000, **Business Component Factory: A Comprehensive Overview of Component-Based Development for the Enterprise**, OMG Press, 2000.

KANG, K.C. *et al.*, 1993, **Feature-Oriented Domain Analysis (FODA) Feasibility Study** (CMU/SEI-90-TR-21). Pittsburg, PA, Software Engineering Institute, Cargegie Mellon University, Nov 1993.

KANG, K.; LEE, J.; DONOHOE, P., 2002, **Feature-Oriented Product Line Engineering**. IEEE Software, July, 58-65p, August 2002.

LEE, S.; YANG, Y.; CHO,E.; KIM, S.; RHEW,S., 1999, **COMO: A UML-Based Component Development Methodology**. Proceedings of the Sixth Asia-Pacific Software Engineering Conference, Takamatsu, Japan, 7-10 Dec. 1999, IEEE CS Press, 54-61pp.

LEE, K.; KANG, K.; LEE, J., 2002, **Concepts and Guidelines of Feature Modeling for Product Line Software Engineering**. 7th International Conference on Software Reuse (ICSR), Austin, Texas, USA, Apr. 15-19, 2002, 62-77pp.

LEE, K.; KANG, K.; KIM, S.; LEE, J., 1999, **Feature-Oriented Engineering of PBX Software**. Sixth Asia Pacific Software Engineering Conference, Takamatsu, Japan, December, 1999.

LÜER, C.; ROSENBLUM, D., 2001, **WREN – An Environment for Component-Based Development**. ESEC/FSE. Vienna, Austria, ACM 2001, 207-217pp..

MANGAN, M.A.S; MURTA, L.G.P.; SOUZA, J.; WERNER, C.M.L., 2001, **Modelos de Domínio e Ontologias: uma comparação através de um estudo de caso prático em hidrologia**. IV International Symposium on Knowledge Management / Document Management (ISKM/DM'2001), Curitiba, agosto 2001.

MEDVIDOVIC, N; TAYLOR, R., 2000, **A classification and comparation framework for software architecture description languages.** IEEE Transations on software engineering. Vol.26, no.1, 2000.

MENCL, V., 1998, **Component Definition Language.** Master Thesis, Universitas Carolina Pragensis.1998.

MORISAWA, Y.; TORII, K., 2001, **An architectural style of product line for distributed processing systems, and practical selection method.** Proceedings of the 8th European software engineering conference held jointly with 9th ACM SIGSOFT international symposium on Foundations of software engineering, Volume 26 Issue 5. ESEC/FSE 2001.

MURTA, L., 2002, **Charon: uma máquina de processo extensível baseada em agentes inteligentes**. Tese de M.Sc., COPPE/UFRJ, Rio de Janeiro, Brasil, março, 2002.

MILER, N., 2000, **A Engenharia de Aplicações no Contexto da Reutilização baseada em Modelos de Domínio.** Tese de M.Sc., COPPE/UFRJ, Rio de Janeiro, Brasil, julho 2000.

POULIN, J., 1997. **Software Architectures, Product Lines, and DSSAs: Choosing the Appropriate Level of Abstraction**. Eighth Annual Workshop on Institutionalizing Software Reuse. Ohio State University, March 23-26, Proceedings On-line.

PRESSMAN, R., 2000, **Software Engineering: A Practitioner's Approach**. McGraw-Hill.Fifth Edition, 2000.

SAMETINGER, J., 1997, **Software Engineering with Reusable Components**. Springer, 1997. 271p.

SCHMIDT, D., 2000, **Pattern-Oriented Software Architecture, Volume 2, Patterns for Concurrent and Networked Objects**, John Wiley & Sons, 2000.

SEI. Software Engineering Institute. Carnegie Mellon University. URL: http://www.sei.cmu.edu/domain-engineering/domain_eng.html. Extraído em 16/11/2003.

SHAW, M., GARLAN, D., 1996, **Software Architecture: perspectives on an emerging discipline.** 1 ed. Nova Jersey. Prentice-Hall. 1996.

SHAW, M.; CLEMENTS, P., 1996, **Toward Boxology: preliminary classification of architectural styles**. International Computer Software and Applications Conference, August 1997, 6-13pp.

SIMOS, M.; ANTHONY, J., 1998, **Weaving the Model Web: A Multi-Modeling Approach to Concepts and Features in Domain Engineering**. Proceedings of 5th International Conference of Software Reuse (ICSR-5), ACM/IEEE, Vitória, Canadá, Junho. 1998, 94-102pp.

SOFA Project. Disponível em: http://nenya.ms.mff.cuni.cz/projects/sofa/tools/doc/compmodel.html#N2662. Extraído em 20/09/2003

TEIXEIRA, H., 2003, **Geração de Componentes de Negócio a partir de modelos de análise**. Tese de M.Sc., COPPE/UFRJ, Rio de Janeiro, Brasil, Março, 2003. 131p.

TRAVASSOS, G.; GUROV, D.; AMARAL, E., 2002. **Introdução a Engenharia de Software Experimental**. Relatório Técnico – RT-ES-590-02. COPPE-Sistemas, 52p.

UML, 2003, **Especificação da UML 2.0 Superestrutura**. Disponível em http://www.omg.org/uml. Extraído em 26/10/2003.

VICI, A.D.; ARGENTIERI, N., 1998, **FODAcom: An Experience with Domain Analysis in the Italian Telecom Industry**. Fifth International Conference on Software Reuse (ICSR5) Abril, 1998.

XAVIER, J.R., 2001, **Criação e Instanciação de Arquiteturas de Software específicas para domínio no contexto de uma infra-estrutura de reutilização**. Tese de Mestrado. COPPE-Sistemas, junho, 2001.

YACOUB, S.; AMMAR, H.; ROBINSON, T., 1999, **Dynamic Metrics for Object Oriented Designs**. Sixth International Software Metrics Symposium, Boca Raton, Florida, USA, 4-6 Nov, 1999, 50-61pp.

WERNER,C; BRAGA,R., 2000, **Desenvolvimento Baseado em Componentes**. Minicurso desenvolvido no XIV Simpósio Brasileiro de Engenharia de Software, 4 a 6 de outubro de 2000, João Pessoa – PB. 33p.

WERNER,C., 2003, **Reutilização de Software**. Disciplina de Reutilização de Software. Material de Aula. COPPE Sistemas – UFRJ. Notas de Aula. 2003.

WOHLIN,C.; RUNESON, P.; HÖST, M.; REGNELL, B.; WESSLÉN, A.; 2000. **Experimentation in Software Engineering**. Kluwer Academic Publishers, 204p.