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

Exame de Qualificação - Doutorado

Uma Abordagem Para Recuperação De Arquitetura De Software Visando Sua Reutilização Em Domínios Específicos

Aluna: Aline Pires Vieira de Vasconcelos

Orientadora:
Cláudia Maria Lima Werner

COPPE – RJ
Junho de 2004

Índice

Índice de Figuras

Índice de Tabelas:

1. Introdução

1.1 Contexto e Motivações

A capacidade de reutilização vem representando um fator de qualidade cada vez mais requerido em artefatos de software. Para o desenvolvimento de sistemas de software grandes e complexos, compatíveis com as atuais exigências do mercado consumidor, o reaproveitamento de componentes de prateleira (COTS – *Components off the Shelf*) ou o desenvolvimento de componentes de software que possam ser reutilizados em diversos sistemas é essencial. A reutilização, se bem empregada, propicia um aumento de produtividade e qualidade no desenvolvimento de sistemas, o que acarreta, conseqüentemente, em uma redução do prazo e dos custos. Porém, não só componentes de código representam, atualmente, o alvo de esforços da reutilização, como ocorria no passado. O reaproveitamento de artefatos em níveis mais altos de abstração, como modelos de análise e projeto, também vem sendo buscado. Com o reaproveitamento de artefatos em níveis mais altos de abstração, os benefícios da reutilização podem ser atingidos desde o início do ciclo de vida do software e, conseqüentemente, os ganhos tendem a ser bem maiores.

Neste cenário, diversas abordagens vêm sendo propostas para suportar a reutilização de software. Dentre elas, destaca-se a Engenharia de Domínio (ED).

A Engenharia de Domínio surge na década de 80 como uma perspectiva promissora para a reutilização de artefatos em todas as fases do ciclo de vida. Ela visa o desenvolvimento de artefatos de análise, projeto e implementação para uma família de aplicações que apresentam um conjunto de requisitos em comum, compondo um domínio de aplicações. As aplicações desta família são construídas, prioritariamente, com base na reutilização e adaptação dos artefatos do domínio, em um processo denominado Engenharia de Aplicação (EA). Portanto, enquanto a ED visa o desenvolvimento para reutilização, a EA visa o desenvolvimento com reutilização, complementando-se, estas duas abordagens, na definição de um ciclo completo de reutilização de software.

Vários métodos de Engenharia de Domínio vêm sendo propostos na literatura (SIMOS, 1996), (KANG et al., 1998), (CHAN et al., 1998), (GRISS et al., 1998),

(BRAGA, 2000) etc. Todos eles concordam que existem três etapas principais na ED, a saber: Análise do Domínio, que determina os requisitos comuns de uma família de aplicações, com o objetivo de identificar as oportunidades de reutilização; Projeto do Domínio, que utiliza os resultados da análise do domínio para identificar e generalizar soluções para os requisitos comuns; e a etapa de Implementação do Domínio, que transforma as oportunidades de reutilização e soluções do projeto em um modelo implementacional, envolvendo a identificação, construção (ou extração) e a manutenção de componentes reutilizáveis no domínio. Em geral, os métodos de ED oferecem um maior suporte à etapa de Análise do Domínio, deixando as etapas de Projeto e de Implementação carentes de um maior apoio.

Na etapa de Projeto do Domínio, etapa para a qual se pretende oferecer contribuições neste trabalho, as arquiteturas de referência de domínio (DSSA's – *Domain Specific Software Architectures*) desempenham um papel fundamental. Elas estabelecem uma organização estrutural para a família de aplicações, em termos dos seus elementos arquiteturais e conectores, sendo a base para a etapa de Implementação do Domínio e para a instanciação de aplicações no processo de EA. Segundo CHUNG et al. (1994), as arquiteturas de software, especialmente preparadas para a reutilização, também conhecidas como arquiteturas de referência de domínio, são os artefatos que efetivamente realizam, através das inter-relações dos seus componentes, as funções essenciais do domínio e dão sustentação às características de qualidade desejadas.

Porém, por apresentar uma característica de generalização, buscando atender uma família de aplicações, o custo de criação de uma DSSA é bem maior do que o custo de criação de uma arquitetura para uma aplicação específica (WENTZEL, 1994). A fim de minimizar este problema e apoiar a etapa de Projeto do Domínio, diversos métodos e ferramentas têm sido propostos para suportar a criação de arquiteturas de referência para o domínio (XAVIER, 2001), (O'BRIEN e SMITH, 2002) e (METTALA e GRAHAM, 1992).

XAVIER (2001), por exemplo, se baseia nos atributos de qualidade do domínio para a indicação de um padrão arquitetural possivelmente adequado para a construção de uma arquitetura de referência para o domínio. Em seu trabalho, padrões arquiteturais são selecionados para um domínio comparando-se os atributos de qualidade desejados no

domínio com a probabilidade em se obter estes atributos através da utilização de um determinado padrão arquitetural. Cada padrão arquitetural proposto por (BUSCHMANN et al., 1996) privilegia um conjunto de atributos de qualidade, enquanto desfavorece outros. XAVIER (2001) apóia a definição da estrutura do sistema, mas não suporta a especificação dos componentes e conectores que devem compor esta estrutura.

O'BRIEN e STOERMER (2001), por outro lado, se baseiam na análise de sistemas legados em um domínio para apoiar a definição de uma arquitetura de referência. Eles propõem um método de mineração de arquiteturas, o qual se inicia com a avaliação do domínio, a seleção dos sistemas candidatos e a recuperação das arquiteturas dos sistemas legados. A seguir, as arquiteturas recuperadas são comparadas e um relatório das semelhanças e diferenças entre elas é gerado. Os dados deste relatório servem como informações de entrada para a construção de uma arquitetura de referência para o domínio, na qual partes semelhantes entre as arquiteturas recuperadas podem ser reutilizadas. Porém, embora os autores proponham um processo de extração de arquiteturas e um conjunto de critérios para a sua comparação, uma contrapartida nesta proposta é que um grande esforço manual é requerido para a realização das atividades.

Convém ressaltar, que neste método de O'BRIEN e STOERMER (2001) a atividade de recuperação das arquiteturas dos sistemas legados assume um papel fundamental. Conforme afirma KANG (1990), uma das fontes de informação essenciais para a ED são os sistemas legados disponíveis no domínio. Embora eles representem uma rica base de artefatos, os sistemas legados raramente apresentam uma documentação de apoio e quando esta documentação existe, ela está geralmente fora de sincronia com o sistema implementado (KAZMAN e CARRIÈRE, 1997). Dessa forma, a fim de fazer uso da base de informações e artefatos disponíveis dos sistemas legados, apoiando as diversas etapas da ED, processos de engenharia reversa são requeridos.

Dentre os processos de engenharia reversa que vêm sendo mais enfatizados no momento está a recuperação da arquitetura de software de sistemas legados, a qual visa não só apoiar a construção de arquiteturas de referência para um domínio, como também apoiar as manutenções de software. Diversas abordagens vêm sendo propostas na literatura para suportar a recuperação da arquitetura de software de sistemas legados (KAZMAN e CARRIÈRE, 1997), (BOJIC e VELASEVIC, 2000), (RIVA e RODRIGUEZ, 2002),

(GALL e PINZGER, 2002) etc. Elas de fato apóiam a reconstrução da arquitetura e oferecem inúmeras contribuições para a área, mas deixam alguns pontos em aberto, como a recuperação da arquitetura de sistemas cujo código fonte não se encontra disponível e a disponibilização de um apoio mais efetivo à tomada de decisão ao longo do processo. A fim de cobrir estes pontos em aberto, novas propostas de abordagens de engenharia reversa visando a reconstrução da arquitetura de sistemas legados se fazem necessárias.

A engenharia reversa representa uma atividade com potencial de oferecer significativas contribuições à Engenharia de Domínio, melhorando o suporte ao Projeto e Implementação do Domínio, atividades estas que apresentam suporte deficiente nos métodos de ED existentes. A decisão de uma organização em iniciar um processo de Engenharia de Domínio para um determinado domínio, normalmente é feita com base na gama de sistemas legados disponíveis no domínio e na sua experiência naquela área de aplicação. Por isso, estes sistemas em geral se encontram disponíveis e a reutilização de seus artefatos pode proporcionar um ganho de produtividade e qualidade consideráveis no processo de ED.

Pode-se concluir, diante do contexto apresentado, que para a atividade de Projeto do Domínio, a qual é de interesse nesta pesquisa, o desenvolvimento de técnicas e ferramental de suporte ainda são necessários e que a engenharia reversa da arquitetura dos sistemas legados, embora ainda requerendo suporte mais efetivo, pode trazer inúmeras contribuições para o projeto arquitetural do domínio.

1.2 Objetivos

O objetivo deste trabalho é oferecer contribuições para a Engenharia de Domínio, principalmente para a etapa de Projeto do Domínio, através do suporte à extração e análise de artefatos de sistemas legados. A proposta envolve a definição de um processo de recuperação de arquiteturas de software de sistemas legados, o estabelecimento de critérios e técnicas para a comparação das arquiteturas recuperadas e o desenvolvimento de um ferramental para suportar a abordagem proposta. O foco deste trabalho é em sistemas orientados a objetos (OO), pois uma das hipóteses estabelecidas nesta proposta é a de que os sistemas orientados a objetos serão os sistemas legados de um futuro próximo dado o volume de desenvolvimento de software que vem ocorrendo neste paradigma.

O processo proposto para a recuperação da arquitetura se baseia mais fortemente nos modelos dinâmicos extraídos dos sistemas, visto que a análise dinâmica vem sendo menos explorada nos trabalhos de recuperação de arquitetura e que os modelos dinâmicos oferecem uma rica fonte de informações sobre o sistema, facilitando a compreensão da sua organização e comportamento. Este fato é ainda mais marcante para aplicações orientadas a objeto, onde as colaborações entre os objetos representam um elemento de modelagem fundamental para se compreender o funcionamento do sistema. A análise dinâmica proposta neste trabalho se baseia no mapeamento de funcionalidades do sistema legado para os elementos do código fonte, através da coleta dos eventos gerados pelo sistema durante a sua execução.

O processo de recuperação de arquitetura proposto visa cobrir alguns pontos em aberto nas atuais abordagens de recuperação de arquitetura, como: oferecer um maior apoio à tomada de decisão na montagem dos elementos arquiteturais, implementar alguns extratores de informações do sistema que atuem sobre a sua versão executável e não sobre o código fonte, ampliar estilos arquiteturais identificados no sistema com base na análise dinâmica da aplicação e mapear funcionalidades da aplicação para a arquitetura, recuperando um modelo de visões arquiteturais que contemple perspectivas estáticas e dinâmicas.

Depois de recuperadas as arquiteturas, o próximo passo é a sua comparação e a extração de um mapa das semelhanças e diferenças entre sistemas legados de um domínio. A comparação dos sistemas pretende ser efetuada tanto em nível arquitetural, comparando-se os elementos arquiteturais recuperados, os conectores e os padrões arquiteturais empregados pelos sistemas, quanto em nível funcional, comparando-se as funcionalidades identificadas para cada sistema legado ao longo do processo de recuperação de arquitetura.

Um dos problemas apresentados nos trabalhos semelhantes existentes, como o de O'BRIEN e STOERMER (2001), é o grande esforço manual requerido para a realização das atividades propostas. Portanto, a fim de minimizar este problema, faz parte do escopo desta proposta a definição e o desenvolvimento de um ferramental de suporte à abordagem. O ambiente selecionado para suportar o processo proposto é o Odyssey (WERNER et al., 2004), o qual representa uma infra-estrutura de reutilização baseada em modelos de domínio, englobando tanto a Engenharia de Domínio quanto a Engenharia de Aplicação. O

Odyssey oferece uma série de funcionalidades e ferramentas que apóiam a execução das atividades propostas, além de um repositório de artefatos do domínio contemplando ligações entre os artefatos nos diferentes níveis de abstração. Para as atividades cujo suporte ferramental não se encontra disponível no Odyssey, ferramentas isoladas serão desenvolvidas e integradas à infra-estrutura Odyssey.

Concluindo, pode-se dizer que a abordagem proposta pretende trazer como principal benefício o apoio à definição de uma arquitetura de referência para um domínio. Contudo, outras contribuições são esperadas, como: um suporte mais efetivo ao processo de recuperação de arquitetura de software de sistemas legados, cobrindo alguns dos pontos em aberto nas atuais abordagens; a geração de artefatos em outros níveis de abstração para o domínio, como artefatos em nível de análise; e o apoio às manutenções dos sistemas legados, através da redocumentação da sua arquitetura e do mecanismo de mapeamento de funcionalidades para o código fonte.

1.3 Organização da Monografia

O restante desta monografia está organizado em cinco capítulos. Os capítulos 2, 3 e 4 apresentam o estado da arte em relação às áreas chave que embasam esta proposta de tese. O capítulo 2 apresenta a área de arquitetura de software, englobando algumas definições propostas, abordagens para a descrição da arquitetura, dentre outros temas. O capítulo 3 apresenta técnicas para suportar a recuperação da arquitetura de software de sistemas legados, como a engenharia reversa estática e dinâmica e os mecanismos de agrupamentos de elementos (*clustering*). No capítulo 4, abordagens significativas na área de recuperação de arquitetura de software são descritas. Nesta descrição, são ressaltados os métodos de análise de aplicação empregados (estáticos ou dinâmicos), as técnicas aplicadas e o ferramental proposto. As abordagens são comparadas e os seus benefícios e pontos em aberto são destacados.

Os capítulos 5 e 6 apresentam a proposta de tese propriamente dita. O capítulo 5 descreve o processo de recuperação de arquitetura proposto e os critérios e as técnicas que devem ser utilizadas para comparar as arquiteturas recuperadas e apoiar a definição de uma arquitetura de referência para um domínio. O capítulo 6 apresenta o ferramental de suporte à abordagem proposta. Para finalizar a monografia, o capítulo 7 apresenta o planejamento

para a realização das atividades, estabelecendo um cronograma que inclui, além das atividades propostas, a realização de estudos de caso para a avaliação da abordagem.

2 Arquitetura de Software

2.1 Introdução

Os sistemas de software vêm se tornando cada vez mais essenciais à sociedade e às organizações. Dentre os fatores que impulsionaram este fato estão a evolução do hardware nas últimas décadas e a explosão do uso da Internet. Software representa atualmente um fator de competitividade para as empresas, as quais apresentam uma demanda crescente por sistemas e uma exigência de prazo e qualidade cada vez maiores. Com tudo isso, o perfil do desenvolvimento de software vai se modificando, com o foco migrando para sistemas distribuídos, mais complexos e com uma produção em grande escala.

Para suportar este novo perfil do desenvolvimento de software, uma disciplina vem emergindo e ganhando importância: a arquitetura de software. A arquitetura enfatiza a organização global do sistema, definindo a sua topologia e permitindo que desenvolvedores voltem a sua atenção para os requisitos funcionais e não-funcionais que precisam ser atendidos, antes de se aterem ao projeto das estruturas de dados e algoritmos. Em outras palavras, projetar a estrutura global de um sistema emerge como um problema novo: o desenvolvimento de software orientado para arquitetura (MENDES, 2002).

Diversas definições são dadas para arquitetura de software na literatura. Dentre elas, destacam-se:

"Descrição dos elementos a partir dos quais os sistemas são construídos (componentes), interações entre estes elementos (conectores), padrões que guiam sua composição, e restrições sobre estes padrões" (SHAW e GARLAN, 1996).

"A estrutura de componentes de um programa/sistema, seus relacionamentos, princípios e diretrizes que governam seu projeto e evolução ao longo do tempo" (GARLAN e PERRY, 1995).

"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), da natureza e das propriedades relevantes que são externamente visíveis destas partes (módulos com interfaces, unidades de hardware, objetos), e dos relacionamentos e restrições entre elas" (D'SOUZA e WILLS, 1999).

A arquitetura desempenha um papel ainda mais importante nas abordagens de engenharia de software voltadas à reutilização, como a ED, onde ela estabelece a estrutura na qual os componentes reutilizáveis do domínio são conectados e instanciados, e para a qual os novos componentes das aplicações devem apresentar compatibilidade. Neste contexto, torna-se cada vez mais evidente a necessidade do projeto arquitetural nos processos de engenharia de software.

O objetivo deste capítulo é apresentar a disciplina arquitetura de software, descrevendo os conceitos relacionados e alguns temas que vêm sendo pesquisados na área.

2.2 Modelos de Descrição Arquitetural

O desenvolvimento de software no nível arquitetural, conforme pôde ser observado nas definições apresentadas, compreende questões estruturais, dentre as quais destacam-se: a divisão dos sistemas em componentes[1] ou subsistemas, interconexões entre os componentes, seleção de alternativas de projeto e atribuição de funcionalidades a componentes de projeto. Porém, embora os aspectos estruturais recebam sempre maior destaque nas definições de arquitetura, estes não são os únicos a serem considerados para se obter uma completa e compreensível descrição arquitetural de um software. Conforme afirma CLEMENTS (1994), para o completo entendimento da arquitetura de um software, é necessário considerar não apenas a perspectiva estrutural, mas também aspectos como distribuição física, processo de comunicação e sincronização, entre outros. Cada um destes aspectos deve ser tratado em uma diferente visão arquitetural.

PENEDO e RIDDLE (1993) reforçam esta idéia, afirmando que uma arquitetura de software deve ser vista e descrita sob diferentes perspectivas (ou visões) e deve identificar seus componentes, relacionamento estático, interações dinâmicas, propriedades, características e restrições.

A maior parte dos trabalhos em arquitetura de software, até o presente momento, reconhece a existência de diferentes visões arquiteturais. Estes trabalhos seguem uma de duas linhas: ou eles tratam as diferentes visões arquiteturais explicitamente, ou eles focam em uma visão particular no sentido de explorar as suas características específicas,

[1] Componentes neste contexto referem-se a qualquer elemento arquitetural de software, não necessariamente a uma unidade auto-contida, independente e substituível, conforme definem os autores da área de Desenvolvimento Baseado em Componentes (DBC).

distinguindo-as das demais (BASS et al., 1998). Neste contexto, o objetivo desta seção é apresentar alguns modelos de descrição arquitetural existentes na literatura, demonstrando de que forma os mesmos propõem a representação da arquitetura em diferentes perspectivas ou visões.

2.2.1 O Modelo de Visões Arquiteturais 4+1

Um dos modelos propostos na literatura para organizar as diferentes visões arquiteturais é o modelo de KRUCHTEN (1995), conhecido como modelo de visões arquiteturais 4+1 (*The 4+1 View Model*). Ele organiza a descrição da arquitetura de software em cinco visões concorrentes. Conforme pode ser observado na figura 1, cada visão trata um conjunto de objetivos específicos do projeto arquitetural de acordo com os interesses dos diferentes *stakeholders* (papéis interessados no desenvolvimento do software, como usuários finais, programadores, engenheiros etc).

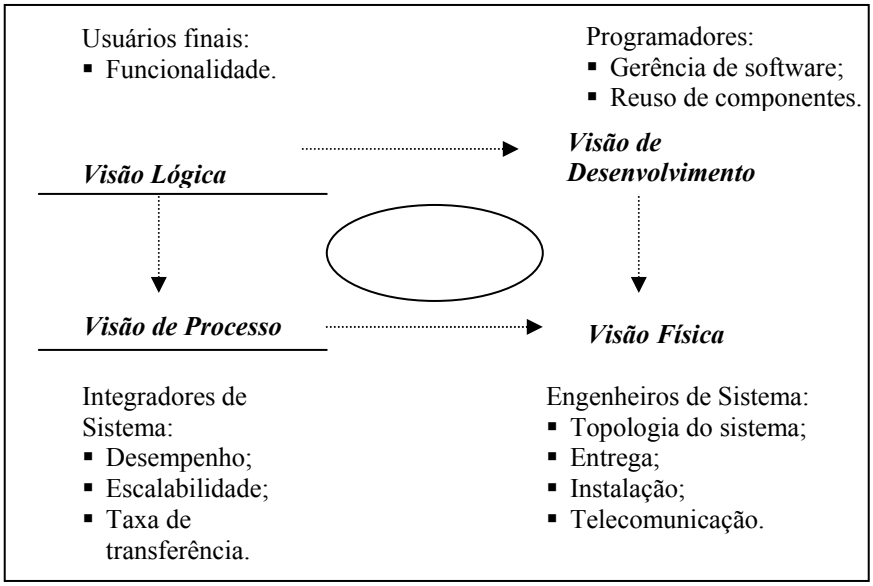

Figura 1. Visões no Modelo *The 4+1 View Model* (KRUCHTEN, 1995).

Os arquitetos de software capturam suas decisões de projeto em quatro visões, usando a quinta visão apenas para ilustrar e validar as demais. As visões propostas pelo modelo são:

- **Visão Lógica**: descreve a perspectiva estática do sistema em nível conceitual, demonstrando seus componentes e conectores. Trata o

mapeamento dos requisitos funcionais para a arquitetura, refletindo abstrações-chave do domínio do problema. Para sistemas orientados a objetos, esta visão pode ser descrita através de um modelo de pacotes e de classes, ao passo que para sistemas centrados em dados, ela poderia ser descrita através de um diagrama entidade-relacionamento.

- **Visão de Processo**: apresenta os diferentes processos do sistema, descrevendo aspectos de concorrência e sincronização do projeto.
- **Visão Física**: descreve o mapeamento do software (componentes, processos) para o hardware e reflete a natureza distribuída do sistema.
- **Visão de Desenvolvimento**: descreve a organização estática do software no seu ambiente de implementação. A visão de desenvolvimento trata da possibilidade de reutilização, ou seja, do uso de componentes de prateleira (COTS – *Commercial-off-the-shelf software components)*, de bibliotecas do ambiente operacional do sistema ou de componentes de terceiros.
- **Visão de Cenários**: ou de casos de uso, utilizada para ilustrar o comportamento do sistema em sua arquitetura, conforme descrita pelas outras quatro visões. Para tal ilustração, alguns cenários de casos de uso devem ser selecionados.

As setas na figura 1 indicam os relacionamentos entre as visões. Existe um mapeamento entre as abstrações definidas nas diferentes visões. Os elementos arquiteturais definidos na visão lógica são implementados através das construções de código fonte encontradas na visão de desenvolvimento. Os componentes arquiteturais definidos na visão lógica são mapeados para processos que, por sua vez, são mapeados para nós da rede na visão física. Por fim, as construções encontradas na visão de desenvolvimento, como as bibliotecas do ambiente de programação usadas pelo software, são alocadas a nós da rede na visão física.

KRUCHTEN (1995) ressalta em seu trabalho que nem toda arquitetura de software requer representações nas quatro visões do modelo. Visões que não são úteis podem ser omitidas da descrição arquitetural. Por exemplo, a visão física poderia ser eliminada se

existisse somente um processador ou a visão de processo poderia ser eliminada caso existisse somente um processo (linha de execução ou executável) no sistema.

2.2.2 O Modelo de 4 Visões Arquiteturais

O modelo de HOFMEISTER et al. (1999) divide a descrição de uma arquitetura de software em quatro visões: conceitual, de módulo, de execução e de código. Em função destas quatro visões, o modelo se denomina o modelo de 4 visões arquiteturais (*The 4 Views Architectural Model*). Assim como KRUCHTEN (1995), HOFMEISTER et al. (1999) prevêem a separação dos aspectos estáticos e dinâmicos do sistema em diferentes visões arquiteturais.

As diferentes visões tratam diferentes objetivos de engenharia, e a separação destes objetivos ajuda o arquiteto a tomar decisões mais sólidas com base no custo-benefício das opções de projeto (HOFMEISTER et al., 1999). As visões do modelo são descritas conforme se segue:

- **Visão Conceitual**: descreve a arquitetura do software em termos dos elementos do domínio, prevendo a decomposição do sistema em um conjunto de componentes e conectores. Neste ponto a preocupação é o mapeamento das características funcionais da aplicação para a arquitetura.
- **Visão de Módulo**: descreve a decomposição do software em subsistemas.
- **Visão de Execução**: é a visão do sistema em tempo de execução. Reflete o mapeamento dos módulos para as tarefas e processos, demonstrando a comunicação entre eles e a sua localização física. A taxa de utilização dos recursos e o desempenho são preocupações chave na visão de execução.
- **Visão de Código**: descreve como os módulos e as interfaces na visão de módulo são mapeados para os arquivos do código fonte e como as tarefas e processos da visão de execução são mapeados para os arquivos executáveis.

2.3 Representação da Arquitetura

Uma descrição arquitetural de software, conforme ressaltado na seção anterior, se compõe de diferentes perspectivas. Perspectivas estáticas, demonstrando os componentes e seus inter-relacionamentos, e perspectivas dinâmicas, demonstrando aspectos como o comportamento do sistema na sua arquitetura, concorrência e sincronização.

Para representar uma arquitetura, diferentes abordagens são propostas. A maior parte acaba dando maior enfoque à visão estrutural ou estática, embora formas de representação ou descrição para todas as visões arquiteturais sejam necessárias.

A forma de representação mais comumente encontrada nos trabalhos na literatura de engenharia de software é o uso de caixas e setas representadas sem nenhuma formalidade em diagramas arquiteturais. Esta representação não tem nenhum valor semântico e acaba conduzindo o leitor a interpretações ambíguas acerca da arquitetura do sistema. Perguntas como: o que as caixas estariam representando? Executáveis? Agrupamentos de arquivos do código fonte? Agrupamentos lógicos de funcionalidade? E as setas? O fluxo de controle ou de dados? Ou simplesmente uma dependência de compilação? São comuns a leitores tentando interpretar os diagramas arquiteturais. E a resposta a estas perguntas seria: estes símbolos representam um pouco disso tudo (KRUCHTEN, 1995). Percebe-se, dessa forma, que as caixas e setas utilizadas, símbolos altamente informais, ambicionando representar muita informação, acabam por não conduzir semântica alguma, sendo, portanto, inadequados para uma representação arquitetural.

Abordagens propostas para a representação da arquitetura compreendem o uso de especificações formais, linguagens de descrição da arquitetura (ADLs – *Architecture Description Languages*), arquiteturas específicas de domínio (DSSAs – *Domain Specific Software Architectures*), *frameworks,* os estilos e padrões arquiteturais e a utilização da UML (*Unified Modeling Language*). Nesta seção, o foco será dado ao emprego de estilos e padrões arquiteturais, ADLs e UML como formas possíveis de representação da arquitetura.

2.3.1 Estilos e Padrões Arquiteturais

Os estilos e padrões arquiteturais traduzem organizações estruturais conhecidas de sistemas de software. Eles definem, entre outros aspectos, os tipos de componentes e conectores utilizados em uma descrição arquitetural. Esta seção aborda a descrição dos estilos e padrões arquiteturais.

2.3.1.1 Estilos Arquiteturais

SHAW e GARLAN (1996) catalogaram em seu conhecido livro sobre arquitetura de software diversos estilos arquiteturais, apontando, para cada um deles, benefícios e

contrapartidas na sua escolha como direcionadores para uma dada organização arquitetural. Dentre os estilos apresentados em (SHAW e GARLAN, 1996), encontram-se: Tubos e Filtros, Camadas, Tipos Abstratos de Dados (ou Orientação a Objetos), Invocação Implícita (ou Orientado a Eventos), Repositórios e Processos Distribuídos. Os estilos definem a topologia do sistema e estabelecem restrições sobre os componentes e suas conexões.

Uma das grandes vantagens no uso de estilos e padrões, tanto na área de arquitetura, como em outras áreas da engenharia de software, é que estes estabelecem um vocabulário comum para a comunicação entre os engenheiros e desenvolvedores.

Os estilos apresentados em (SHAW e GARLAN, 1996) foram definidos observando-se a organização de sistemas existentes. Percebeu-se, com base em estudos, que os sistemas estavam organizados de forma similar e faziam uso de estruturas semelhantes. Pode-se dizer, portanto, que um estilo de arquitetura caracteriza uma família de sistemas em termos de seus padrões de organização estrutural (SHAW, 1993).

Os estilos arquiteturais definem quatro propriedades importantes dos sistemas (MONROE et al. 1997), a saber:

- Um vocabulário dos elementos de projeto de alto nível, envolvendo os tipos de componentes e conectores que podem ser empregados;
- Regras de projeto determinando as composições de componentes permitidas;
- Interpretação semântica, onde a composição dos componentes, restringida pelas regras de projeto, tem um significado bem definido;
- Análises que podem ser realizadas sobre sistemas construídos em um estilo. Exemplos incluem análises de agendamento de tarefas para estilos orientados ao processamento em tempo real e detecção de *deadlock* (congelamento do sistema por espera de recursos) para o estilo baseado em passagem de mensagens entre clientes e servidores.

Para uma maior compreensão dos estilos, uma classificação dos mesmos se faz necessária. Porém, SHAW e GARLAN (1996) defendem a falta de uma taxonomia compreensiva para os idiomas arquiteturais. Eles apresentam uma lista com algumas categorias propostas, a qual se apresenta na tabela 1.

Sistemas de Fluxo de Dados	Sistemas de Chamada e Retorno	Componentes Independentes	Máquinas Virtuais	Sistemas Centrados em Dados (Repositórios)
Seqüenciais *Batch*	Programa Principal e Sub-rotinas	Processos Comunicantes	Interpretadores	Bancos de Dados
Tubos e Filtros (*Pipes & Filters)*	Sistemas Orientados a Objetos	Sistemas Orientados a Eventos	Sistemas Baseados em Regras	Sistemas de Hipertexto
	Camadas Hierárquicas			Quadro negro (*Blackboards)*

Tabela 1. Taxonomia Proposta para Estilos Arquiteturais (SHAW e GARLAN, 1996).

Um aspecto importante a ser ressaltado é que dificilmente uma aplicação segue um único estilo arquitetural. Normalmente, os estilos são utilizados de forma combinada. O mais comum de fato é encontrarmos arquiteturas heterogêneas para sistemas de computador. Há diversas formas de se combinar os estilos arquiteturais em uma aplicação. Dentre os recursos utilizados para a combinação dos estilos, encontram-se:

- **Hierarquia**: um componente de um sistema organizado segundo um estilo arquitetural pode ter sua estrutura interna desenvolvida em um estilo completamente diferente. Como exemplo, podemos citar um sistema em camadas no qual cada camada internamente está organizada segundo o estilo de Orientação a Objetos ou Tipos Abstratos de Dados. Além dos componentes, os conectores também podem ser hierarquicamente decompostos.
- **Diferentes Tipos de Conectores**: um componente utiliza diferentes tipos de conectores. Por exemplo, um componente pode ter uma interface para acesso a um repositório e, ao mesmo tempo, interagir através de passagem de mensagem com outros componentes em um sistema. Seria um caso típico em um sistema orientado a objetos utilizando banco de dados relacional, onde algumas classes acessam um repositório utilizando uma interface de comandos em SQL[2] (*Structured Query Language)* e ao mesmo tempo seus objetos se comunicam com outros objetos via troca de mensagens.

[2] A SQL é a linguagem padrão para acesso a bancos de dados relacionais. Ela apresenta comandos para definição de dados, através da LDD (Linguagem de Definição de Dados) e comandos para manipulação dos dados, ou seja, para inclusão, alteração, exclusão e consulta de registros, através da LMD (Linguagem de Manipulação de Dados).

- **Diferentes Níveis de Descrição Arquitetural**: elaborar cada nível de descrição arquitetural em um estilo arquitetural diferente.

A tabela 2 resume as características dos principais estilos arquiteturais descritos na literatura (SHAW e GARLAN, 1996) (MENDES, 2002).

Estilo	Componentes	Conectores	Regras	Questões de Controle e de Dados	Benefícios	Contrapar-tidas
Camadas	Camadas (geralmente representam agregados de componentes menores)	Protocolos que determinam como as camadas interagem (geralmente, chamadas de procedimentos).	Restrições topológicas: as camadas só devem se comunicar com as camadas adjacentes.	O controle e os dados fluem da camada mais acima para a camada mais baixa da arquitetura, retornando no sentido contrário. A topologia é hierárquica.	Suporte ao reuso, manutenção e evolução do esquema.	Degradação do desempe-nho e complexidade de implemen-tação.
Organização Orientada a Objetos (ou Abstração de Dados)	Objeto ou Tipo Abstrato de Dados que encapsula dados e operações.	Envio de mensagem.	Os objetos são responsáveis pela manutenção da integridade de sua representação. A representação é escondida dos outros objetos.	Objetos transmitem o controle e dados a outros objetos através do envio de mensagem. Este controle retorna (bem como dados) como resposta à mensagem.	Possibilidade da alteração da implementação de um objeto sem afetar os seus clientes. Decom-posição de proble-mas em uma coleção de agentes colaboradores.	Objetos precisam conhecer a identidade dos outros. Modificações na interface dos objetos afeta seus clientes.
Tubos e Filtros	Filtros: transformadores de dados.	Tubos (ou *pipes*): transmissores de cadeias de dados (*streams*).	Filtros são independentes.	Os dados e o controle fluem de um filtro a outro através dos *pipes*, seguindo uma topologia linear. Em alguns casos pode apresentar ciclos.	Suporte ao reuso, à evolução e manutenção do esquema.	Inadequado para sistemas interativos.
Processos Distribuídos	Processos, que podem ser: filtros, clientes, servidores ou pares.	Invocação remota de procedimentos (RPC) , comunicação entre clientes e servidores via soquetes etc.	O servidor não conhece a identidade e o número de clientes. Clientes conhecem o servidor.	O servidor pode trabalhar de forma síncrona, retornando o controle junto com os dados aos clientes; ou, assíncrona, retornando somente os dados.	Escalabilidade e facilidade de modificações.	Clientes têm que aguardar o retorno do servidor para continuar o seu trabalho.

Tabela 2. Resumo dos Principais Estilos Arquiteturais.

Estilo	Componentes	Conectores	Regras	Questões de Controle e de Dados	Benefícios	Contrapar-tidas
Repositórios	Repositório de dados central e componentes independentes que operam sobre o repositório.	A comunicação entre os componentes se dá exclusiva-mente através do acesso ao repositório compartilhado.	Dependem da organização dos dados utilizada.	O fluxo de controle ocorre duas formas: transações disparam os processos (BD tradicionais); estado corrente do repositório é responsável pela seleção dos processos (quadro- negro ou *blackboard*).	Garantem a consistência dos dados. Comunicação entre componentes se dá via o compartilha-mento de dados.	Organização dos dados explícita. Componentes precisam de interface de acesso ao repositório. Necessidade de controle de concorrência no acesso aos recursos compartilha-dos.
Baseado em Eventos (ou Invocação Implícita)	Componentes que apresentam uma coleção de eventos em sua interface. Componentes que registram interesse em um evento, associando um procedimento a ele (*listeners*).	Amarração entre o anúncio de um evento e a chamada do procedimento registrado para o evento. O próprio sistema faz esta amarração em tempo de execução.	Os anunciantes dos eventos não sabem que componentes serão afetados pelo evento.	Interação assíncrona entre componentes. Eventos podem transmitir dados.	Suporte à reutilização. Facilidade de evolução do esquema, uma vez que os componentes não conhecem a identidade uns dos outros.	Componentes não têm controle sobre a computação realizada no sistema. Verificação da corretude do sistema pode ser problemática.

Tabela 2. Resumo dos Principais Estilos Arquiteturais (continuação).

2.3.1.2 Padrões Arquiteturais

Com relação aos padrões arquiteturais, a maior fonte de referência é (BUSCHMANN et al., 1996), sendo que os padrões arquiteturais oferecem uma estrutura mais concreta que os estilos, estabelecendo um conjunto de subsistemas pré-definidos, com suas responsabilidades.

O termo padrão começou a ser empregado primeiramente na área de arquitetura através de definição dada por ALEXANDER et al. (1977):

"Um padrão descreve um problema que ocorre repetidas vezes em nosso meio e inclui uma solução genérica para o mesmo, de tal maneira que se pode usá-la mais de um milhão de vezes, sem nunca fazê-lo de forma idêntica".

Embora tenha sido reconhecida a importância do uso de padrões inicialmente para projetos de cidades e urbanismo, o conceito apresenta grande importância na área de engenharia de software. Para apoiar o desenvolvimento de software, nas suas diferentes fases, três categorias de padrões são fundamentais:

- **Padrões Arquiteturais**: expressam um esquema de organização estrutural fundamental para sistemas de software, definindo seus subsistemas, especificando suas responsabilidades e incluindo regras e orientações para a organização dos relacionamentos entre os subsistemas (BUSCHMANN et al., 1996).
- **Padrões de Projeto**: provêem um esquema para o refinamento dos subsistemas (ou componentes) de um sistema de software e dos relacionamentos entre eles. Descrevem uma estrutura recorrente de objetos comunicantes que resolve um problema de projeto genérico dentro de um contexto particular (GAMMA et al., 1995).
- **Padrões de Codificação (ou Idiomas)**: são padrões de baixo nível de abstração, específicos para uma linguagem de programação particular. Descrevem como implementar aspectos particulares de componentes ou dos relacionamentos entre eles, usando as características de uma dada linguagem de programação (BUSCHMANN et al., 1996).

Os padrões arquiteturais, que são os de nosso interesse neste trabalho, apresentam uma maior escala do que os padrões de projeto e idiomas. Sua aplicação afeta a organização estrutural do sistema como um todo. Como exemplo de padrão arquitetural, pode-se citar o conhecido MVC (*Model-View-Controller*) (KRASNER e POPE, 1988), que define um sistema em termos de três tipos de componentes:

- **Modelo**: encapsula os dados e funcionalidade do negócio. O modelo deve ser independente de representações de saída específicas e do tratamento das interações dos usuários com a aplicação.

- **Visão:** mostra as informações para os usuários, sendo responsável pelos formatos de saída específicos. Uma visão obtém os dados que ela mostra a partir do modelo. Pode haver múltiplas visões de um mesmo modelo.
- **Controlador:** tratam os eventos de entrada dos usuários disparados nas interfaces. Eventos são traduzidos em requisições de serviços ao modelo ou à visão. Cada visão tem um componente controlador associado.

A separação do modelo dos componentes de controle e da visão permite múltiplas visões do mesmo modelo. Se o usuário altera o modelo via o controlador de uma visão, todas as outras visões dependentes dos dados do modelo devem ser atualizadas para refletir esta mudança. Para que esta atualização seja possível, o modelo notifica todas as suas visões a cada vez que seu estado se modifica. Figura 2 ilustra o diagrama de classes genérico em UML[3] para o padrão MVC.

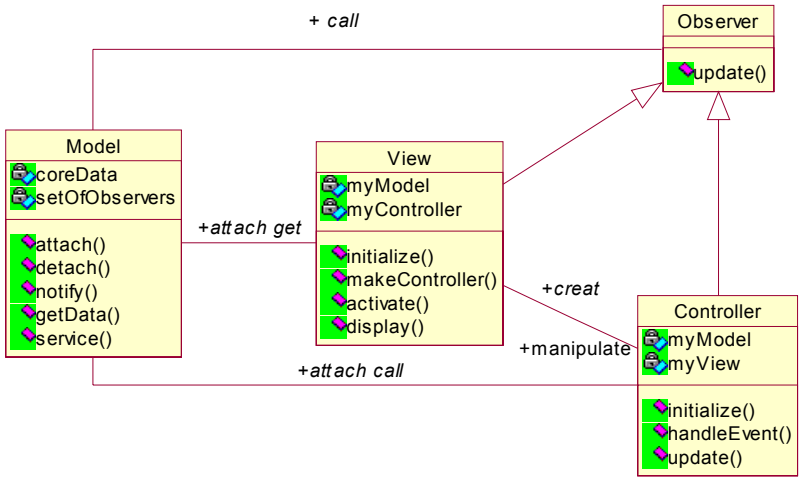

Figura 2. Modelo de Classes do Padrão MVC Adaptado de BUSCHMANN et al. (1996).

Utilizando este padrão para a instanciação da arquitetura de uma aplicação, obtém-se flexibilidade para a modificação das interfaces. O código central do negócio fica desacoplado das suas formas de representação. Portanto, este padrão se mostra adequado ao desenvolvimento de aplicações interativas.

[3] UML (*Unified Modeling Language*) é a linguagem de modelagem para sistemas orientados a objetos que surgiu da unificação dos principais métodos de modelagem OO e foi adotada pelo OMG (*Object Management Group*) em 1997 como notação padrão para análise e projeto OO.

A partir da apresentação do MVC, pode-se ter uma melhor idéia da forma como os padrões arquiteturais visam auxiliar na solução de problemas de projeto arquitetural recorrentes. Embora tenha sido ressaltado no início desta seção que os padrões apresentam estruturas mais concretas que os estilos arquiteturais, convém ressaltar que alguns dos padrões arquiteturais apresentados em (BUSCHMANN et al., 1996) são também estilos arquiteturais em (SHAW e GARLAN, 1996), como é o caso do padrão tubos e filtros e camadas. Outros exemplos de padrões arquiteturais bastante utilizados, apresentados em (BUSCHMANN et al., 1996) incluem:

- *Broker:*para aplicações distribuídas, onde uma aplicação pode acessar serviços de outras aplicações simplesmente pelo envio de mensagens a objetos mediadores, sem se preocupar com questões específicas relacionadas à comunicação entre processos, como a sua localização.
- *Presentation-Abstraction-Control (PAC):* define uma estrutura para sistemas na forma de uma hierarquia de agentes cooperativos. Adequado para sistemas interativos, onde cada agente é responsável por um aspecto específico da funcionalidade da aplicação e é composto por três componentes: apresentação, abstração e controle.
- *Microkernel:* propõe a separação de um conjunto de funcionalidades mínimas das funcionalidades estendidas e partes específicas de clientes. O encapsulamento dos serviços fundamentais da aplicação é realizado no componente "*microkernel*". As funcionalidades estendidas e específicas devem ser distribuídas entre os componentes restantes da arquitetura.

2.3.1.3 Suporte à Seleção de Estilos e Padrões Arquiteturais

Um outro importante aspecto a ser levado em conta, considerando-se o uso de estilos e padrões arquiteturais, é a seleção dos mesmos durante o projeto arquitetural de software. Cada estilo\padrão arquitetural é adequado a uma classe de problemas, mas nenhum é adequado para solucionar todos os problemas (SHAW e CLEMENTS, 1997). O arquiteto de software (ou projetista) precisa de algum suporte para a tomada de decisão neste momento. Basear-se apenas na experiência pode ser perigoso por dois motivos: primeiro, o projetista pode não conhecer uma grande variedade de estilos e padrões

arquiteturais; segundo, ele pode não fazer a melhor escolha pelo fato de estar acostumado a usar um ou outro padrão.

Para oferecer este suporte, algumas abordagens são propostas na literatura. Dentre elas, podemos destacar:

- A seleção de um estilo/padrão arquitetural com base nos requisitos não-funcionais desejados para o sistema. Esta abordagem é apresentada em (XAVIER, 2001), onde são definidos os requisitos não-funcionais[4] (ou atributos de qualidade) desejados para o software - como interoperabilidade, segurança de acesso, maturidade, tolerância a falhas etc - com o grau de importância destes para a aplicação[5]. Ao mesmo tempo, é atribuído um peso aos requisitos não-funcionais para cada padrão arquitetural de acordo com o grau de esforço em se obter aquele atributo de qualidade com a aplicação do padrão. Mediante os valores atribuídos, a ferramenta apresentada em (XAVIER, 2001) é capaz de gerar uma lista ordenada dos padrões arquiteturais de acordo com o seu grau de adequação para o sistema.
- Basear a seleção em dois tipos de informação: classificação das arquiteturas candidatas e diretrizes de projeto a respeito de como realizar a escolha adequada (SHAW e CLEMENTS, 1997). A classificação envolve a discriminação dos estilos arquiteturais com base na distinção de suas características. Esta distinção foi realizada neste trabalho na tabela 2. As diretrizes são na realidade heurísticas que indicam, com base na experiência dos autores, possíveis caminhos a seguir. Como exemplos destas heurísticas, tem-se: para problemas que podem ser decompostos em estágios seqüenciais, considere o uso de arquiteturas *pipelines*[6] ou seqüenciais-*batch*; para problemas cuja solução requer um alto grau de flexibilidade, acoplamento fraco entre as tarefas e tarefas reativas, considere o uso de processos comunicantes. Uma lista destas heurísticas pode ser obtida em (SHAW e CLEMENTS, 1997).

[4] O trabalho de XAVIER (2001) utiliza as características e sub-características de qualidade da ISO/IEC 9126-1.

[5] Convém ressaltar que o trabalho de XAVIER (2001) visa a criação de arquiteturas de referência para um domínio, embora o mesmo raciocínio possa ser utilizado para o nível arquitetural de uma aplicação em particular.

[6] *Pipeline* representa uma variação do estilo tubos e filtros, que restringe a topologia a seqüências lineares de filtros.

- Métodos de análise da arquitetura. Para a seleção de um estilo/padrão arquitetural, aplique métodos de análise da arquitetura, como o SAAM (*Software Architecture Analysis Method*) (KAZMAN et al., 1994) e o ATAM (*Architecture Tradeoff Analysis Method*) (KAZMAN et al., 1999a).

2.3.2 Linguagens de Descrição Arquitetural (ADLs)

Uma outra forma de representar arquiteturas, além do uso de estilos e padrões arquiteturais, é através do uso de ADLs. Uma variedade de linguagens para projeto arquitetural vem sendo proposta a fim de prover os arquitetos de software de notações para a especificação e análise das arquiteturas (MONROE et al., 1997). As linguagens de descrição arquitetural (ADLs) visam buscar uma representação mais formal para as arquiteturas de software do que os diagramas informais comumente usados. Embora elas pareçam vir a resolver os problemas relacionados à representação arquitetural, as ADLs ainda não apresentam uso expressivo no projeto de software. Este problema se deve a uma série de fatores, dentre eles: a complexidade das ADLs, a falta da compreensão da comunidade de engenharia de software acerca das abstrações arquiteturais, a indefinição do que de fato representa uma ADL e a não existência de uma linguagem padrão para a descrição de arquitetura de software. As ADLs representam tema de pesquisa na área de arquitetura de software desde a década de 90.

Uma ADL deve descrever os componentes e os conectores em uma arquitetura. Os componentes representam o local onde uma computação é realizada e o local onde um estado é estabelecido. Componentes possuem interfaces, que especificam os seus serviços providos e requeridos e são definidas através de portas. As portas determinam os pontos de interação do componente com o seu ambiente. Para descrever um componente, as seguintes seções são sugeridas em (SHAW e GARLAN, 1996): interface, tipo (e.g. filtro, processo, banco de dados etc.), participante (*player*) e implementação.

Conectores, por sua vez, representam os blocos de construção que descrevem as interações entre os componentes e as regras que governam estas interações. Diferentemente dos componentes, conectores podem não corresponder a unidades de compilação no sistema implementado. Eles podem se encontrar espalhados pelo código fonte dos componentes. Os conectores também possuem interfaces, as quais são definidas através de um conjunto de papéis. Os papéis identificam os participantes (componentes) esperados em

uma interação. SHAW e GARLAN (1996) argumentam que os conectores devem ser tratados como entidades de primeira grandeza (*first-class entities*), assim como os componentes. Eles sugerem que os conectores devem ser descritos na arquitetura como elementos independentes, sendo especificados através dos seguintes itens de descrição: protocolo de comunicação (regras, compromissos) entre participantes, tipo (e.g. *pipe*, chamada de procedimento etc), papéis e implementação.

Além de descrever os componentes e conectores, uma ADL deve apresentar um conjunto de características (SHAW e GARLAN, 1996). As características estão divididas em seis áreas, conforme se segue:

1. **Composição**: deve ser possível descrever um sistema como uma composição de componentes e conexões independentes. Uma ADL deve permitir a um projetista dividir um sistema complexo hierarquicamente em componentes menores e mais gerenciáveis, e, por outro lado, deve permitir também que um sistema grande e complexo possa ser formado a partir da combinação de partes independentes. A composição é uma característica recursiva, extensível aos componentes e conectores arquiteturais, permitindo que uma arquitetura seja descrita em diferentes níveis de detalhe. Dessa forma, os componentes podem ser primitivos, apresentando uma interface e uma implementação, ou compostos, apresentando, neste caso, uma lista com as suas partes constituintes. Os conectores também podem ser primitivos ou compostos.
2. **Abstração**: deve ser possível descrever em um alto nível de abstração os componentes e os conectores de um sistema, sem se deter a detalhes de implementação, de forma que os papéis destes elementos fiquem claros na arquitetura. Por exemplo, abstrações arquiteturais em uma ADL devem ser capazes de indicar que os componentes estão relacionados por um relacionamento cliente-servidor, independente da forma como os componentes e o relacionamento estejam implementados.
3. **Configuração**: uma ADL deve permitir a descrição da estrutura do sistema (sua topologia), independentemente dos elementos sendo estruturados. De certa forma, a característica de configuração nos remete à característica de composição descrita anteriormente: a configuração determina a forma como os componentes e os

conectores vão estar combinados para formar uma arquitetura, o que traz a necessidade de descrever esta combinação sem levar em conta a descrição da estrutura interna de cada elemento. Para o estabelecimento de uma configuração arquitetural, é necessário buscar uma correspondência entre as portas dos componentes e os papéis definidos pelos conectores. Os componentes são os participantes do sistema que se comunicam via conectores desde que o protocolo de comunicação que um conector estabelece e os papéis que este define sejam compatíveis com aquilo que cada participante espera.

4. **Reusabilidade:** deve ser possível reutilizar componentes, conectores e padrões de relacionamento componente-conector (como uma estrutura cliente-servidor), em descrições arquiteturais diferentes daquela para a qual os elementos foram originalmente especificados.
5. **Heterogeneidade:** a heterogeneidade envolve dois aspectos: primeiramente, deve ser possível combinar diferentes estilos arquiteturais em uma descrição arquitetural; segundo, a combinação de componentes escritos em diferentes linguagens de programação também deve ser permitida. Com relação ao segundo aspecto, uma vez que as descrições arquiteturais encontram-se em um nível de abstração mais alto que as estruturas encontradas em linguagens de programação específicas, estas não devem impedir a interoperabilidade entre componentes.
6. **Análise**: deve ser possível realizar ricas e diversificadas análises da arquitetura com base em suas descrições. Estas análises podem ser automatizadas (realizadas através de simulações) ou não-automatizadas.

Em (SHAW e GARLAN, 1996) e (MEDVIDOVIC e TAYLOR, 2000), é apresentado um conjunto de ADLs conhecidas, com suas características e suporte ferramental. A tabela 3 apresenta algumas ADLs de destaque descritas nestes trabalhos.

ADL	Foco	Composição/ Configuração	Heterogenei-dade	Conec-tores (pri-meira classe)	Visões	Ferramental
Unicon	Compilação de descrições arquiteturais para a geração de código executável.	Não apresenta construções explícitas para modelar arquiteturas, utilizando componentes compostos.	Estilos: Sim. Ling.Progr.: Não	Sim.	Permite o mapeamento de construções do código para construções de mais alto nível de abstração.	Editor gráfico que opera de maneira pró-ativa, evitando erros no projeto ao longo da especificação da arquitetura. .
Aesop	Sistema para a geração de ambientes de projeto arquitetural para estilos específicos.	Configuração explícita através de representação gráfica.	Desenvolvimento de componentes em C++. Combinação de estilos através da aplicação de estilos específicos em diferentes níveis de descrição arquitetural.	Sim.	Múltiplas visões, específicas de estilo.	Ambientes apresentam paleta de tipos de componentes e conectores e checagem de consistência com base no estilo.
Wright	Provê uma base formal para especificar as interações entre componentes através de protocolos.	Especificações textuais explícitas e concisas. Permite tanto componentes quanto conectores compostos.	Sim.	Sim.	Visão estática/ lógica. Especificação independente de implementa-ção.	Ferramentas para análises especializa das (e.g.,*deadlock*).
Rapide	Enfatiza especifica-ções comportamentais e a simulação de projetos arquiteturais.	Descrições de configuração confusas devido à apresentação explícita de detalhes de conectores.	Sim.	Não.	Visualização do comportamen-to dinâmico da aplicação via simulação.	Ferramentas para simulação da execução e ferramentas de animação para animar a execução.

Tabela 3. Análise de ADLs existentes.

Além das ADLs apresentadas na tabela 3, MEDVIDOVIC e TAYLOR (2000) falam de outras linguagens, como: C2 e DARWIN, para sistemas distribuídos e ACME, que visa suportar o mapeamento entre especificações arquiteturais de uma ADL para outra.

Uma outra ADL mais recente que merece destaque é a xADL (DASHOFY et al., 2002). A xADL é na verdade uma meta-ADL baseada em esquemas XML que permite a geração de ADLs através da composição e extensão destes esquemas. Além da meta-ADL, os autores propõem uma ADL de fato, a xADL 2.0 (DASHOFY et al., 2001), que é construída a partir dos esquemas pré-definidos pela xADL. Os benefícios da xADL incluem a possibilidade de geração de ADLs que atendam a diferentes necessidades e a descrição da própria arquitetura em XML, o que permite a sua interpretação e análise por ferramentas externas. Com relação ao ferramental, a xADL possui um conjunto de ferramentas de

suporte, as quais são capazes de se adaptar à medida que as características das ADLs evoluem ou se modificam.

2.3.3 Utilizando UML para a Representação da Arquitetura

Visto que a UML representa a notação padrão para sistemas orientados a objeto e que apresenta um largo uso, tanto em nível acadêmico quanto industrial, a sua utilização na representação da arquitetura vem sendo contemplada em diversos trabalhos encontrados na literatura.

Em (HOFMEISTER et al., 1999), os autores demonstram de que forma a UML pode ser utilizada como notação para a representação da arquitetura, seguindo o modelo *The 4 Views Architectural Model* (veja seção 2.2.2). Para cada visão arquitetural no modelo, os autores apresentam um tipo de diagrama ou elementos de modelagem da UML que poderiam ser aplicados, conforme se segue:

- **Visão Conceitual**: classes estereotipadas para representar os componentes, suas portas e os conectores. Para os conectores e portas, símbolos especiais são associados. Os papéis nos conectores são representados através de papéis nas associações. Os diagramas de classes são utilizados para representar a configuração estática do sistema, diagramas de seqüência e de estado para demonstrar os protocolos aos quais as portas (interfaces) dos componentes aderem e diagramas de seqüência da UML para demonstrar seqüências de interações particulares entre grupos de componentes.
- **Visão de Módulo**: módulos são representados como classes estereotipadas e subsistemas como pacotes estereotipados. Diagramas de pacotes e de classes da UML são utilizados para demonstrar a decomposição dos subsistemas em módulos e as dependências de uso entre os módulos. Tabelas são utilizadas para demonstrar o mapeamento entre as visões conceitual e de módulo.
- **Visão de Execução**: a visão de execução descreve como os módulos serão combinados em um produto particular, sendo atribuídos a imagens de *run-time* (imagens do sistema em tempo de execução, como processos e tarefas). Classes estereotipadas da UML são utilizadas para representar estas imagens e os caminhos de comunicação entre elas são representados através de associações entre as classes. Os diagramas de classe da UML são utilizados para representar a

configuração estática do sistema, formada pelos processos e suas comunicações. Diagramas de seqüência são utilizados para demonstrar o comportamento dinâmico de uma configuração.

- **Visão de Código**: arquivos do código fonte, arquivos binários e os executáveis são representados como classes estereotipadas e os diretórios onde os mesmos se localizam são representados como pacotes estereotipados. Relacionamentos de inclusão, compilação e ligação são mostrados como dependências estereotipadas. Tabelas são utilizadas para descrever o mapeamento entre elementos nas visões de módulo e de execução para os elementos na visão de código.

KRUCHTEN (2000) utiliza a UML para representar os elementos das visões arquiteturais propostas no seu modelo, *The 4+1 View Model* (seção 2.2.1) (KRUCHTEN, 1995). Para a visão lógica, ele utiliza os diagramas de pacotes e de classes da UML. Os componentes são representados através de pacotes e de classes e os conectores através dos relacionamentos de associação, agregação/composição, dependência e generalização. Ambos são tratados como elementos de primeira classe, aos quais podem ser atribuídas diversas propriedades.

A visão de desenvolvimento (ou de implementação) é representada através do elemento de modelagem componente da UML e de pacotes. Cada componente da UML representa um módulo do software (arquivo fonte, binário, executável, uma DLL - biblioteca de ligação dinâmica - etc) com uma interface bem definida. Porém, conforme já foi ressaltado anteriormente, estes componentes não são aqueles empacotados, encapsulados e independentes tratados pelos métodos de DBC. Aqui se tem uma visão mais genérica de componente. Os pacotes representam um subsistema na visão de implementação, consistindo de um conjunto de componentes. Figura 3 apresenta exemplos de componentes da UML e um subsistema na visão de implementação.

A visão física é representada pelo diagrama de implantação (*deployment diagram*) da UML, o qual é capaz de retratar os processadores em uma rede e a alocação de processos aos processadores. A visão de cenários pode ser obtida através do modelo de casos de uso e da realização dos casos de uso via colaborações entre objetos.

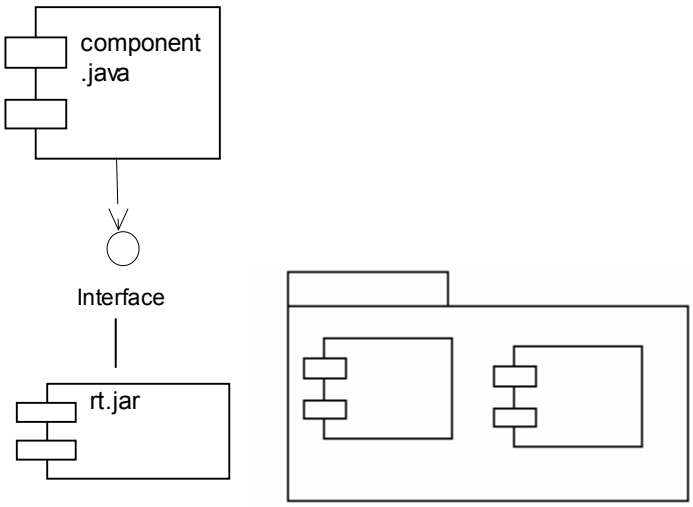

Figura 3. Componentes da UML e Subsistema na Visão de Implementação.

2.4 Arquiteturas de Software no Contexto da Engenharia de Domínio

No contexto da Engenharia de Domínio (ED), a arquitetura representa um dos possíveis artefatos para a reutilização. A ED produz artefatos reutilizáveis, em diferentes níveis de abstração, para a construção de aplicações em um dado domínio. Em nível de análise, a ED se preocupa em produzir modelos de requisitos do domínio, e em nível de projeto e implementação, a ED visa a produção de arquiteturas específicas de domínio (DSSAs) e de componentes de software (i.e. código), compatíveis com esta arquitetura. Estes artefatos representam pré-requisitos uns para os outros e apresentam ligações entre si, o que permite que a reutilização na construção de aplicações no domínio seja atingida desde o início do ciclo de vida. Nesta seção, o nosso foco é em arquiteturas específicas de domínio (ou arquiteturas de referência).

Uma DSSA representa uma arquitetura possível para um domínio, sendo descrita através de uma coleção de elementos arquiteturais, especializados para um determinado tipo de tarefa (serviços do domínio), generalizados para que seja possível seu uso efetivo no domínio, e compostos em uma estrutura padronizada (topologia) efetiva para a construção de aplicações (HAYES, 1994). Por apresentar uma característica de generalização, buscando atender uma família de aplicações, o custo de criação de uma DSSA é bem maior do que o custo de criação de uma arquitetura para uma aplicação específica (WENTZEL, 1994). Porém, a expectativa na redução dos gastos em desenvolvimento pode ser observada

através da diminuição dos custos de desenvolvimento de uma aplicação (em aproximadamente um quarto), e dos custos em manutenção de aplicações (cerca de um terço), baseados em relatos de experiência de programas de reutilização de software descritos na literatura (JONES, 1986) (TRACZ, 1987) (WENTZEL, 1994).

A fim de apoiar a construção de arquiteturas específicas de domínio, métodos e infra-estruturas de suporte à sua geração vêm sendo propostos. Dentre eles, destacam-se os esforços de pesquisa do projeto ARPA (METTALA e GRAHAM, 1992), propondo processos de criação e instanciação de uma DSSA, e a ferramenta MENTOR, integrada ao ambiente de reutilização Odyssey, a qual apóia a criação e instanciação de arquiteturas de referência de domínio, através do suporte à seleção e instanciação de padrões arquiteturais (XAVIER, 2001). Na ferramenta MENTOR, a sugestão de padrões arquiteturais para o domínio é feita com base nos requisitos não-funcionais privilegiados pelo engenheiro do domínio. Os estilos e padrões arquiteturais (veja tabela 2) privilegiam alguns atributos de qualidade de software em detrimento de outros. A análise de sistemas legados também representa uma alternativa para suportar a geração de arquiteturas de referência para o domínio. No trabalho de O'BRIEN e SMITH (2002), por exemplo, as arquiteturas de aplicações legadas são recuperadas e comparadas para suportar a criação de uma arquitetura de Linha de Produtos[7], através da análise das suas semelhanças e variações.

No capítulo 5, deste trabalho, assim como em (O'BRIEN e SMITH, 2002), será explorada a criação de arquiteturas de referência para domínios com base na análise de sistemas legados.

2.5 Considerações Finais

Conforme pôde ser observado ao longo deste capítulo, o projeto de software em nível arquitetural traz uma série de vantagens ao desenvolvimento de sistemas. A arquitetura ajuda a gerenciar a complexidade do sistema, facilita a comunicação das decisões de projeto entre os diferentes interessados no software (usuários, gerentes, desenvolvedores etc), suporta os requisitos não-funcionais do software, orienta o projeto e a evolução dos sistemas, permite a realização de análises especializadas (*deadlock*, satisfação

[7] Uma linha de produtos representa um conjunto de sistemas de software que compartilham características comuns, satisfazendo às necessidades específicas de um segmento de mercado particular, e que são desenvolvidos a partir de um conjunto comum de artefatos centrais (SEI, 2004).

de atributos de qualidade estabelecidos, como desempenho, confiabilidade, segurança etc) etc. O projeto arquitetural representa o passo necessário para orientar o projeto detalhado e a construção do software. A arquitetura orienta não só o desenvolvimento, mas as manutenções do sistema. A partir de uma arquitetura devidamente documentada, futuras manutenções e evoluções da aplicação são direcionadas pela sua estrutura e suas restrições.

Um outro importante benefício da arquitetura é a possibilidade de reutilização. Componentes de software para serem reutilizados precisam ser compatíveis com a estrutura do sistema. Eles precisam ser encaixados em pontos específicos da arquitetura, atender aos seus requisitos de qualidade e apresentar compatibilidade no que tange às suas interfaces providas e requeridas e as interfaces dos demais componentes.

A área de arquitetura representa tema de pesquisa na engenharia de software desde a década de 80. Diversos temas relacionados à arquitetura vêm sendo explorados ainda nos dias atuais, como: linguagens e notações para a descrição da arquitetura, arquiteturas específicas de domínio (DSSAs), métodos para a recuperação da arquitetura de sistemas legados etc. Alguns destes temas pretendem ser investigados na proposta apresentada neste trabalho.

3 Técnicas para a Recuperação da Arquitetura de Software de Sistemas Legados

3.1 Introdução

Embora a formalização do conhecimento sobre arquitetura seja recente na área de engenharia de software, os sistemas de software vêm sendo desenvolvidos ao longo de décadas seguindo determinados padrões de organização estrutural. Estilos arquiteturais, como o da divisão hierárquica de sistemas em programa principal e sub-rotinas, vêm sendo empregados há muitos anos pelos desenvolvedores de software, representando uma das organizações estruturais possíveis de ser encontrada em sistemas legados[8].

As grandes organizações apresentam, em geral, uma significativa base de sistemas legados. Eles representam um alto esforço em desenvolvimento já investido e trazem consigo um rico conhecimento sobre o negócio, o qual muitas vezes não pode mais ser obtido de nenhuma outra fonte de informação disponível na organização. Compreender estes sistemas e a sua organização estrutural vêm sendo a preocupação constante dos engenheiros de software. Conforme afirmam KAZMAN e CARRIÈRE (1997), o desenvolvimento de software raramente se inicia do zero. Ele é, normalmente, restringido por compatibilidade com, ou uso de sistemas legados. Além disso, a fim de manterem seus sistemas em operação, as organizações têm buscado a modernização destes sistemas, adaptando-os a novas tecnologias.

Porém, os sistemas legados raramente apresentam uma documentação de apoio que facilite a sua compreensão e esta, quando existe, se encontra em geral defasada em relação a atual implementação do software. O código fonte acaba sendo a única fonte de informação disponível para a compreensão dos sistemas. Devido ao fato do código se apresentar em um nível de abstração baixo, por ser direcionado a ferramentas do gênero dos compiladores e das dimensões de um software serem, geralmente, grandes (considerando a quantidade de linhas de código), o entendimento destes sistemas acaba se tornando uma tarefa não trivial. É necessário, então, prover suporte, de preferência automatizado, a este processo (VERONESE e NETTO, 2001).

[8] Sistemas legados é um eufemismo para sistemas grandes, geralmente antigos, cuja documentação inexiste ou se encontra desatualizada, e para o qual os desenvolvedores originais não se encontram mais disponíveis.

Neste contexto, a fim de auxiliar a compreensão de programas, se insere a engenharia reversa, que representa o processo de análise dos componentes do sistema e dos seus relacionamentos, a fim de descrever este sistema em um nível de abstração mais alto do que o código fonte original (GANNOD, 1999). Para a compreensão da organização estrutural, ou da arquitetura dos sistemas legados, uma forma particular de engenharia reversa vem sendo buscada: a engenharia reversa da arquitetura ou a recuperação da arquitetura de software. A recuperação da arquitetura de sistemas legados envolve um conjunto de métodos para a extração de informações arquiteturais a partir de representações de mais baixo nível de um software, como o seu código fonte (SARTIPI et al., 1999). RIVA e RODRIGUEZ (2002) afirmam que a descrição arquitetural de um software comunica decisões de projeto essenciais, as quais foram tomadas no passado, com relação aos sistemas legados. Recuperar a arquitetura requer, portanto, a recuperação de decisões passadas de projeto, levando o desenvolvedor a inferir informações arquiteturais que não estão imediatamente evidentes.

Porém, o tipo de informação recuperada com a aplicação de técnicas tradicionais de engenharia reversa apresenta uma escala diferente se comparada à informação desejada em um processo de recuperação da arquitetura: árvores sintáticas ou diagramas de fluxos de dados descrevem detalhadamente as estruturas de um sistema, estando em um nível de abstração de projeto detalhado. Uma descrição arquitetural se encontra em um nível mais alto de abstração do que as representações de informações extraídas diretamente do código. Ela contém ainda uma semântica que traduz a natureza da aplicação (estilo arquitetural, conceitos etc.) e descreve relacionamentos com componentes externos (base de dados, bibliotecas etc.), que muitas vezes não são recuperados com a engenharia reversa tradicional. A informação recuperada na engenharia reversa da arquitetura está em nível de abstração de projeto arquitetural ou projeto de alto nível. A engenharia reversa tradicional representa apenas uma etapa, essencial, ao processo de recuperação da arquitetura de software.

A fim de suportar a recuperação da arquitetura de software, diversas técnicas, métodos e ferramentas vêm sendo propostos na literatura. Com o interesse crescente em arquitetura, a recuperação da arquitetura vai ganhando seu espaço, já havendo diversos artigos publicados na área, além de workshops nas grandes conferências dedicados à

discussão do tema, como é o caso do SWARM Fórum - Fórum de Recuperação e Modelagem de Arquitetura de Software (DEURSEN, 2001), que vem ocorrendo desde 2001 nas conferências internacionais de engenharia reversa (WCRE – *Working Conference on Reverse Engineering*).

Diante do contexto apresentado, o objetivo deste capítulo é descrever o processo de recuperação de arquitetura de software e técnicas que vêm sendo utilizadas para suportar as atividades do processo.

3.2 Processo de Recuperação da Arquitetura de Software

O processo de recuperação da arquitetura de software de sistemas legados é um processo bifásico, contemplando uma fase de extração de informações do código fonte e uma fase de manipulação e análise dos modelos extraídos (MENDONÇA e KRAMER, 1996), (KAZMAN e CARRIÈRE, 1997). Um *framework* com uma visão geral do processo está representado na figura 4.

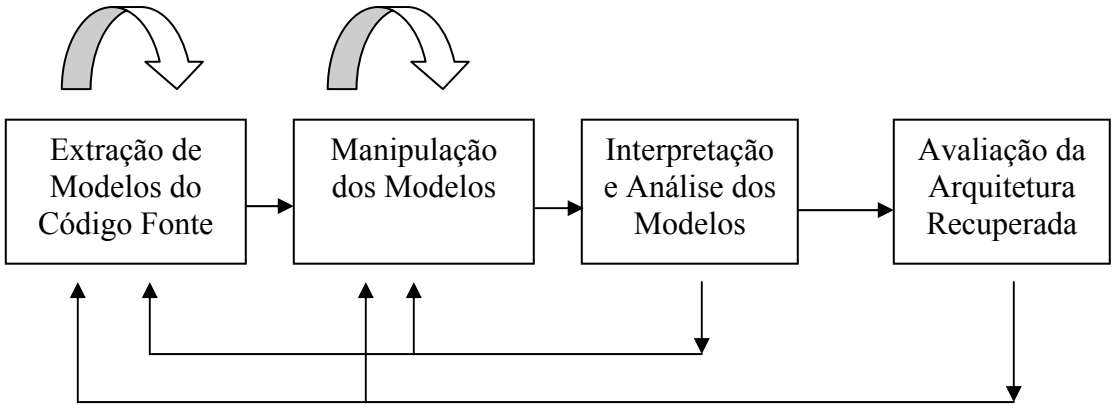

Figura 4. Framework do Processo de Recuperação de Arquitetura de Software.

O processo é iterativo e incremental. A fase de extração é caracteristicamente *bottom-up* e envolve a extração de modelos estáticos e comportamentais do código fonte. Os sistemas legados são em geral grandes e complexos, apresentando milhares de linhas de código, e, neste sentido, os modelos extraídos são conseqüentemente extensos e de difícil análise. Dessa forma, manipulações para elevar seu nível de abstração são necessárias. A cada reorganização do modelo, uma análise é realizada e esta pode identificar a necessidade de extração de informações de mais uma parte do código fonte ou a necessidade de realização de novas manipulações. Ao final, a arquitetura recuperada deve ser avaliada,

podendo ocorrer, nesta avaliação, a identificação da necessidade de re-execução do processo para algumas partes do sistema.

A recuperação da arquitetura é então cíclica, na qual, a execução de um ciclo completo de atividades exige o estabelecimento prévio do foco daquele ciclo. Conforme argumentam DING e MEDVIDOVIC (2001), estabelecendo-se o foco, reduz-se a quantidade de informação a ser trabalhada na reconstrução da arquitetura e simplifica-se o processo. A figura 5 apresenta os artefatos consumidos e gerados em cada etapa do processo de recuperação da arquitetura. Na fase de manipulação, os modelos extraídos do código são reorganizados, na busca por uma representação arquitetural do sistema. O objetivo nesta fase é que se vá, de forma incremental, encontrando agrupamentos lógicos de elementos físicos do código (*clusters*), os quais conduzem, em última instância, aos elementos arquiteturais da aplicação. Estes agrupamentos podem ser realizados de forma manual, pelo desenvolvedor, automatizada ou semi-automatizada.

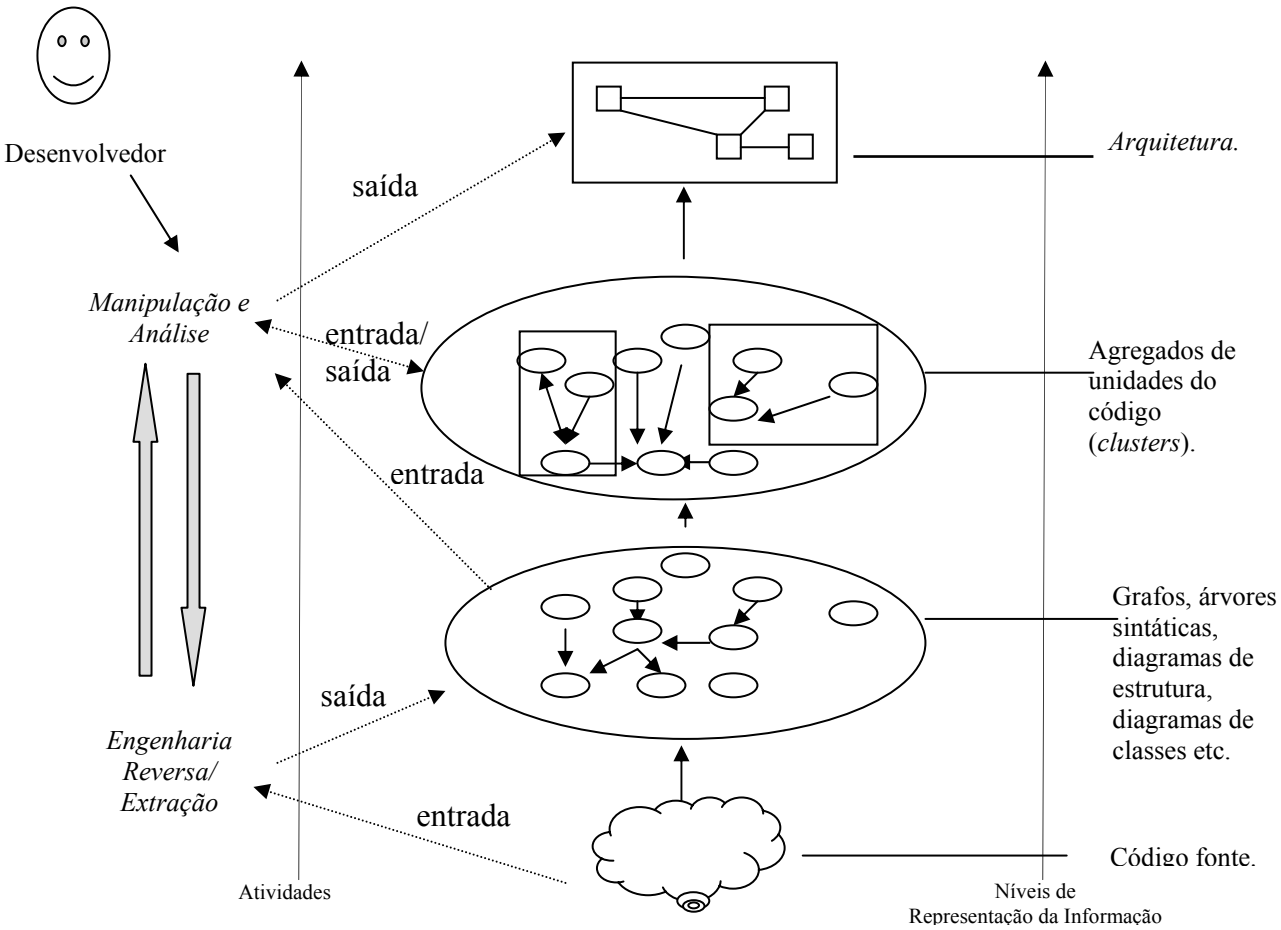

Figura 5. Uma Visão Geral do Processo de Recuperação de Arquitetura.

3.2.1 Fontes de Informação para a Recuperação da Arquitetura

O processo de recuperação da arquitetura é semi-automatizado e, conforme pode ser observado na figura 5, exige um desenvolvedor com conhecimento da aplicação para guiar o processo. Quando este conhecimento inexiste, o desenvolvedor deve levantar informações sobre o domínio da aplicação e sobre a própria aplicação junto a especialistas, desenvolvedores originais do sistema, documentação existente etc. Conforme afirmam PASHOV e RIEBISCH (2003), embora a maioria das abordagens de recuperação da arquitetura se apóie no código como a fonte de informação mais confiável e disponível sobre o sistema, um conhecimento significativo sobre o domínio do problema é requerido para facilitar a extração de informações arquiteturais úteis sobre o software. A figura 6 apresenta algumas fontes de informação que podem ser utilizadas no processo de recuperação da arquitetura de software.

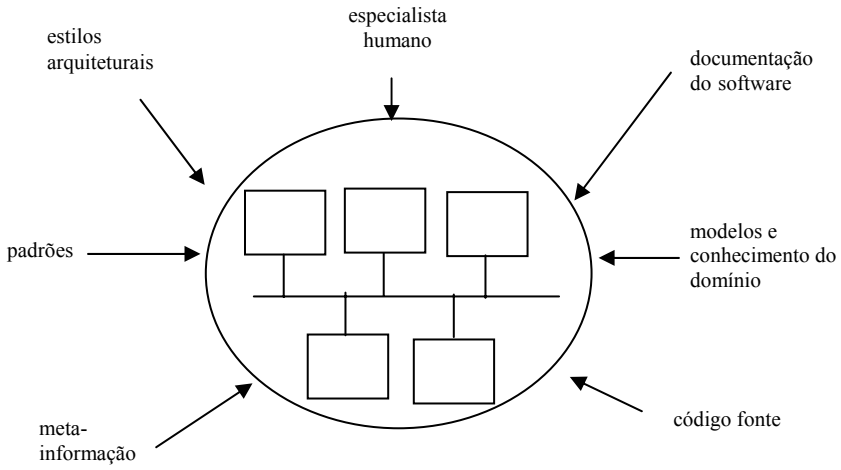

Figura 6. Fontes de Informação para a Recuperação da Arquitetura (GALL et al., 1996).

3.3 Técnicas de Apoio ao Processo

Diversas técnicas são utilizadas para suportar as atividades do processo de recuperação da arquitetura, sendo algumas delas passíveis de ser automatizadas e outras somente semi-automatizadas. Dentre as técnicas, destacam-se a engenharia reversa e os mecanismos de agrupamento (*clustering*).

3.3.1 Engenharia Reversa

Uma vez que o código é a fonte de informação mais acessível e confiável dos sistemas legados, sendo, muitas vezes, a única fonte de informação disponível, a engenharia reversa representa uma técnica essencial no processo de reconstrução da arquitetura. Analisar milhares de linhas de código diretamente sem o apoio de um ferramental que auxilie na obtenção de uma representação mais clara e de fácil manipulação do software pode se tornar exaustivo e confuso. Para apoiar esta análise, a engenharia reversa entra em cena, extraindo modelos concretos[9] do sistema em nível de projeto detalhado a partir do código.

VERONESE e NETTO (2001) destacam a existência de duas vertentes na engenharia reversa que parte do nível de implementação para o nível de projeto: uma que trata da parte estática, extraindo as informações diretamente do código (engenharia reversa estática); e outra que trata da parte dinâmica (engenharia reversa dinâmica), apresentando duas estratégias de solução: extração da dinâmica do sistema a partir da análise estática do código e extração com análise dinâmica do código. Neste trabalho, somente será tratada a engenharia reversa dinâmica com base na análise dinâmica do software, visto que esta possibilita a extração mais completa das interações entre os objetos.

3.3.1.1 Engenharia Reversa Estática

As representações de software extraídas com a engenharia reversa estática envolvem árvores sintáticas (*Abstract Syntax Trees*), grafos, diagramas de classes, diagramas de estrutura etc., e podem ser recuperadas de forma automatizada com a utilização de ferramentas de engenharia reversa estática. Inúmeras ferramentas se apresentam no mercado e em ambiente acadêmico para a extração de modelos estáticos do sistema a partir do código. A maioria delas se encontra integrada a ferramentas CASE (*Computer Aided Software Engineering*) ou a ambientes de desenvolvimento de software.

Como exemplos de ferramentas que extraem diagramas de pacotes e de classes da UML a partir do código, temos: a ferramenta CASE Rational Rose (IBM, 2003), com suporte para engenharia reversa de Java, C++ e Visual Basic; a ferramenta CASE Together (TOGETHER SOFT, 2001), com suporte para as linguagens Java, C++ e Visual

[9] Modelos concretos são modelos que podem ser obtidos diretamente do código fonte.

Basic: e a ferramenta ARES (VERONESE e NETTO, 2001), com suporte para a linguagem Java, integrada ao ambiente Odyssey (WERNER et al., 2004).

Convém ressaltar, que na engenharia reversa de sistemas OO, o modelo estrutural da UML recuperado não contém os relacionamentos de agregação e composição entre classes. Estes tipos de relacionamento não são mapeados diretamente para nenhuma estrutura de código fonte OO e a sua recuperação recai no problema de associação de conceitos, bastante explorado na literatura de engenharia reversa (*Concept Assignment Problem*) (BIGGERSTAFF *et al*., 1994). A detecção destes tipos de relacionamento teria que ser feita analisando-se os algoritmos que freqüentemente são utilizados para a sua implementação. Porém, estes mesmos algoritmos, em outras aplicações, poderiam estar sendo empregados para fins diferentes. Daí o problema de nem sempre se conseguir resgatar, com uma análise do código, o conceito original vislumbrado pelo projetista.

Por fim, observamos que as ferramentas de engenharia reversa estática necessitam dos arquivos do código fonte da aplicação para a extração dos modelos. Em função disso, elas não são capazes de analisar sistemas legados para os quais somente o executável se apresente disponível.

3.3.1.2 Engenharia Reversa Dinâmica

O objetivo da engenharia reversa dinâmica, por sua vez, é extrair modelos que reflitam o comportamento do sistema em tempo de execução. Estes modelos são constituídos dos rastros aos eventos gerados pelo programa em execução (*traces*) e podem ser representados, considerando sistemas orientados a objetos, através de diagramas de interação da UML.

Cada rastro, no caso de sistemas OO, consiste de: uma origem, constando do objeto, seu tipo e o método emissor da mensagem; e um destino, constando do objeto, seu tipo e o método receptor da mensagem. A importância dos modelos dinâmicos na reconstrução da arquitetura está na recuperação de informações que ficam omitidas dos modelos estáticos (KAZMAN et al., 1999b). No caso dos sistemas orientados a objetos, por exemplo, os mecanismos de polimorfismo e "*late binding*" (vinculação tardia de operações à sua implementação) levam à identificação do objeto referenciado e método chamado somente em tempo de execução do sistema. Além deste, outros mecanismos para sistemas distribuídos, como chamadas remotas a procedimentos (RMI – *Remote Method Invocation*)

em Java e camadas de mediação (*middleware*), em tecnologias como CORBA, encapsulam conexões a componentes remotos, tornando difícil a remontagem da arquitetura sem uma análise da dinâmica do software (KAZMAN et al., 1999b).

Os rastros de execução são recuperados de forma automática, através do uso de técnicas e ferramentas de monitoramento de programas. Técnicas que podem ser empregadas para a coleta dos rastros de execução incluem: aspectos (AOSD STEERING COMMITTEE, 2004), instrumentação do código, depuração e registro de tarefas (*logging*). Como categorias de ferramentas que aplicam estas técnicas, temos os *profilers* (e.g. EJP - *Extensible Java Profiler* - (VAUCLAIR, 2002) e Jinsight (IBM RESEARCH, 2001)), que visam a análise do perfil do programa, considerando desempenho e consumo de recursos, e ferramentas de análise de cobertura de código. Com base nas informações extraídas por ferramentas de monitoramento do código, modelos comportamentais do software, como diagramas de seqüência da UML, podem ser reconstruídos, completando-se um ciclo de engenharia reversa dinâmica.

Os modelos comportamentais permitem a compreensão de visões arquiteturais dinâmicas, como as visões de processo e de cenário do modelo de visões 4+1(seção 2.2.1). JERDING (1997) afirma que os modelos dinâmicos são essenciais para a compreensão da arquitetura:

*"*Compreender a arquitetura de um programa requer a determinação de ambos: os componentes maiores nos quais o sistema é quebrado e os meios através dos quais os componentes interagem para realizar os objetivos do programa".

Porém, um problema que se apresenta aos modelos comportamentais é o volume de eventos extraídos durante as execuções do software. Os modelos tendem a ser muito extensos, dificultando a sua manipulação e análise. A fim de contribuir com a solução deste problema, trabalhos de análise dinâmica de sistemas buscam detectar padrões de interação nos rastros de eventos.

Os padrões de interação são seqüências de mensagens que ocorrem repetidamente nos rastros. TAKAHASHI e ISHIOKA (2000) e JERDING et al. (1997) afirmam que a detecção de padrões de interação ajuda a elevar o nível de abstração dos modelos comportamentais, identificando funcionalidades que são utilizadas repetidamente pela aplicação. Os autores identificaram, em seus estudos de caso, com sistemas em diferentes

domínios, que mais de 80% dos eventos coletados compõem padrões de interação. Concluíram, portanto, que a compreensão destes padrões auxilia muito a compreensão do comportamento da aplicação. TAKAHASHI e ISHIOKA (2000) e JERDING et al. (1997) apresentam algoritmos de detecção de padrões de interação, integrados às suas ferramentas de coleta, análise e visualização de rastros.

TAKAHASHI e ISHIOKA (2000) utilizam algoritmos baseados em técnicas de compressão de dados para a detecção dos padrões de interação. JERDING et al. (1997) identificam os padrões de interação, representando o *trace* na forma de árvore e mapeando os métodos para nós da árvore e as chamadas de método para os ramos ou ligações entre os nós. Eles detectam galhos idênticos nestas árvores, ou seja, trechos de chamadas de métodos replicados. Para simplificar os modelos comportamentais, os autores computam um grafo a partir da árvore, agrupando, neste grafo, os galhos idênticos encontrados.

A ferramenta ISVIS de JERDING et al. (1997) visa suportar a compreensão do comportamento de programas, trabalhando sobre a visualização gráfica e navegação dos rastros coletados. A ferramenta oferece, além da detecção de padrões de interação, recursos interativos para a redução dos modelos, permitindo ao usuário filtrar os eventos e as classes a serem visualizados e focar em apenas uma parte do rastro de interesse. O usuário visualiza todo o *trace* em uma espécie de mural de informação global e tem a opção de focar e visualizar, em uma área expandida da tela, apenas a parte do rastro que interessa.

Nas ferramentas de representação gráfica de rastros de execução, os padrões de interação, de uma forma geral, são agrupados em uma mensagem única de mais alto nível, apoiando, juntamente com os mecanismos de manipulação e filtro dos modelos, a simplificação dos rastros. Cada mensagem de alto nível está associada a uma funcionalidade da aplicação.

3.3.2 Agrupamentos de Elementos

Os *clusters* representam agrupamentos lógicos de unidades físicas de programas (como processos, arquivos, pacotes, classes, funções, variáveis etc), representando módulos ou subsistemas da aplicação. A técnica de agrupamento de elementos (*clustering*) é essencial ao processo de recuperação de arquitetura, pois é através dos agrupamentos que se formam as abstrações no modelo e se chega a uma representação do software em nível arquitetural.

Mecanismos de agrupamentos de elementos vêm sendo empregados há anos pela comunidade de engenharia reversa para a recuperação de estruturas modulares de sistemas legados. O problema é complexo, envolvendo o particionamento do sistema em conjuntos disjuntos de entidades. Modelos estáticos de um software são na realidade grafos completamente conectados, e encontrar partições significativas sobre estes grafos não é uma tarefa simples.

Em (ANQUETIL e LETHBRIDGE, 2003), é apresentado um estudo comparativo de algoritmos de *clustering* para a remodularização de software. Os autores expõem a problemática envolvida e a abrangência do tema, apontando questões que devem ser consideradas para a adoção de um mecanismo de agrupamento adequado, como: a seleção de métricas de acoplamento e de um algoritmo de *clustering* adequados ao problema. As métricas de acoplamento representam um papel fundamental na técnica de *clustering*. É através destas que um algoritmo de agrupamento pode decidir em que grupos as entidades do software devem ficar. ANQUETIL e LETHBRIDGE (2003) apresentam dois tipos de métricas de acoplamento: métricas baseadas na ligação direta entre as entidades do código fonte e métricas baseadas nas similaridades entre elas. Ligação direta é determinada pelas dependências entre duas entidades e similaridade pelas características em comum que estas possuem, sendo que características, neste caso, podem envolver: acesso a variáveis, chamada de procedimento, nomenclaturas utilizadas etc.

Com relação aos algoritmos, diversas propostas se apresentam na literatura. A seleção de um algoritmo de agrupamento adequado ao problema é importante, pois a qualidade da modularização recuperada depende não só das métricas, mas também do algoritmo de agrupamento selecionado. Os algoritmos propostos na área de engenharia reversa prevêem a extração automática dos módulos de uma aplicação. Porém, mecanismos semi-automatizados de agrupamentos, nos quais o desenvolvedor interfere nas tomadas de decisão, também são possíveis.

Como exemplo de *clustering* automático, podem ser citados os mecanismos para colapso dos nós de um grafo, com base na análise das propriedades dos seus relacionamentos (DI LUCCA et al., 1994). Relacionamentos recursivos entre os nós, como o que pode ser observado entre os nós 2, 3 e 4 da figura 7 e os de dominância forte, indicam oportunidades de agrupamentos. A dominância forte e direta entre nós se dá quando um nó

A apresenta uma dependência para um nó B, sendo A o único caminho para se chegar ao nó B. Neste tipo de relacionamento, a dependência no sentido contrário não ocorre. Este exemplo de relacionamento pode ser observado entre os nós 9 e o 10, 11 e 12 da figura 7.

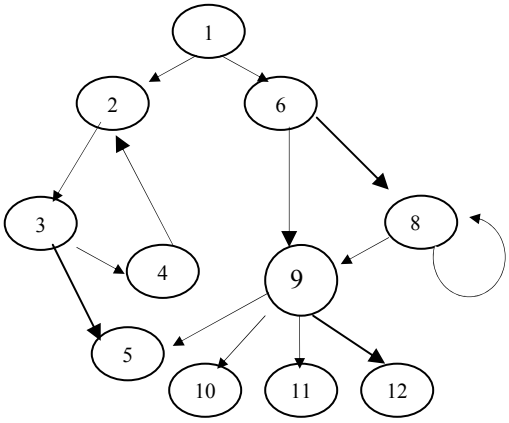

Figura 7. Grafo Direcionado de Chamadas.

Os algoritmos de *clustering* de *hill-climbing* e genéticos também geram, automaticamente, partições do sistema sobre um grafo de chamadas direcionado. MANCORIDIS et al. (1998) aplicam estes algoritmos, juntamente com métricas de inter-conectividade (acoplamento entre módulos) e de intra-conectividade (coesão nos módulos), para resgatar uma estrutura modular de sistemas legados. Os algoritmos analisam o sistema, buscando minimizar a diferença entre a média de inter-conectividade e intra-conectividade nos particionamentos gerados. O objetivo é buscar um particionamento com o valor mais alto possível de diferença, resultando em módulos altamente coesos e fracamente acoplados.

Conforme afirmam ANQUETIL e LETHBRIDGE (2003), os algoritmos de *clustering* propostos na engenharia reversa visam a recuperação de uma estrutura modular possível para o software, não apresentando o compromisso de descobrir a arquitetura da aplicação. Estes algoritmos impõem uma estrutura sobre o conjunto de entidades do software, não se preocupando em resgatar decisões passadas de projeto ou inferir conceitos a partir do código. Portanto, na área de recuperação de arquitetura, mecanismos automatizados de agrupamento não são adequados, sendo priorizados, neste caso,

mecanismos semi-automatizados, os quais serão abordados nas abordagens de recuperação de arquitetura apresentadas no capítulo 4.

3.3.2.1 Utilização de Métricas para o Agrupamento de Classes

Métricas são necessárias nos algoritmos de *clustering* para apoiar a tomada de decisão sobre a distribuição das entidades do software em módulos. Os algoritmos de *clustering* trabalham com métricas de acoplamento e de similaridade entre as entidades do software. Neste trabalho, são abordados sistemas orientados a objetos (OO) e, portanto, são investigadas métricas que se aplicam a este tipo de sistema.

O acoplamento estático entre classes, na visão de CHIDAMBER e KEMERER (1994), ocorre quando objetos de uma classe atuam sobre objetos de outra classe. Por atuar, entenda-se que métodos de uma classe referenciam métodos ou variáveis de instância da outra classe. Esta é a definição dos autores para a métrica CBO (acoplamento entre objetos de classes), a qual é medida verificando-se o número de classes às quais uma classe X está acoplada no sistema. LORENZ e KIDD (1994), por sua vez, alegam que as conexões entre classes são formas de acoplamento e que se pode medir a quantidade de acoplamentos entre classes por meio do número de conexões entre elas. Embora CHIDAMBER e KEMERER (1994) estabeleçam, assim como LORENZ e KIDD (1994), a necessidade de verificar as conexões entre classes para medir o acoplamento, eles não contabilizam o número de acessos entre as classes nesta medição, apenas contabilizam o relacionamento estabelecido entre as classes. Neste trabalho, concordamos com a visão de LORENZ e KIDD (1994), entendendo que a contagem do número de conexões entre classes é importante para medir o seu acoplamento.

Porém, a contagem de conexões estaticamente pode sofrer alguns problemas em função do polimorfismo em sistemas OO. Métodos polimórficos não demonstram, estaticamente, o tipo do objeto destino de uma chamada. A utilização de métricas dinâmicas ajuda a solucionar este problema.

Contribuindo com a discussão sobre métricas de acoplamento, HITZ e MONTAZERI (1996) alegam que não se pode medir todo o acoplamento gerado por conexões entre classes da mesma forma, uma vez que existem atributos do software que influenciam na medida de acoplamento. Como exemplos de atributos, temos: as estruturas iterativas, ou seja, chamadas de métodos em *loops*; o número de parâmetros nas chamadas

de métodos; o tipo de relacionamento entre as classes (agregação, herança etc.); a diferenciação do peso entre conexões por acesso a variáveis de instância e por chamadas de métodos; a diferenciação entre os acessos a variáveis de instância de classes estrangeiras e os acessos a variáveis de instância de superclasses, dentre outras. Atributos que influenciam na medição do acoplamento entre classes de software OO pretendem ser investigadas neste trabalho.

Um outro aspecto a ser considerado, no que tange às métricas de acoplamento de software OO, é a diferenciação que vem sendo feita na literatura entre métricas estáticas e dinâmicas. Em (CHO et al., 1998), os autores consideram que as métricas estáticas podem ser medidas em nível de classe e as métricas dinâmicas somente em nível de objeto. Em (EMAM, 2001), uma definição mais precisa para métricas dinâmicas é apresentada, onde o autor afirma que métricas estáticas podem ser computadas pela análise do código fonte ou do projeto, ao passo que métricas dinâmicas requerem a execução do programa para serem coletadas.

Métricas de acoplamento dinâmicas levam em conta o volume de mensagens trocadas entre os objetos das classes durante a execução do software. Devem ser extraídas, portanto, dos rastros de eventos coletados do sistema. Algumas métricas dinâmicas para software OO vêm sendo propostas na literatura (YACOUB et al., 1999) (CHO et al., 1998) (ARISHOLM, 2002).

Neste trabalho, conforme será explorado na seção 5.2.5.1, a proposta é combinar valores de métricas de acoplamento estáticas e dinâmicas para apoiar a reconstrução da arquitetura. Acreditamos que estas métricas se complementem em uma medida mais efetiva do acoplamento entre classes. Além disso, as métricas dinâmicas para sistemas OO foram menos experimentadas até o momento do que as métricas estáticas. A sua utilização, nesta proposta, pretende oferecer contribuições na avaliação da aplicabilidade prática deste tipo de métrica para a medição do acoplamento entre classes em sistemas OO.

Como observação final em relação à utilização de métricas, convém ressaltar que o acoplamento entre classes e entre objetos deve sempre ser analisado nos dois sentidos: da classe/objeto A para o B e vice-versa.

3.3.3 Recorte de Programas e Mapeamento de Padrões

Recorte de programas (p*rogram slicing)* é uma técnica automatizada utilizada para determinar que parte do código em um programa é relevante para uma computação particular (BALL, 1994). A técnica de *program slicing* foi proposta por WEISER (1984) com o intuito de auxiliar a depuração de sistemas complexos e apoiar a execução paralela. WEISER (1984) percebeu que programas grandes e complexos poderiam ser decompostos em pedaços ou fatias (*slices*), onde a determinação das fatias levaria em conta os componentes do programa que implementavam uma funcionalidade particular, realizavam um cálculo, acessavam uma variável etc. A técnica vem sendo muito utilizada para suportar a compreensão de programas, uma vez que reduz a quantidade de código a ser analisada em cada etapa do processo.

No processo de recuperação da arquitetura, a técnica de recorte de programas visa suportar o mapeamento de abstrações arquiteturais para o código, delimitando a região do programa que implementa um componente arquitetural.

Em (SYSTÄ, 1999), a técnica de recorte de programas é utilizada para recortar modelos estáticos do software. Neste trabalho, os modelos dinâmicos são utilizados para promover um recorte do modelo estático, permanecendo, neste último, apenas os objetos que aparecem nos modelos dinâmicos extraídos. Dessa forma, apenas os objetos que implementam determinados comportamentos do sistema permanecem no modelo, reduzindo a sua complexidade e facilitando a sua análise.

Por fim, a técnica de mapeamento de padrões para o código (*pattern matching)* também é aplicada na recuperação da arquitetura. A técnica de *pattern matching* visa analisar o código para descobrir as estruturas que implementam um determinado padrão, recortando e agrupando estas estruturas para a representação do padrão na arquitetura. Os padrões detectados incluem: os estilos e padrões arquiteturais mencionados na seção 2.2 e os padrões específicos da aplicação ou do domínio.

3.3.4 Técnicas de Outras Áreas

Além das técnicas mencionadas, técnicas de outras áreas também vêm sendo empregadas para a reconstrução da arquitetura.

Técnicas de *Data Mining*[10] (mineração de dados), da área de banco de dados, por exemplo, vêm sendo utilizadas em trabalhos como o de SARTIPI et al. (1999) para apoiar a reconstrução da arquitetura (veja seção 4.3.3).

Da área de matemática, a técnica de análise de conceito formal (f*ormal concept analysis* ou m*athematical concept analysis*) também é utilizada. Ela vem sendo empregada em trabalhos de engenharia reversa, assim como a técnica de *clustering*, para apoiar a recuperação dos módulos de um sistema. A técnica é baseada em um modelo teórico matemático para a identificação de hierarquias de conceitos. O modelo oferece uma base matemática para a identificação dos conceitos de um sistema, definindo conceitos como unidades que consistem de duas partes: uma extensão, constando dos objetos ou entidades de um conceito e uma intenção, cobrindo os atributos ou propriedades do conceito. Os atributos são válidos para todos os objetos considerados em um conceito. Portanto, a técnica provê um mecanismo para a identificação de grupos de objetos em um sistema que compartilham atributos em comum (BURMEISTER, 1998). Para a engenharia reversa, os grupos de objeto são entidades do software (como procedimentos e funções) e os atributos, elementos acessados pelos objetos (como variáveis globais).

O auxílio da técnica na recuperação de estruturas modulares de um sistema está na identificação das entidades que compartilham elementos e que, devido a este compartilhamento, devem ser agrupadas em módulos. Um exemplo de compartilhamento de atributos entre objetos está ilustrado na tabela 4.

	A1	A2	A3	A4	A5	A6	A7	A8	A9
O1	X	X							
O2			X	X	X				
O3			X	X		X	X	X	
O4			X	X	X	X	X	X	X

Tabela 4. Matriz Descrevendo uma Relação de Contexto (BOJIC E VELASEVIC, 2000).

[10] *Data Mining* (mineração de dados) refere-se a uma coleção de algoritmos para a descoberta de relações interessantes e não-triviais entre os dados em um grande banco de dados (FAYYAD, 1996).

Sobre a tabela 4, a hierarquia de conceitos apresentada na figura 8 é computada. Cada conceito representa um agrupamento dos objetos e dos atributos que eles compartilham. Um conceito possui sub-conceitos, que representam sub-conjuntos dos objetos do conceito pai.

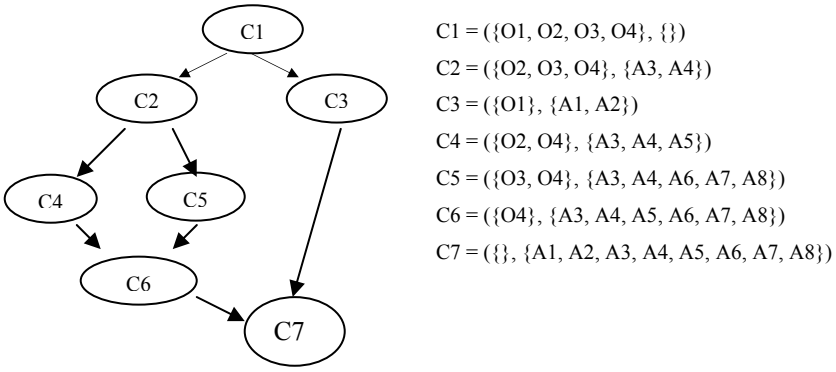

Figura 8. Um Exemplo de Trama Conceitual (BOJIC E VELASEVIC, 2000).

Considerando que no exemplo da figura 8 os objetos sejam rotinas do programa e os atributos as variáveis globais acessadas, os conceitos do grafo representam conjuntos de rotinas que acessam variáveis em comum. Destes conceitos, os módulos do sistema são extraídos. Um exemplo de aplicação da técnica para a recuperação de arquitetura de software pode ser observado na seção 4.4.1.

Embora, a técnica de análise de conceito formal venha sendo empregada em trabalhos de engenharia reversa e de recuperação de arquitetura, ela gera uma hierarquia de conceitos com muitos níveis de aninhamento e redundância. No grafo gerado, um mesmo objeto (como o O2 da tabela 4) pode aparecer em diversos conceitos (C1, C2 e C4), o que dificulta a tomada de decisão sobre o mapeamento de conceitos para módulos do software. Apesar disso, a técnica detecta unidades compartilhadas entre entidades de um software e, por este motivo, ela pretende ser explorada neste trabalho.

3.4 Considerações Finais

Para a recuperação da arquitetura de software, uma combinação de técnicas deve ser empregada. As métricas de acoplamento, por exemplo, são essenciais aos algoritmos de agrupamentos de elementos. Os agrupamentos de elementos, por sua vez, complementam a técnica de *pattern matching,* agrupando as entidades do código fonte que representam o

componente ou conector de um padrão. Através da detecção de padrões, as abstrações arquiteturais são mapeadas para os modelos extraídos do código, conduzindo a uma representação arquitetural do sistema.

As técnicas apresentadas neste capítulo são empregadas nas abordagens de recuperação da arquitetura de software apresentadas no próximo capítulo. As abordagens de recuperação seguem o *framework* do processo discutido na seção 3.2, instanciado-o e estendendo-o de acordo com as suas necessidades. Para cada atividade do processo e abordagem de recuperação, uma configuração e combinação particular das técnicas se mostra mais adequada.

4 Abordagens para a Recuperação da Arquitetura de Software

4.1 Introdução

Diversas abordagens vêm sendo propostas na literatura para a recuperação da arquitetura de software. Cada uma delas apresenta objetivos específicos, mas todas se baseiam, de alguma forma, no processo e nas técnicas apresentados no capítulo 3. Dentre os objetivos para a reconstrução da arquitetura, destacam-se:

- Avaliação da conformidade da arquitetura implementada[11] com a arquitetura projetada (ou documentada) da aplicação;
- Geração de uma documentação do software atualizada e adaptável às diferentes necessidades de manutenção do sistema;
- Compreensão de programas;
- Suporte à reengenharia dos sistemas, oferecendo a base para a análise do software e sua reestruturação;
- Extração de componentes de código legado;
- Migração de sistemas legados para uma linha de produtos.

Em (O'BRIEN et al., 2002) é apresentada uma classificação para as abordagens e ferramentas de recuperação de arquitetura. Ela leva em conta o nível de automatização atingido no processo e as técnicas empregadas. A classificação proposta inclui as seguintes categorias: reconstrução manual da arquitetura (LAINE, 2001); reconstrução manual com suporte ferramental (WONG, 1994) (KLOCWORK, 2002); consultas (*queries*) para o mapeamento de padrões, levando à geração automática de agregados no sistema (HARRIS et al., 1995) (HARRIS et al., 1997a) (KAZMAN e CARRIÈRE, 1997) (GUO et al., 1999); reconstrução utilizando outras técnicas, como *data mining* (SARTIPI et al., 2000) (MENDONÇA e KRAMER, 2001) (KRIKHAAR, 1999), e utilizando ADLs (EIXELSBERGER et al., 1998).

[11] A arquitetura implementada (ou concreta) corresponde à arquitetura real da aplicação, derivada do modelo concreto extraído do código fonte. A arquitetura projetada (ou idealizada) corresponde à arquitetura projetada para a aplicação durante o processo de engenharia direta (engenharia de software na direção análise-projeto-codificação), a qual segue, em geral, um ou mais estilos arquiteturais.

Neste trabalho, é proposta uma classificação para as abordagens de recuperação de arquitetura baseada na fonte de informação priorizada no processo. A tabela 5 apresenta a taxonomia proposta.

Recuperação Baseada em Estilos e Padrões Arquiteturais	Recuperação Baseada no Conhecimento do Domínio e da Aplicação	Recuperação baseada nos Requisitos Funcionais da Aplicação
Reconhecimento de estilos arquiteturais (HARRIS et al., 1997a).	Consultas para a identificação de padrões do domínio (KAZMAN e CARRIÈRE, 1997).	Abordagem direcionada por casos de uso (BOJIC e VELASEVIC, 2000).
Recuperação suportada por padrões (GALL e PINZGER, 2002).	Agrupamentos de elementos com base nos conceitos do domínio (RIVA e RODRIGUEZ, 2002).	Abordagem direcionada pelos requisitos de evolução do software (DING e MEDVIDOVIC, 2001)
	Recuperação baseada na combinação de grafos (SARTIPI et al., 2000)	

Tabela 5. Taxonomia Proposta para as Abordagens de Recuperação de Arquitetura.

Neste capítulo, são descritas as abordagens apresentadas na tabela 5. A descrição segue o *template* utilizado para a análise das abordagens, englobando: uma descrição geral da abordagem; o processo e as técnicas empregadas para a recuperação da arquitetura; a generalidade da abordagem, tanto no que diz respeito à linguagem de programação, quanto no que tange ao domínio dos sistemas legados; o método utilizado para a avaliação dos resultados; as contribuições dadas pela abordagem à área e os pontos ainda não cobertos pela proposta.

4.2 Recuperação Baseada em Estilos e Padrões Arquiteturais

Os estilos e padrões arquiteturais apresentados na seção 2.3.1 foram definidos observando-se a organização de sistemas existentes. Através de estudos dos sistemas legados, observou-se que os mesmos faziam uso de estruturas semelhantes. Destas estruturas derivaram-se diversos estilos e padrões catalogados, o que justifica a sua utilização como fonte de informação para a reconstrução da arquitetura. Nesta seção, são apresentadas abordagens que fazem uso, prioritariamente, da semântica dos estilos e padrões arquiteturais para a recuperação da arquitetura.

4.2.1 Reconhecimento de Estilos Arquiteturais

Em (HARRIS et al., 1997a), a recuperação da arquitetura se apóia na identificação dos componentes e conectores de estilos arquiteturais no código, através do reconhecimento de padrões de codificação (idiomas) no programa.

- ***Processo e Técnicas Aplicadas:***

Os autores propõem um *framework* para a recuperação da arquitetura, contemplando três componentes: um módulo de extração de informações do código fonte, que recupera uma árvore sintática (AST – *Abstract Syntax Tree*); um módulo para a representação da arquitetura (implementada e idealizada); e um módulo de reconhecimento de estilos arquiteturais no código, contando com uma biblioteca de reconhecedores (programas para a identificação de componentes e conectores) e uma biblioteca de estilos. Uma visão geral dos artefatos e atividades do *framework* é ilustrada na figura 9.

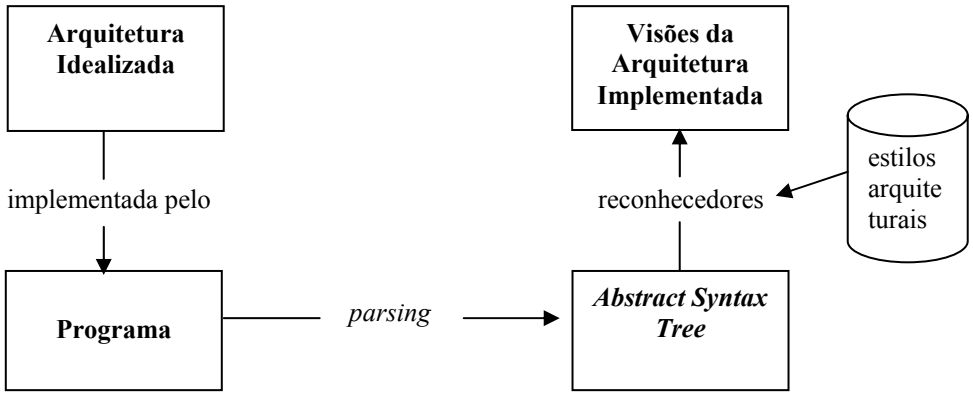

Figura 9. *Framework* de Recuperação de Arquitetura. Adaptado de HARRIS et al. (1997a).

Os reconhecedores percorrem a árvore sintática, buscando estruturas do programa que indiquem a presença de um componente ou conector de estilo arquitetural. A cada trecho de código onde é identificada uma instância de componente, os reconhecedores realizam um recorte do programa, delimitando a região que implementa o componente. Na região recortada, é feita uma análise de valores de variáveis e de parâmetros para se descobrir o destino da conexão. Um exemplo de reconhecedor para o estilo *Task Spawning* (tarefas encadeadas, que disparam outras tarefas) é apresentado na figura 10.

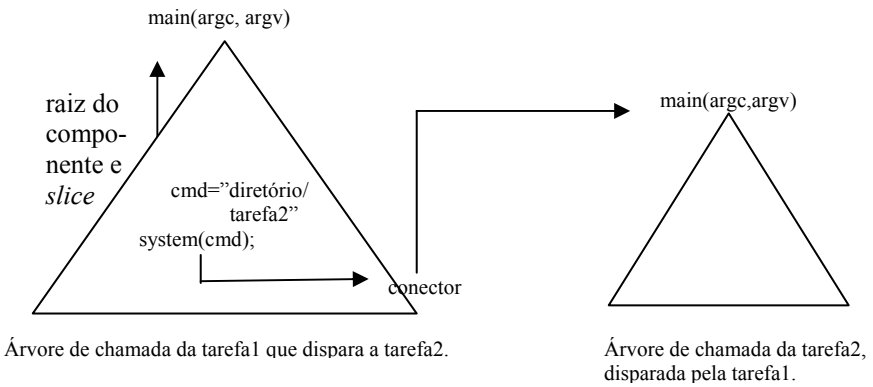

Figura 10. Reconhecedor do Estilo *Task Spawning* (HARRIS et al., 1997a).

No exemplo da figura 10, o reconhecedor de tarefas encadeadas descobre através do comando "system" o estilo arquitetural *Task Spawning*. A partir deste comando, o reconhecedor realiza um recorte do programa desde a raiz da tarefa até o final de sua árvore, delimitando o componente. Sobre este recorte, o reconhecedor analisa valores de variáveis e de parâmetros para descobrir a tarefa que é disparada. O parâmetro/variável "cmd" indica que o componente destino da conexão é a tarefa 2. O reconhecedor delimita também a região da tarefa 2, realizando o mapeamento do estilo arquitetural *Task Spawning* para o código.

- ***Generalidade:***

É possível, além dos estilos específicos para C/Unix, como o *Task Spawning*, a identificação de alguns estilos arquiteturais genéricos, como orientação a objetos. O reconhecimento de objetos, ou tipos abstratos de dados, é realizado de forma semi-automática, com a participação de um desenvolvedor. Este tipo de reconhecimento não se baseia exclusivamente nos padrões de codificação, embora a utilização de idiomas seja o ponto chave da abordagem.

Além de alguma generalidade em nível dos estilos, os reconhecedores são genéricos na sua estrutura, na sua lógica, mas específicos para o ambiente C/Unix nos padrões de codificação que são capazes de detectar.

- ***Avaliação dos Resultados:***

Utilizam métricas de cobertura de código para indicar o número de linhas de código e rotinas cobertas pelos estilos identificados. Ainda que as métricas não sejam as mais

adequadas e necessitem alguma calibração, elas fornecem um indicativo para os analistas do percentual do sistema que pode ser redocumentado aplicando o método proposto.

- ***Contribuições e Pontos em Aberto:***

O método e ferramental propostos permitem a automatização do reconhecimento de alguns estilos arquiteturais no código, oferecendo considerável contribuição à área de recuperação de arquitetura de software. Alguns pontos permanecem em aberto no trabalho e pretendem ainda ser explorados pelos autores, como: a recuperação da arquitetura de sistemas que utilizam COTS (componentes de prateleira), cujo código fonte não se encontra disponível; a modelagem de requisitos funcionais, permitindo o mapeamento das funcionalidades da aplicação para os elementos arquiteturais; a extensão do *framework* para cobrir outras linguagens de programação.

4.2.2 Recuperação Suportada por Padrões

Em (GALL e PINZGER, 2002), o processo de recuperação da arquitetura também é voltado à identificação de estilos e padrões arquiteturais no código fonte. A abordagem se baseia na identificação de padrões de codificação ou idiomas, assim como a anterior. Porém, o que a abordagem acrescenta de novo em relação à anterior é a identificação dos "*hot spots*" (elementos-chave) de um padrão e a possibilidade do refinamento sucessivo na especificação de um padrão. *Hot-spots* são os elementos que sempre devem aparecer no código quando o padrão é utilizado, independente das suas variações de implementação (por exemplo, implementação de um mesmo padrão em diferentes linguagens de programação).

Como exemplo de padrão de codificação, pode-se citar o uso de estruturas de soquete (*socket*) em C++ e Java, indicando a implementação de uma arquitetura cliente-servidor no sistema. Independente das variações na implementação dos *sockets*, a criação de um objeto do tipo *Socket* sempre indicará, nos dois ambientes, que uma arquitetura cliente-servidor está sendo empregada.

- ***Processo e Técnicas Aplicadas:***

A recuperação se inicia com a identificação dos padrões de codificação utilizados no sistema. Os padrões identificados são descritos através de expressões regulares em XML. A figura 11 ilustra a descrição do padrão soquete.

```
<pattern id = "C-socket">
   <match>
        <block start = "{" end="}">
           <text> = socket(</text>
           <anytext />
           <text>);</text>
         </block>
   </match>
</pattern>
```

Figura 11. Definição de Padrão para Encontrar Implementações de *Socket* em C e Java (GALL e PINZGER, 2002).

Uma ferramenta de reconhecimento de padrões, ESPaRT (KNOR'S et al., 1998), realiza a análise léxica do código e identifica os blocos de texto que implementam as expressões regulares que caracterizam um padrão. Por ser escrita em XML, a especificação de um padrão pode ser estruturada, permitindo o refinamento da descrição. Por exemplo, para se identificar um Cliente, numa arquitetura cliente-servidor implementada em C, além de se identificar a criação de um objeto do tipo *Socket* em um bloco de programa, deve-se verificar a existência de uma chamada a *connect()*neste mesmo bloco. Dessa forma, para identificar um Cliente, primeiro especifica-se o padrão *socket* e depois refina-se a especificação. A figura 12 ilustra o refinamento sucessivo da especificação de padrões.

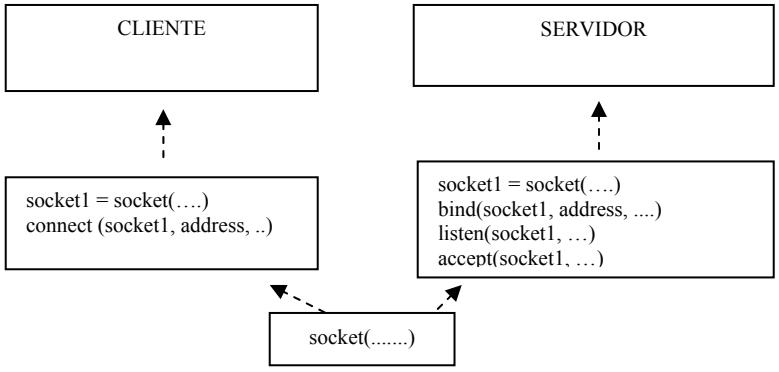

Figura 12. Refinamento de Padrões para a Identificação de Elementos Arquiteturais.

Após a identificação dos padrões, ferramentas de engenharia reversa são utilizadas para detalhar os elementos que compõem o padrão e decifrar as conexões. Diferentemente da abordagem anterior, a detecção de uma conexão é manual.

- ***Generalidade:***

Com relação às linguagens de programação, a inclusão de novas linguagens, diferentes de C, C++ e Java, depende da compatibilidade das mesmas com a ferramenta ESPaRT. Com relação aos padrões especificados, a cada nova linguagem de programação ou novo padrão arquitetural, novos padrões de codificação devem ser especificados em XML.

- ***Avaliação dos Resultados:***

Os autores utilizaram um estudo de caso real para a avaliação dos resultados, porém focaram em apenas uma característica arquitetural da aplicação: a comunicação de dados. Discutiram os resultados obtidos com um especialista da aplicação e obtiveram uma avaliação satisfatória.

- ***Contribuições e Pontos em Aberto:***

Os autores demonstram a viabilidade da identificação de alguns padrões arquiteturais em sistemas legados partindo da identificação de estruturas do código fonte. Propõem uma abordagem que embora seja leve em termos de computação, por ser baseada na análise léxica do código, apresenta limitações com relação aos padrões que é capaz de identificar. A abordagem carece ainda de um mecanismo automatizado de mapeamento de requisitos funcionais para os elementos arquiteturais identificados, mapeamento este necessário para suportar futuras manutenções do sistema. Por hora, o mapeamento é realizado de forma manual com o auxílio de especialistas.

4.2.3 Considerações sobre a Recuperação Baseada em Estilos e Padrões Arquiteturais

A identificação de estilos e padrões arquiteturais em sistemas legados representa um valioso caminho para a reconstrução da arquitetura. Os estilos e padrões trazem consigo uma semântica que ajuda a caracterizar as estruturas de uma aplicação. Porém, a sua identificação no código não é uma tarefa trivial. De acordo com HARRIS et al. (1997a), há uma grande distância de vocabulário entre o domínio arquitetural e o domínio das linguagens de programação, o que dificulta a identificação de componentes e conectores no código. Por exemplo, não existe um construtor “camada” ou “filtro” nas linguagens de programação. O estilo arquitetural camadas é implementado através de regras que são estabelecidas pelos projetistas e que, se espera, sejam seguidas pelos programadores.

Regras como, o padrão de nomenclatura para os componentes alocados em uma camada ou a utilização de herança entre os componentes de uma mesma camada.

Esta distância de vocabulário caracteriza um problema já conhecido na comunidade de engenharia reversa, o "Problema da Associação de Conceitos" (*Concept Assignment Problem*) (BIGGERSTAFF *et al*., 1994). Cada conceito da arquitetura pode ser implementado de várias formas no código, não havendo um mapeamento de um para um (ou mesmo um para vários, de forma precisa) do conceito arquitetural para a sua implementação no código fonte. Isto dificulta a recuperação do conceito arquitetural original a partir do programa.

Para alguns estilos e padrões arquiteturais, cuja implementação em determinadas linguagens de programação é feita, aplicando-se padrões de codificação característicos, o reconhecimento baseado em estruturas sintáticas torna-se viável. É o caso do padrão arquitetural cliente-servidor, cuja implementação em linguagens como C++ e Java é feita, geralmente, utilizando-se estruturas do tipo soquete. Para outros estilos, que envolvem principalmente aspectos relacionados ao fluxo de dados e de controle, como camadas e tubos e filtros, uma análise mais ampla da aplicação, possivelmente envolvendo uma análise comportamental, é requerida.

Um outro problema que se apresenta à recuperação dos estilos e padrões é que os sistemas, em geral, não seguem um, mas diversos padrões arquiteturais, apresentando arquiteturas híbridas. Recuperar uma representação da arquitetura que combine estes padrões não é trivial. Em (HARRIS et al., 1997b), há propostas para a combinação destas visões arquiteturais referentes aos diferentes estilos detectados.

Pode-se concluir, que esforços para a identificação de padrões arquiteturais em trabalhos de recuperação de arquitetura ainda são necessários, visando, sobretudo, linguagens de programação não muito exploradas, como Java (a maioria dos trabalhos identifica padrões de codificação em C e C++), e os benefícios obtidos com a análise dinâmica da aplicação.

4.3 Recuperação Baseada no Conhecimento do Domínio e da Aplicação

Algumas abordagens utilizam o conhecimento do domínio como fonte de informação prioritária para a reconstrução da arquitetura. Em geral, estas fazem uso de mecanismos semi-automatizados, baseados em consultas, para o reconhecimento de

padrões de domínio no código. Os padrões de domínio envolvem abstrações arquiteturais conhecidas do domínio ou da própria aplicação. Segue, nesta seção, a descrição de abordagens significativas nesta categoria.

4.3.1 Consultas para a Identificação de Padrões do Domínio

Em (KAZMAN e CARRIÈRE, 1997), é apresentado um ambiente para a recuperação da arquitetura - Dali *workbench* - que visa prover facilidades para a manipulação do modelo extraído do código. O ambiente prevê duas formas de manipulação dos modelos: direta, com interação do usuário, e indireta, através de um mecanismo de mapeamento de padrões. A recuperação da arquitetura tem como principal objetivo a verificação da conformidade entre as arquiteturas projetada e implementada da aplicação.

- ***Processo e Técnicas Aplicadas:***

Visto que a busca da conformidade com a arquitetura projetada é o maior objetivo da recuperação, os autores buscam nos documentos do sistema e do domínio as abstrações arquiteturais que devem ser mapeadas para o código. Utilizam a técnica de *pattern matching* (seção 3.3.3) para o mapeamento. Quando não existe qualquer arquitetura documentada, os desenvolvedores pesquisam os conceitos do domínio para mapeá-los para o código.

O ambiente Dali está ilustrado na figura 13. Através da análise dos módulos que compõem o ambiente, pode-se ter uma compreensão do processo proposto para a reconstrução da arquitetura. O ambiente Dali é composto por cinco módulos, a saber:

- **Extração**: incorpora extratores para diferentes visões do código (acesso a variáveis, chamadas de procedimentos etc.), possibilitando a geração de um modelo concreto mais completo e confiável do sistema.
- **Combinação:** visa a combinação dos modelos extraídos do código, gerando um modelo concreto único e inserindo-o no repositório central. Em caso de conflitos entre modelos, aplica um mecanismo de votação para a seleção dos elementos.
- **Visualização e Interação**: módulo para a visualização do modelo e manipulação direta. Utilizam a ferramenta Rigi (WONG, 1994).

- **Manipulação Externa**: módulo para a definição e mapeamento dos padrões de domínio para o modelo. Os padrões são escritos em SQL e em Perl (WALL, 1991). As consultas em SQL selecionam no repositório central os elementos do sistema que deverão ser agregados e o código em Perl atua como uma cola, agrupando os nós do grafo correspondentes aos elementos selecionados. Dessa forma, a arquitetura vai sendo remontada.
- **Análise:** módulo para análise da arquitetura reconstruída.

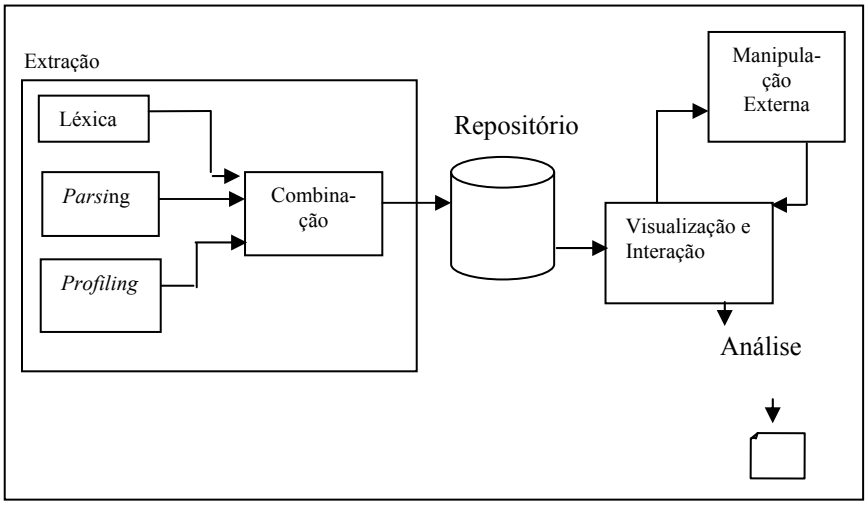

Figura 13. Dali *Workbench*: um Ambiente para Suporte à Recuperação da Arquitetura (KAZMAN e CARRIÈRE, 1997).

❖ ***Generalidade:***

O ambiente Dali visa a integração de diferentes ferramentas, que se comunicam através do acesso ao repositório central ou da troca de arquivos. Como o método de recuperação se apóia nos elementos do repositório e no grafo extraído, tem-se independência em relação à linguagem de programação. No que tange à generalidade em nível de domínio, observa-se que os padrões precisam ser reescritos para cada novo domínio e/ou aplicação.

❖ ***Avaliação dos Resultados:***

A avaliação da arquitetura recuperada é feita verificando-se os desvios que esta apresenta em relação à arquitetura projetada do sistema. Em alguns casos, obtém-se como resultado uma malha de dependências tão complexa entre os elementos arquiteturais

recuperados, que se chega à conclusão que a arquitetura implementada não apresenta qualquer afinidade com a que foi projetada inicialmente, ou que esta não reflete qualquer nível de coesão funcional nos seus componentes. A avaliação da arquitetura traz à tona necessidades de reestruturação do sistema legado.

- ***Contribuições e Pontos em Aberto:***

As maiores contribuições deste trabalho são as facilidades para a manipulação dos modelos e o ambiente Dali para suporte à recuperação da arquitetura. As consultas, juntamente com o mecanismo de geração automática de agregados, permitem a manipulação de um grupo de elementos de uma vez e a reconstrução semi-automática dos módulos. Considerando o volume de informação em um sistema legado, estas facilidades contribuem consideravelmente para a recuperação da arquitetura.

As consultas permitem ainda a aplicação de filtros no modelo, eliminando informações que não são de interesse para a reconstrução da arquitetura, como funções de bibliotecas externas. Com isso, reduz-se o volume de informações para análise.

Como contrapartida, há uma necessidade de maior exploração de visões arquiteturais dinâmicas.

4.3.2 Agrupamentos de Elementos com Base nos Conceitos do Domínio

Em (RIVA e RODRIGUEZ, 2002), o agrupamento de elementos do código também é realizado com base no conhecimento do domínio. Os modelos recuperados do código são armazenados em uma base de fatos em Prolog e os agrupamentos de elementos são realizados através da definição de regras sobre os fatos.

O modelo estático recuperado é visualizado através de um grafo na ferramenta Rigi (WONG, 1994) e os modelos dinâmicos são visualizados através de diagramas de seqüência de mensagens (semelhantes aos diagramas de seqüência da UML). A ferramenta Rigi foi estendida para suportar a visualização dos diagramas de seqüência de mensagens, permitindo a sincronização destes com o modelo estático. O ponto forte da abordagem está de fato nesta capacidade de sincronização entre a visão estática e as visões dinâmicas recuperadas: o usuário navega na visão desejada, expandindo ou agregando os nós, e as modificações sobre os componentes são refletidas nas outras visões.

❖ ***Processo e Técnicas Aplicadas**:*

RIVA e RODRIGUEZ (2002) extraem modelos estáticos e dinâmicos do código e aplicam regras para a agregação de elementos nestes modelos. Para extrair os modelos dinâmicos, executam uma seqüência de passos: instrumentam o código, selecionam cenários de uso do sistema e simulam a execução do sistema para os cenários, coletando rastros de execução. Após a extração dos modelos, mecanismos de manipulação dos mesmos são oferecidos para facilitar a sua compreensão e complementar a reconstrução da arquitetura.

❖ ***Visões Arquiteturais:***

Das abordagens estudadas, esta é uma das únicas que explora um modelo dinâmico do sistema. RIVA e RODRIGUEZ (2002) trabalham sobre visões estáticas e dinâmicas da arquitetura, apresentando um mecanismo de integração entre estas duas categorias de visões. O trabalho apresenta mecanismos interessantes para a sincronização das visões e simplificação do modelo dinâmico.

Os rastros de eventos obtidos com o monitoramento do programa são, em geral, extensos e de difícil manipulação e análise. Para a recuperação de diagramas de seqüência sobre estes rastros, é fundamental que mecanismos de redução do modelo sejam disponibilizados. Como primeira estratégia para a simplificação dos rastros de execução, os autores aplicam sobre eles as mesmas regras de agrupamentos de elementos utilizadas no modelo estático. Em complementação a esta forma de manipulação indireta dos modelos, mecanismos de manipulação direta são propostos através das abstrações horizontais e verticais.

A figura 14 ilustra os mecanismos de abstração horizontal e vertical sobre os diagramas de seqüência. A abstração horizontal permite o agrupamento dos componentes representados no topo do diagrama. Na figura 14, os componentes A e B foram agrupados, passando a formar um único componente. Este agrupamento é refletido no modelo estático do sistema. Com este agrupamento, as mensagens trocadas entre os componentes A e B são encapsuladas. A qualquer momento que o usuário deseje, ele pode expandir os componentes agrupados e voltar a visualizar as mensagens escondidas.

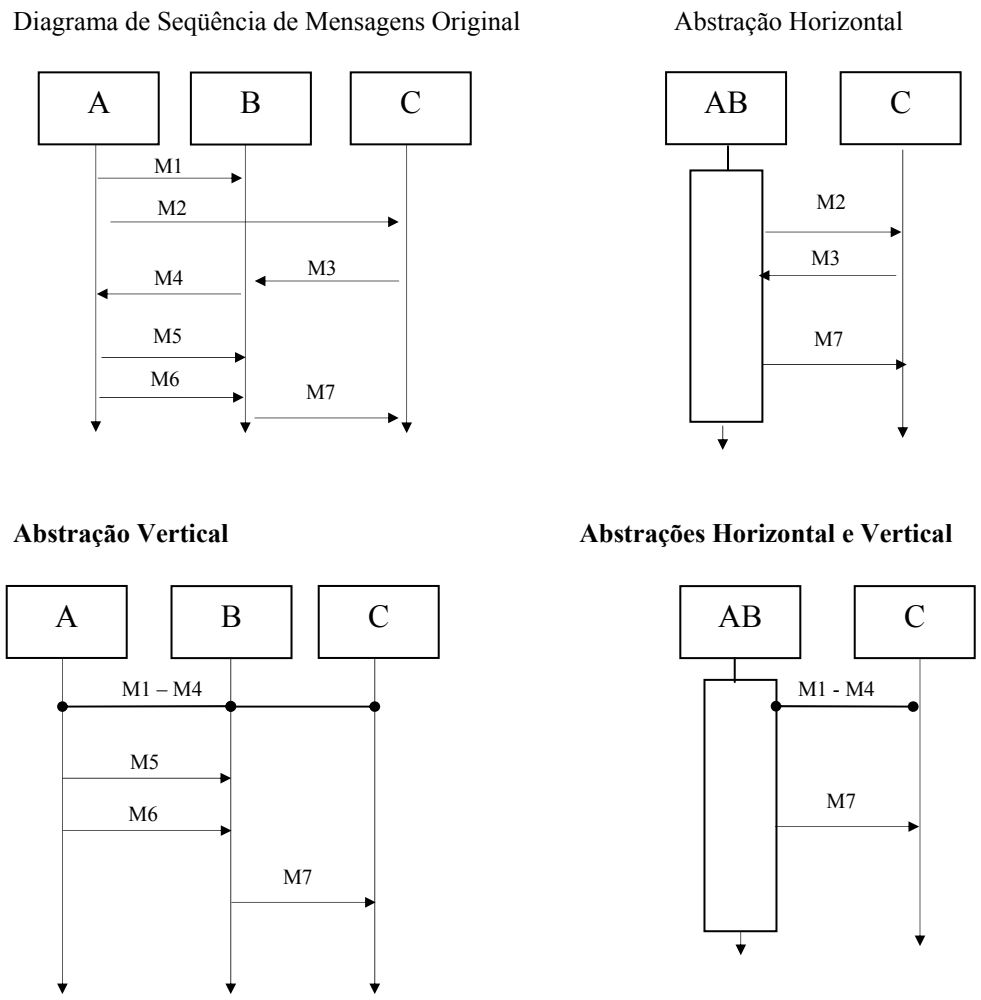

Figura 14. Abstrações Horizontais e Verticais em Diagramas de Mensagens (RIVA e RODRIGUEZ, 2002).

Por outro lado, o mecanismo de abstração vertical permite o agrupamento de uma série de mensagens contíguas (ou não) em uma mensagem de mais alto nível. No exemplo da figura 14, as mensagens M1 a M4 foram agrupadas, permitindo uma simplificação considerável do diagrama. No caso do agrupamento de mensagens não contíguas, múltiplos grupos são criados. É claro que para a aplicação de qualquer um dos mecanismos de abstração, o usuário terá que decidir em que momentos estes agrupamentos são válidos ou quando eles escondem informações relevantes para a compreensão do modelo dinâmico.

RIVA e RODRIGUEZ (2002) aplicam ainda um terceiro mecanismo de redução dos diagramas: a partição dos rastros de execução em uma série de casos de uso hierárquicos

através da inserção de marcações nos rastros de execução. As marcações indicam onde se inicia e termina um caso de uso e, com base nestas marcações, recortes no modelo dinâmico são efetuados. Dessa forma, pode ser gerado um diagrama de seqüência por caso de uso e se obter um particionamento do comportamento do sistema por funcionalidade.

- ***Generalidade**:*

O armazenamento dos modelos extraídos do código em uma base de fatos em Prolog torna o mecanismo de abstração independente de linguagem de programação.

- ***Avaliação dos Resultados:***

Aplicam a metodologia proposta sobre um estudo de caso, que envolve uma versão simplificada de um sistema real.

- ***Contribuições e Pontos em Aberto:***

A abordagem carece de um suporte à tomada de decisão no agrupamento dos componentes. O processo de abstração é conduzido manualmente por desenvolvedores. Os autores prevêem como trabalho futuro a detecção de padrões de interação nos diagramas de seqüência. Esta detecção pode representar um mecanismo que apóie a tomada de decisão acerca dos elementos do diagrama de seqüência que devem ser agregados para formar as abstrações arquiteturais.

Como maiores contribuições do trabalho estão a sincronização dos modelos e os mecanismos de simplificação dos modelos dinâmicos. Porém, ficam algumas questões em relação à sincronização: o quanto de recursos e tempo esta consome? É viável agrupar N componentes distantes no diagrama de seqüência? A técnica se aplicaria para sistemas reais?

4.3.3 Recuperação Baseada na Combinação de Grafos

A abordagem apresentada em (SARTIPI et al., 1999) (SARTIPI et al., 2000) propõe uma solução baseada na busca de correspondências (combinação) entre grafos para o problema de recuperação da arquitetura de software. Uma vez que um programa pode ser expresso na forma de um grafo, os autores propõem que um desenvolvedor expresse a arquitetura conceitual da aplicação também na forma de um grafo. Dessa maneira, o método proposto visa buscar correspondências e o casamento entre os dois grafos para obter uma arquitetura concreta para a aplicação.

- ***Processo e Técnicas Aplicadas:***

Um *framework* ilustrativo do processo se apresenta na figura 15.

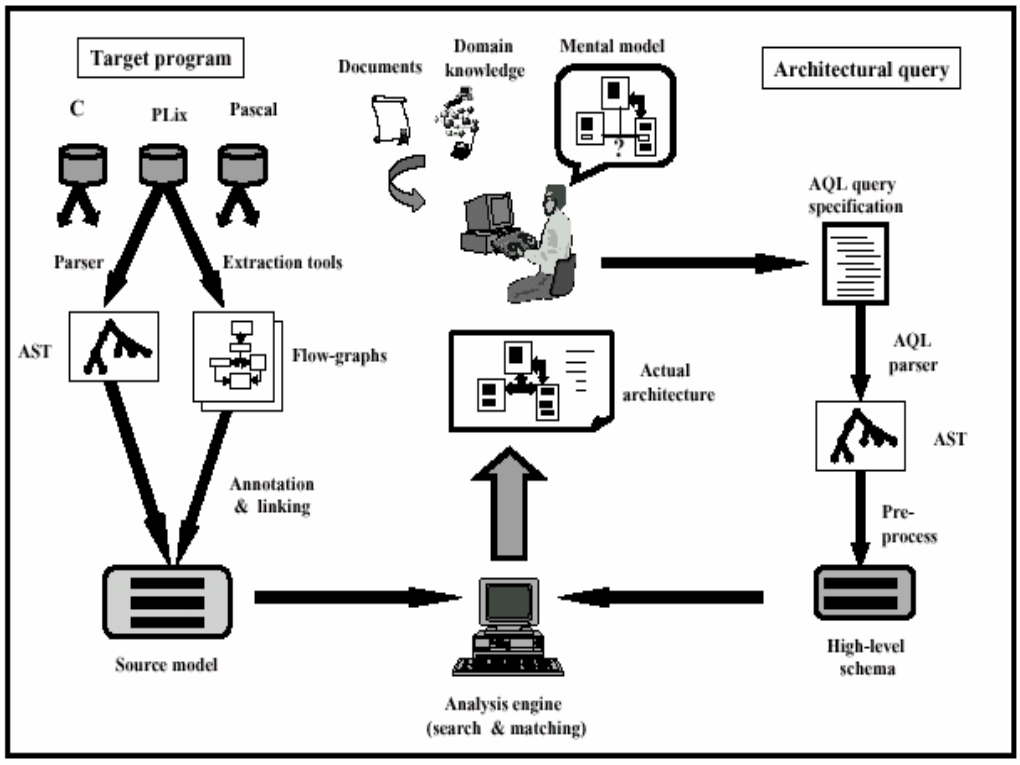

Figura 15. Um Ambiente Interativo para a Recuperação da Arquitetura baseada em Grafos (SARTIPI et al., 1999).

O usuário especifica a arquitetura conceitual ou lógica do sistema utilizando uma linguagem de consulta arquitetural (AQL – *Architectural Query Language*). A AQL é semelhante a uma ADL, exceto que ela permite deixar pontos em aberto na especificação da arquitetura (rotinas exportadas ou importadas por um módulo etc.). A máquina de análise atua justamente sobre estes pontos em aberto, buscando preenchê-los com entidades do código fonte. Este processo de preenchimento dos pontos em aberto da arquitetura conceitual representa o mecanismo de combinação de grafos ou *graph matching* adotado pelos autores.

O problema no método é o espaço de busca para a máquina de análise preencher os pontos em aberto da arquitetura. O espaço de busca, a princípio, é todo o grafo extraído do código fonte. Para sistemas grandes, o espaço de busca pode inviabilizar a aplicação do método. Para lidar com este problema, SARTIPI et al. (2000) utilizam técnicas de *data*

mining. Eles mapeiam conceitos do domínio de banco de dados para o domínio de engenharia reversa. Neste mapeamento, o conceito de transação é mapeado para rotina e o conceito de itens de dados para elementos utilizados por uma rotina, como: variáveis, tipos de dados, funções etc. O algoritmo de *data mining* utilizado descobre os itens de dados freqüentes no código, ou seja, os conjuntos de elementos em comum que são acessados por um conjunto de rotinas.

A máquina de *pattern matching* se baseia nos itens freqüentes de dados encontrados para limitar o espaço de busca.

- ***Generalidade:***

O método é genérico do ponto de vista da linguagem de programação e do domínio, bastando apenas integrar à infra-estrutura (figura 15) um *parser* para a linguagem destino quando se desejar trabalhar sobre uma nova linguagem.

- ***Avaliação dos Resultados:***

Aplicaram o método sobre sistemas de tamanho médio (em torno de 30 a 50 KLOC) e obtiveram resultados satisfatórios com relação à precisão e escalabilidade da abordagem. Porém, os próprios autores apontam a necessidade de estudos em sistemas maiores.

- ***Contribuições e Pontos em Aberto:***

O método se assemelha ao Dali de KAZMAN e CARRIÈRE (1997), no sentido de basear a recuperação da arquitetura na busca de correspondências entre a arquitetura conceitual e implementada da aplicação. Porém, introduz novas técnicas e um mecanismo diferenciado de *pattern matching*.

4.3.4 Considerações sobre a Recuperação Baseada no Conhecimento do Domínio

As abordagens apresentadas nesta seção contribuem com a área de recuperação de arquitetura, oferecendo, principalmente, mecanismos que facilitam a manipulação dos modelos extraídos do código fonte. Estes mecanismos são essenciais ao processo, pois, conforme já discutido, o processo é semi-automatizado, requerendo a interação de um desenvolvedor.

Convém ressaltar ainda que, diferentemente das abordagens baseadas em estilos e padrões arquiteturais, os métodos apoiados no conhecimento do domínio exigem a reescrita dos padrões a cada nova aplicação ou domínio.

4.4 Recuperação Baseada nos Requisitos Funcionais do Sistema

Nesta seção, são apresentadas abordagens para a recuperação da arquitetura baseadas nos requisitos funcionais da aplicação. Elas utilizam o modelo de casos de uso da aplicação como ponto de partida para a recuperação da arquitetura.

4.4.1 Abordagem Direcionada por Casos de Uso

Em (BOJIC e VELASEVIC, 2000), é apresentada uma abordagem para a recuperação da arquitetura baseada na análise dinâmica do sistema. Os relacionamentos estabelecidos entre os rastros de execução do software e os casos de uso são a base para a reconstrução da arquitetura.

- ***Processo e Técnicas Aplicadas:***

A recuperação se inicia com a identificação dos casos de uso do sistema (ou de uma parte deles) e a definição de casos de teste para os casos de uso. A seguir, o sistema é executado sob uma ferramenta do tipo *profiler* (Microsoft Visual C++ profiler) e os rastros de execução são coletados e armazenados. Os rastros são relacionados a casos de uso.

BOJIC e VELASEVIC (2000) utilizam a técnica de análise de conceito formal, apresentada na seção 3.3.4, para a definição dos módulos do sistema. O objetivo da aplicação da técnica é verificar o compartilhamento de métodos por casos de uso e utilizar este critério como base para a geração dos módulos.

Os casos de uso são relacionados aos métodos que os implementam através dos rastros de execução. Métodos compartilhados por um conjunto de casos de uso são agrupados. Mas, este agrupamento não reflete os módulos do software. Como a unidade a ser agrupada em sistemas OO é a classe, os autores utilizam como critério, no agrupamento, a manutenção da classe no grupo onde se encontra a maior parte dos seus métodos.

- ***Generalidade:***

  A abordagem é genérica do ponto de vista da aplicação da técnica de análise de conceito formal, porém específica para a linguagem C++ na ferramenta de *profiler* utilizada.

- ***Avaliação dos Resultados:***

Os autores utilizam uma aplicação bastante simples, como estudo de caso, para a demonstração do método. Isso dificulta a verificação da eficácia da abordagem. Além disso, nenhum método formal de avaliação é empregado.

Os autores destacam, entretanto, a necessidade de avaliar a abordagem em exemplos reais.

- ***Contribuições e Pontos em Aberto:***

Embora a geração dos módulos na arquitetura se dê com base na análise dinâmica do software, o que representa um diferencial em relação às abordagens anteriores, não fica claro no trabalho a eficácia da aplicação da técnica de análise de conceito formal. Estudos de caso para sistemas reais precisam ser conduzidos.

4.4.2 Abordagem Direcionada pelos Requisitos de Evolução do Software

Em (DING e MEDVIDOVIC, 2001), é apresentada uma abordagem para a recuperação da arquitetura direcionada pelas necessidades de evolução do software e aplicada iterativamente. Os ciclos de recuperação são determinados pelos requisitos de evolução do software e a cada ciclo, os componentes impactos pela evolução do sistema são refinados.

- ***Processo e Técnicas Aplicadas:***

A recuperação se inicia com a extração de um modelo concreto do sistema na forma de um diagrama de classes e a identificação de componentes sobre este modelo através do agrupamento das classes (atividade 1). As classes são agrupadas com base nos relacionamentos entre elas, onde a herança é o relacionamento mais forte, seguido da agregação e composição, associações bi-direcionais e associações unidirecionais.

A segunda fase do processo envolve duas atividades: a elaboração de um modelo arquitetural em alto nível de abstração (arquitetura lógica) com base em um estilo arquitetural identificado para a aplicação (atividade2); e a modelagem de casos de uso para os casos de uso afetados pela evolução ou correspondentes aos novos requisitos funcionais do sistema (atividade 3). A arquitetura em nível lógico é preenchida através dos componentes identificados na atividade 1. Para este preenchimento, os autores analisam diagramas de seqüência construídos para os casos de uso identificados. Convém ressaltar que os diagramas de seqüência são construídos em nível de componentes, refletindo interações entre os componentes identificados na atividade 1.

A arquitetura é recuperada em ciclos e somente os casos de uso afetados pelos requisitos de evolução do software são considerados em cada ciclo. Dessa forma, somente os diagramas de seqüência para estes casos de uso e os seus correspondentes componentes

são mapeados para a arquitetura em cada ciclo. A cada novo pedido de evolução do software, novos componentes são refinados e mapeados para a arquitetura lógica definida na atividade 2.

- ***Generalidade:***

A abordagem é genérica tanto do ponto de vista da aplicação quanto da linguagem de programação. Apresenta restrição apenas quanto ao paradigma de desenvolvimento, que deve ser OO. Porém, a cada nova aplicação, um grande esforço manual é requerido para a elaboração da arquitetura em nível lógico e o conseqüente mapeamento dos componentes.

- ***Avaliação dos Resultados:***

Os autores não detalham os mecanismos de avaliação utilizados.

- ***Contribuições e Pontos em Aberto:***

Assim como a abordagem anterior, de (BOJIC e VELASEVIC, 2000), a arquitetura é refinada com base no modelo dinâmico do software. Porém, nesta proposta, os autores não mencionam o ferramental de suporte utilizado para reconstruir os diagramas de seqüência relacionados aos casos de uso da aplicação. Esta atividade não é trivial, principalmente no que tange à reconstrução de diagramas de seqüência em nível de componentes.

De fato, a abordagem apresentada requer um grande esforço manual na recuperação da arquitetura. A elaboração da arquitetura idealizada da aplicação, por exemplo, também é realizada sem qualquer auxílio de apoio ferramental. Além disso, os agrupamentos de classes em componentes, primeira atividade do processo, dependem da identificação de certos tipos de relacionamentos que não são recuperados diretamente por ferramentas de engenharia reversa (i.e. agregação e composição).

Por outro lado, as maiores contribuições do trabalho estão no apoio à evolução e alterações no software e na proposta de um processo de recuperação de arquitetura iterativo, com um foco bem definido a cada ciclo de recuperação.

4.4.3 Considerações sobre a Recuperação Baseada em Requisitos Funcionais

A única abordagem encontrada, dentre as pesquisadas, que de fato se baseia na análise dinâmica da aplicação para a reconstrução dos elementos arquiteturais do sistema é a de BOJIC e VELASEVIC (2000), apresentada nesta seção.

A maior contribuição das abordagens apoiadas nos requisitos funcionais é o suporte à análise dinâmica da aplicação. A maior parte das abordagens utiliza o modelo estático do software para a reconstrução da sua arquitetura.

O mapeamento de funcionalidades para o código fonte, através dos rastros de execução, é uma informação valiosa para apoiar a reconstrução da arquitetura de software. Na proposta que será apresentada no capítulo 5 deste trabalho, assumimos que este mapeamento apóia a identificação de funcionalidades que devem ser agrupadas em subsistemas por apresentarem objetos compartilhados em sua implementação. Esta hipótese será mais bem explorada no capítulo 5.

4.5 Considerações Finais

A recuperação da arquitetura não representa um problema simples. A arquitetura representa uma abstração, ou seja, algo que não existe concretamente em nenhum artefato de software implementado. Conforme ressaltam KAZMAN e CARRIÈRE (1997), mesmo que um sistema esteja organizado em camadas, a localização e os limites das camadas não ficam nítidos a partir de um exame do código fonte. As abordagens apresentadas neste capítulo visam propor soluções para o problema e oferecem importantes contribuições. Um resumo comparativo das abordagens é obtido na tabela 6.

Abordagens	Objetivos	Visões Recuperadas	Técnicas Centrais	Generalidade	Suporte Ferramental
Reconhecedores de estilos arquiteturais (HARRIS et al., 1997a).	Gerar uma documentação do sistema que demonstre os diferentes estilos arquiteturais identificados.	Estática (lógica), demonstrando os estilos arquiteturais identificados. É gerada uma visão para cada estilo arquitetural.	Consultas para a identificação de estilos, recorte de programas e mapeamento dos padrões para a arquitetura (*pattern matching*). Padrões de codificação para detectar estilos.	Consultas genéricas na sua estrutura, mas específicas para C/Unix nos padrões de codificação pesquisados. Identificam estilos gerais e alguns específicos de C/Unix.	*Framework* integrando ferramentas para a engenharia reversa estática, o reconhecimento de estilos e a visualização da arquitetura.
Recuperação suportada por padrões (GALL e PINZGER, 2002).	Compreensão de programas.	Estática (lógica).	Identificação de expressões no código para a detecção dos padrões arquiteturais.	A aplicação das técnicas depende da escrita de padrões de codificação para a linguagem de programação destino.	Ferramenta de engenharia reversa estática e EsPART p/ o reconhe-cimento de padrões (KNOR et al., 1998).
Dali (KAZMAN e CARRIÈRE, 1997).	Avaliar a conformidade entre a arquitetura projetada e a implementada do software.	Estática (lógica).	Busca de padrões do domínio, através de *queries* no grafo, e agrupamento dos elementos detectados, completando um ciclo de *pattern matching*.	Esquema definido para o repositório atende a sistemas em C++. Os padrões precisam ser reescritos a cada aplicação ou domínio.	Ambiente Dali; Rigi p/ a visualização e manipulação de grafos; PostgreSQL p/ armazenamento etc.

Tabela 6. Quadro Comparativo das Abordagens de Recuperação de Arquitetura.

Abordagens	Objetivos	Visões Recuperadas	Técnicas Centrais	Generalidade	Suporte Ferramental
Agrupamentos de elementos com base nos conceitos do domínio (RIVA e RODRIGUEZ, 2002).	Compreensão de programas.	Estática (lógica) e dinâmica (de cenário).	Pontos fortes: abstrações horizontais e verticais sobre os diagramas de seqüência e a sincronização entre os modelos.	Genérico na manipulação dos modelos, mas específico para a aplicação nas regras de agrupamento aplicadas.	Rigi - visualização de grafo; Prolog - armazenamento do modelo e agregações; extensão da Rigi para diagramas de mensagens.
Recuperação baseada na combinação de grafos (SARTIPI et al., 2000).	Compreensão de programas.	Estática (lógica).	Recuperação baseada no casamento entre dois grafos. Uso de *Data Mining* para identificar itens de dados freqüentes no código, que devem constar de um mesmo módulo.	Mecanismos de *Data Mining* e *matching* são genéricos. Requer um *parser* para a linguagem de programação destino.	Uma ferramentas de engenharia reversa estática e ferramentas desenvolvidas pelos autores.
Recuperação direcionada por casos de uso (BOJIC e VELASEVIC, 2000).	Compreensão de programas e reengenharia para reuso.	Estática (lógica).	Análise de conceito formal (*formal concept analysis*).	Ferramentas de extração de informações do código específicas para C++.	Microsoft Visual Studio *profiler* para a coleta de rastros; ConImp para computar a matriz conceitual (BURMEISTER, 1998) e ferramentas específicas.
Recuperação direcionada pelos requisitos de evolução do software (DING E MEDVIDOVIC, 2001).	Apoiar a evolução do sistema.	Estática (lógica) e de cenário. Fluxos de controle são descritos no nível de componentes para reduzir os diagramas de seqüência.	Engenharia reversa dinâmica, análises estáticas e dinâmicas da aplicação de forma manual.	Genérico para linguagens e domínios, porém requerendo a definição da arquitetura idealizada para cada domínio.	Não mencionado no trabalho.

Tabela 6. Quadro Comparativo das Abordagens de Recuperação de Arquitetura (continuação).

O conjunto de abordagens apresentadas não tem a intenção de ser exaustivo, cobrindo toda a pesquisa e literatura na área. Mas, oferece uma cobertura adequada, descrevendo abordagens de grupos de pesquisa que se destacam e contribuem, de formas alternativas, para a solução do problema. Embora as abordagens apresentem soluções relevantes para a recuperação da arquitetura de software, alguns pontos ainda permanecem em aberto. Dentre eles, destacam-se:

- Necessidade de mecanismos de apoio à tomada de decisão no processo. As abordagens, em geral, apresentam facilidades para a manipulação dos modelos, mas as decisões na montagem dos elementos arquiteturais ficam a cargo do desenvolvedor que recupera a arquitetura.
- Ampliação dos estilos e padrões que podem ser identificados, com base na análise dinâmica do sistema.
- Recuperação da arquitetura de sistemas onde diversas linguagens são usadas.

- Recuperação para sistemas onde somente o código binário se encontra disponível.
- Análise de sistemas que fazem uso de componentes de prateleira (COTS).
- Necessidade de mecanismos que facilitem o mapeamento de requisitos funcionais para os elementos arquiteturais.
- Representação mais formal da arquitetura, seguindo um dos modelos de visões propostos na literatura (ver exemplos na seção 2.2) e, empregando, por exemplo, uma linguagem de descrição arquitetural. RIVA e RODRIGUEZ (2002) apresentam contribuições neste sentido, propondo a integração de visões estáticas e dinâmicas. Porém, a maioria das abordagens trabalha somente sobre a visão lógica (do modelo de visões 4+1 (KRUCHTEN, 1995)). Com relação à representação, a maioria utiliza grafos, notações informais próprias ou UML.

Um outro ponto que merece atenção é quanto à forma de avaliação dos resultados. As abordagens propostas se baseiam em estudos de casos, geralmente representando simplificações de sistemas reais, e tentam medir os resultados através de mecanismos informais, como, por exemplo, a apresentação da arquitetura recuperada para especialistas da aplicação. O problema com os estudos de casos simplificados é que estes não demonstram a escalabilidade do método e podem mascarar os resultados.

Com relação à verificação da arquitetura recuperada, a utilização de métricas é importante para se obter resultados mais precisos. A avaliação informal dos resultados é necessária, pois muitos critérios de verificação da arquitetura são subjetivos. Porém, as métricas ajudam a verificar a eficácia da abordagem, permitindo, por exemplo, comparar diferentes métodos de recuperação. HARRIS et al., 1997a apresentam propostas de métricas, verificando a cobertura de código atingida por cada estilo arquitetural recuperado.

5. Proposta de Abordagem para a Recuperação da Arquitetura de Software Visando a Geração de Artefatos para a Engenharia de Domínio

5.1 Introdução

Neste capítulo, será apresentada uma proposta de abordagem para a recuperação da arquitetura de software de sistemas legados, visando apoiar a definição de uma arquitetura de referência para um domínio. A abordagem proposta está dividida em duas fases: a primeira envolve a definição de um processo para a recuperação da arquitetura de software de sistemas legados; a segunda envolve o estabelecimento de critérios e técnicas para a comparação das arquiteturas recuperadas, visando a definição de uma arquitetura de referência para um domínio.

O processo proposto se baseia em modelos estáticos e dinâmicos extraídos do código fonte para a reconstrução da arquitetura. Foca mais fortemente, entretanto, nos modelos dinâmicos. A análise dos modelos dinâmicos será priorizada porque a maior parte das abordagens de recuperação de arquitetura, encontradas na literatura, exploram a análise estática do sistema. Além disso, acreditamos que os modelos dinâmicos representem uma rica fonte de informação sobre o software, a qual oferece um conhecimento útil para a descoberta da sua organização e comportamento. O'BRIEN e STOERMER (2001), por exemplo, utilizam as visões de execução e de fluxo de dados[12] das arquiteturas de sistemas legados para a identificação de alguns estilos arquiteturais empregados.

Diversos trabalhos de análise dinâmica de sistemas vêm sendo propostos, visando apoiar a compreensão de programas (TAKAHASHI e ISHIOKA, 2000) e (JERDING et al., 1997). Eles apresentam idéias que podem ser aproveitadas, estendidas e incorporadas a processos de recuperação da arquitetura de software.

A recuperação da arquitetura, neste trabalho, prevê o mapeamento de funcionalidades para o código fonte. Os rastros de execução coletados do sistema, durante a engenharia reversa dinâmica, são relacionados a cenários de casos de uso da aplicação. Este relacionamento visa não só apoiar a tomada de decisão na montagem da arquitetura, uma

[12] A visão de execução descreve o comportamento do sistema, ao passo que a visão de fluxo de dados demonstra como os dados fluem entre os componentes de software do sistema. Ambas as perspectivas retratam uma visão dinâmica da arquitetura.

vez que classes que participam da realização de funcionalidades similares devem ser agrupadas em conjunto, como também permitir a reconstituição de outras visões arquiteturais, além da lógica (estática), como a visão de processo e de cenários do modelo de visões 4+1 de KRUCHTEN (1995) – veja seção 2.2.1, o qual é adotado neste trabalho.

A geração de artefatos para a Engenharia de Domínio (ED) representa o objetivo principal desta proposta. Faz parte, portanto, do escopo deste trabalho, o estabelecimento de critérios e técnicas para a comparação das arquiteturas recuperadas, apoiando a definição de uma arquitetura de referência para um domínio. Além da arquitetura de referência, a abordagem visa suportar a geração de outros artefatos para a Engenharia de Domínio, como as características funcionais (KANG et al., 1990) do domínio, por exemplo. Como a ED envolve o desenvolvimento para reutilização, a arquitetura de referência e os artefatos gerados a partir desta proposta, serão posteriormente reutilizados em novas aplicações desenvolvidas no domínio.

Para a apresentação desta proposta de tese, o restante do capítulo está organizado da seguinte forma: seção 5.2 apresenta o processo proposto para a recuperação de arquitetura, através das suas atividades e técnicas empregadas; seção 5.3 apresenta a abordagem para representação da arquitetura adotada; seção 5.4 descreve os critérios e técnicas para a comparação das arquiteturas recuperadas, apoiando a definição de uma arquitetura de referência para o domínio; seção 5.5 finaliza o capítulo, tecendo considerações acerca das questões de pesquisa abordadas.

5.2 Processo Proposto para a Recuperação de Arquitetura

O processo prevê a recuperação da arquitetura de software de sistemas legados orientados a objeto. O foco em sistemas OO se deve à primeira premissa deste trabalho: (**hipótese 1**) os sistemas orientados a objeto serão os sistemas legados de um futuro próximo. Esta premissa se estabelece em função do volume de desenvolvimento de software que vem ocorrendo neste paradigma.

O processo proposto para a recuperação da arquitetura engloba sete atividades e prevê uma recuperação iterativa e incremental, seguindo princípios do modelo de ciclo de vida em espiral (PRESSMAN, 2001) para o desenvolvimento de software. De acordo com as

fases estabelecidas pelo ciclo de vida em espiral, o processo está organizado da seguinte forma:

1. **Análise de Viabilidade e Planejamento da Recuperação**
2. **Engenharia**
    a. **Modelagem de Casos de Uso**
    b. **Engenharia Reversa Estática**
    c. **Engenharia Reversa Dinâmica**
    d. **Definição dos Elementos Arquiteturais**
    e. **Mapeamento de Padrões para a Arquitetura**
3. **Avaliação da Arquitetura Recuperada**

A figura 16 ilustra o processo proposto. Este modelo foi elaborado na máquina de processos do ambiente Odyssey, a Charon (MURTA, 2002), a qual segue a notação do diagrama de atividades da UML, com algumas extensões. O diagrama de atividades é utilizado para a modelagem de *workflows*, sendo adequado à representação de processos. Na máquina Charon, além da notação proposta pela UML, é possível definir se uma atividade é primitiva ou composta e a probabilidade, em termos percentuais, em se seguir um caminho ou outro a cada decisão do processo. Atividades compostas, como "Planejar" na figura 16, possuem um pequeno símbolo de *workflow* em seu interior à direita, indicando que elas possuem um sub-processo interno com as suas sub-atividades relacionadas. Neste processo, a maior parte das atividades propostas é composta e pode ser subdividida em atividades ou tarefas menores, com exceção da engenharia reversa estática, que é primitiva.

A arquitetura é recuperada em ciclos, de forma gradual. Algumas atividades, como a modelagem de casos de uso e a engenharia reversa dinâmica, devem ser executadas repetidamente.

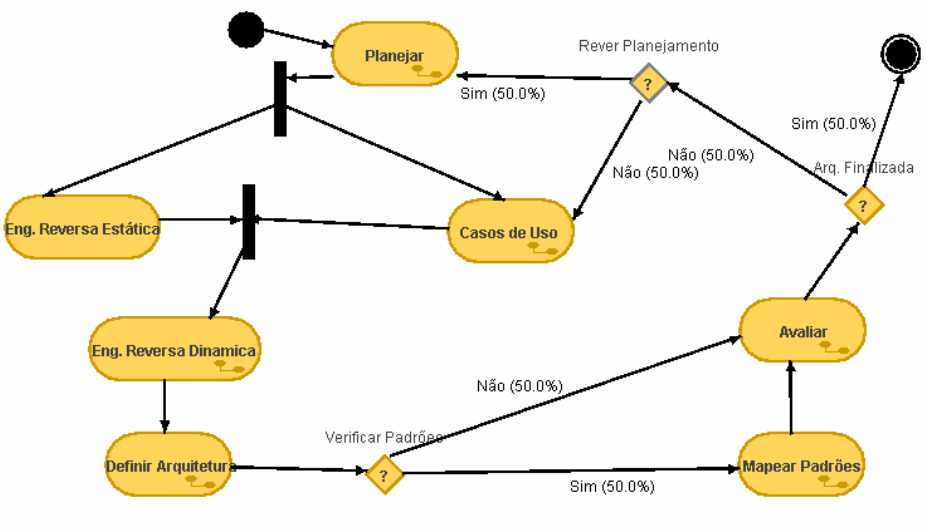

Figura 16. Processo Proposto para a Recuperação de Arquitetura.

A primeira atividade, conforme mostra a figura 16, é o planejamento do processo. Ele envolve uma análise de viabilidade da recuperação da arquitetura para o sistema legado em questão e a elaboração de um plano de recuperação de arquitetura, o qual deve prever os ciclos de recuperação necessários.

Depois de realizado o planejamento inicial, a modelagem de casos de uso e a engenharia reversa estática podem ser iniciadas em paralelo. A engenharia reversa estática é executada uma única vez. Todo o modelo estático do sistema é extraído de uma vez. A modelagem de casos de uso, por sua vez, é executada em ciclos. A cada ciclo de recuperação de arquitetura, um conjunto de funcionalidades será tratado.

A engenharia reversa dinâmica só pode ser iniciada depois de extraído o modelo estático do software e especificados os casos de uso a serem tratados no ciclo corrente de recuperação da arquitetura. Esta atividade visa a coleta, análise e representação gráfica dos rastros de execução da aplicação, os quais são relacionados a cenários de casos de uso. A análise dos rastros de execução apóia a identificação dos elementos arquiteturais do software, atividade esta que sucede a engenharia reversa dinâmica.

Uma vez que alguns elementos arquiteturais tenham sido definidos, o desenvolvedor pode decidir pelo mapeamento ou não de padrões arquiteturais (e de projeto) para a arquitetura. Esta atividade envolve a identificação de padrões na arquitetura que está sendo recuperada e não precisa ocorrer em todos os ciclos do processo. O desenvolvedor decide quando quer realizá-la. O mapeamento de padrões arquiteturais para a arquitetura

pode influenciar na própria definição dos elementos arquiteturais ou pode gerar novas visões da arquitetura. Os padrões de projeto apenas refinam elementos arquiteturais já recuperados.

A cada ciclo de recuperação, uma avaliação da arquitetura recuperada deve ser conduzida. Após a avaliação, o desenvolvedor decide se o processo está finalizado ou se é necessário iniciar um novo ciclo de recuperação da arquitetura, o qual se inicia pela modelagem de casos de uso. Neste momento, o desenvolvedor também pode decidir pela revisão do planejamento inicial estabelecido.

Uma descrição detalhada das atividades propostas no processo é apresentada nas seções que se seguem.

5.2.1 Análise de Viabilidade e Planejamento do Processo

A abordagem proposta, conforme já mencionado, visa apoiar a Engenharia de Domínio, mais especificamente a etapa de Projeto do Domínio. Dessa forma, assume-se que antes da recuperação da arquitetura dos sistemas legados, a organização terá realizado um Estudo de Viabilidade do Domínio, etapa esta contemplada em processos de ED, como o Odyssey–DE de BRAGA (2000) e o processo proposto por SIMOS (1996). Nesta atividade, os domínios de interesse na empresa são avaliados quanto ao seu potencial para a reutilização. São analisados critérios técnicos, organizacionais e sociais[13] em relação aos domínios, como: a existência de uma base significativa de sistemas legados no domínio, os futuros sistemas a serem desenvolvidos no domínio, a necessidade de melhoria dos sistemas existentes, o conhecimento em reutilização na empresa e a experiência prática na adoção de reutilização por parte dos desenvolvedores envolvidos no processo, a maturidade do domínio, a disponibilidade de recursos na organização para a ED etc.

Uma vez que um domínio tenha sido aprovado para a aplicação do processo de ED, sistemas legados candidatos no domínio deverão ser selecionados. Um número comum de aplicações para análise em um domínio, que propicie a identificação de artefatos reutilizáveis, é de 3 a 4 aplicações (O'BRIEN e STOERMER, 2001). Além do número de aplicações, O'BRIEN e STOERMER (2001) destacam a importância da seleção de sistemas representativos no domínio, ou seja, sistemas que envolvam, por exemplo,

[13] Para maiores detalhes sobre os critérios de avaliação de domínios, consulte (BRAGA, 2000).

diferentes clientes, diferentes plataformas de hardware, protocolos ou conjuntos de características.

Neste trabalho, a seleção dos sistemas candidatos em um domínio leva em conta questões diferentes daquelas propostas por O'BRIEN e STOERMER (2001), englobando critérios, como: o paradigma de desenvolvimento empregado; a disponibilidade de fontes de informação sobre o sistema; atributos do software; e a disponibilidade de código fonte. Segue a descrição detalhada dos critérios para a avaliação dos sistemas legados candidatos em um domínio.

a) ***Paradigma de desenvolvimento***: para o processo proposto, os sistemas devem necessariamente seguir o paradigma OO, mais especificamente devem estar escritos na linguagem Java.
b) ***Fontes de informação:*** é necessário que haja alguma documentação disponível sobre o sistema, que facilite a sua compreensão, ou que especialistas estejam disponíveis para esclarecer as dúvidas do desenvolvedor que guia o processo. O processo é semi-automatizado e o desenvolvedor terá que tomar decisões ao longo da recuperação da arquitetura. Por este motivo, ele deve explorar as fontes de informação sobre o software e adquirir conhecimento. Além da documentação e dos especialistas, outras fontes de informação podem apoiar a recuperação da arquitetura (veja seção 3.2.1).
c) ***Disponibilidade de código fonte:*** as ferramentas de engenharia reversa estática, conforme mencionado na seção 3.3.1.1, ainda requerem a análise do código fonte para a extração dos modelos de projeto do software.
d) ***Atributos do software***: alguns atributos de software, como seu tamanho, complexidade e acoplamento entre classes[14], oferecem um indicativo, juntamente com o volume de fontes de informação disponíveis, do grau de dificuldade e conseqüente esforço necessário para a reconstrução da arquitetura. Quanto maior for o sistema, em termos de número de classes e

[14] Métricas para extrair o tamanho, a complexidade e o acoplamento de entidades de software OO são descritas na literatura (LORENZ e KIDD, 1994) (CHIDAMBER e KEMERER, 1994) e podem ser geradas automaticamente por ferramentas disponíveis gratuitamente (e.g. Eclipse (ECLIPSE ORG, 2003b) e JDepend (CLARKWARE CONSULTING INC., 2003)).

linhas de código, e mais acopladas e complexas forem as classes, maior o esforço requerido para a reconstrução da arquitetura. Em função do esforço estimado e da sua disponibilidade de recursos, a organização pode decidir pela viabilidade ou não de se recuperar a arquitetura do sistema.

Uma vez verificada a viabilidade de recuperação da arquitetura de cada sistema legado, o planejamento do processo para cada sistema deve ser iniciado. Ele envolve a estimativa de esforço, prazo e custo do processo, com base nos critérios mencionados no item d), e a organização dos ciclos de recuperação, estabelecendo-se um cronograma previsto.

A definição dos ciclos de recuperação da arquitetura é feita com base na listagem e priorização das funcionalidades[15] do software. A priorização leva em conta o impacto da funcionalidade na arquitetura e o seu nível de importância para o usuário. Uma vez listadas e ordenadas as funções do software, elas são divididas em grupos, levando em conta a sua prioridade e complexidade. Cada grupo caracteriza e direciona um ciclo de recuperação da arquitetura.

Convém ressaltar que, nesta etapa, as funcionalidades listadas ainda não são detalhadas. O artefato resultante da atividade é o **Plano de Recuperação da Arquitetura de Software**.

5.2.2 Modelagem de Casos de Uso

As funcionalidades listadas na etapa de planejamento, e que devem ser tratadas no ciclo corrente de recuperação da arquitetura, são detalhadas através de casos de uso da UML.

A cada caso de uso, um ou mais cenários de execução da aplicação são associados, representando possíveis caminhos para a utilização do software. Nesta proposta, deve ser descrito o cenário principal de cada caso de uso e alguns cenários alternativos, que sejam relevantes. Porém, o grau de formalismo necessário nesta descrição é menor do que o que é exigido em um processo de desenvolvimento de software tradicional. O objetivo desta

[15] Uma das fontes de informação para a identificação das funcionalidades do sistema é a própria aplicação em sua versão executável. A análise das interfaces de usuário do software e das suas opções de menu ajuda na identificação das funcionalidades.

especificação de cenários é guiar o desenvolvedor na execução do sistema legado e auxiliar no mapeamento de funcionalidades para o código fonte.

Os **artefatos** resultantes desta atividade são: **o modelo de casos de uso do software,** contendo os casos de uso especificados até o momento **e os cenários associados**.

5.2.3 Engenharia Reversa Estática

As atividades de modelagem de casos de uso e de engenharia reversa estática podem ser executadas em paralelo. Uma não é pré-requisito para a outra. O objetivo da engenharia reversa estática é a recuperação do modelo estrutural da UML, o qual contém: classes; pacotes (onde as classes se localizam no código fonte); interfaces realizadas pelas classes e utilizadas; e os relacionamentos entre estes elementos, como herança, associação e dependência. Relacionamentos de agregação e composição não são recuperados pela engenharia reversa estática.

A atividade de engenharia reversa estática não é executada de forma iterativa. O modelo estrutural é extraído de uma única vez, devendo ser armazenado em um repositório para posterior análise.

O **artefato** resultante da atividade é o **modelo estrutural da UML**.

5.2.4 Engenharia Reversa Dinâmica

A engenharia reversa dinâmica, por sua vez, deve ser realizada de forma iterativa. A cada ciclo de recuperação da arquitetura, a engenharia reversa dinâmica é executada para os cenários de casos de uso selecionados naquele ciclo. Por este motivo, a modelagem de casos de uso é um pré-requisito para esta atividade.

A engenharia reversa dinâmica é uma atividade composta, envolvendo um conjunto de sub-atividades, a saber:

- Coleta dos rastros de execução;
- Análise dos rastros coletados;
- Extração de diagramas de seqüência.

Para a extração dos diagramas de seqüência, é necessário que o modelo estrutural da aplicação contenha os tipos de objetos que aparecem nas interações. Por isso, a engenharia reversa estática também é um pré-requisito à engenharia reversa dinâmica, neste processo.

Os **artefatos** gerados pela engenharia reversa dinâmica são: os **rastros de eventos** coletados**, associados aos cenários de casos de uso**; **padrões de interação** (veja seção 3.3.1.2) detectados nos rastros; e **diagramas de seqüência**, associados aos cenários de casos de uso.

5.2.4.1 Coleta dos Rastros de Execução

Os rastros (ou *traces*) são compostos dos eventos gerados ao longo da execução do sistema, os quais, para sistemas OO, envolvem as mensagens enviadas entre os objetos. Cada mensagem é composta de um objeto, seu tipo e método emissores da mensagem e um objeto, seu tipo e método receptores. A fim de permitir a coleta dos rastros, o sistema deve ser exercitado, sob monitoramento, para os cenários de caso de uso descritos.

Uma vez coletados, os rastros devem ser associados aos cenários de casos de uso, garantindo, dessa forma, o mapeamento de funcionalidades para o código. A figura 17 ilustra este esquema. Nesta figura, os rastros estão representados através de diagramas de seqüência da UML. O mapeamento das funções do software para as suas estruturas de código é a base para a reconstrução dos elementos arquiteturais, discutida na seção 5.2.5.

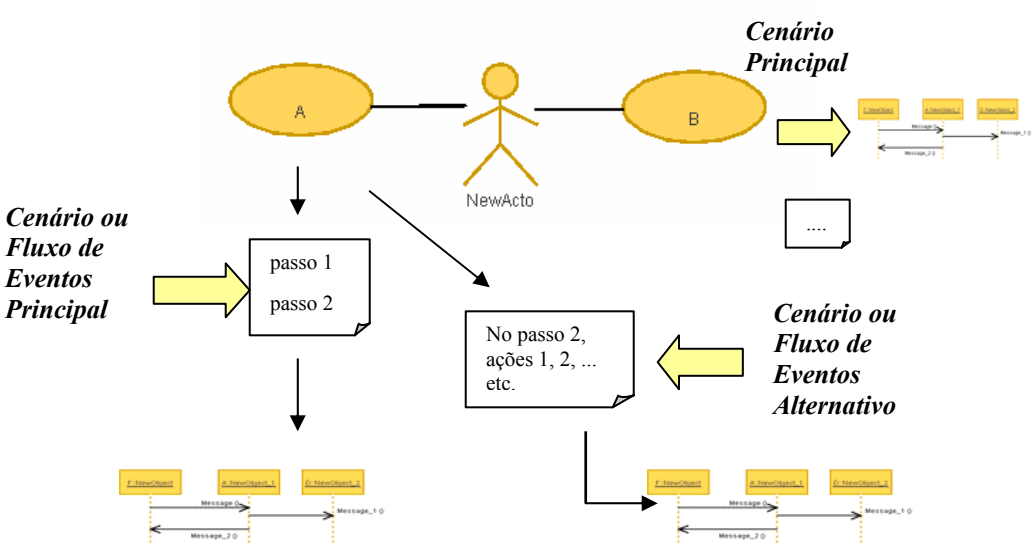

Figura 17. Relacionamento entre Cenários de Casos de Uso e *Traces* de Execução.

Convém ressaltar, que o sistema deverá ser executado para diferentes cenários de casos de uso, permitindo, dessa forma, alcançar uma maior cobertura de código na recuperação da arquitetura.

5.2.4.2 Análise dos Rastros Coletados

O volume de eventos gerados durante a execução de um caso de uso é muito grande. São centenas, às vezes milhares de eventos, para um cenário simples de caso de uso. Este é um problema enfrentado em todo trabalho de análise dinâmica de software: o volume de eventos coletados durante a execução. A fim de minimizar este problema, o qual se intensifica quando os rastros são traduzidos para diagramas de seqüência da UML, filtros podem ser estabelecidos, antes e após a coleta, e padrões de eventos podem ser detectados nos rastros, sendo representados através de uma mensagem de mais alto nível nos diagramas de seqüência.

Neste trabalho, pretendemos investigar a detecção de padrões de interação nos rastros de execução, assim como nos trabalhos de TAKAHASHI e ISHIOKA (2000) e JERDING et al. (1997). Esta detecção visa não só apoiar a simplificação dos rastros, em função dos problemas citados, como também suportar a definição dos elementos arquiteturais, que é discutida na seção 5.2.5.

Neste trabalho, a proposta é detectar padrões de interação utilizando técnicas de *Data Mining*. Mapeando os conceitos da área de Banco de Dados para a Engenharia Reversa Dinâmica, caracterizamos: as transações como os rastros de eventos; os itens de dados de uma transação, como as mensagens enviadas entre os objetos; e os itens de dados freqüentes buscados, como as seqüências de mensagens que ocorrem repetidamente nos rastros, ou, os padrões de interação. A proposta é estudar algoritmos de *Data Mining* e selecionar um que seja adequado ao problema. Neste mecanismo de detecção de rastros, aplicando princípios de *Data Mining*, os cenários de casos de uso funcionam como as transações de suporte aos padrões de eventos detectados.

Duas observações devem ser feitas em relação aos padrões de interação que pretendem ser detectados. Primeiro, as mensagens que compõem os padrões são caracterizadas pelos seus tipos e métodos emissores da mensagem e seus tipos e métodos receptores, sem levar em conta as instâncias. As instâncias tendem a variar nos diferentes cenários rastreados, o que poderia mascarar um padrão de interação. Em segundo lugar, os padrões de interação assumem mensagens entremeadas, ou seja, mensagens que ocorrem entre as mensagens que fazem parte do padrão. O importante, para a caracterização de um

padrão de interação, é encontrar uma mesma seqüência de mensagens sendo enviadas em uma mesma ordem, independente de serem contíguas ou não.

5.2.4.3 Extração de Diagramas de Seqüência

A extração de diagramas de seqüência completa o ciclo de engenharia reversa dinâmica. Os diagramas também devem ser associados aos cenários de casos de uso, assim como os *traces* de execução. Para contribuir com a simplificação dos diagramas de seqüência, além da representação dos padrões de interação como uma mensagem única de mais alto nível, outros mecanismos são propostos, como, filtros sobre os rastros coletados e a manipulação dos diagramas pelo desenvolvedor.

Nos filtros, mensagens enviadas a classes de bibliotecas externas podem ser omitidas. A idéia é que o desenvolvedor também estabeleça os seus próprios critérios de filtro. Quanto à manipulação interativa dos diagramas, o desenvolvedor pode decidir agrupar mensagens ou objetos, como no trabalho de RIVA e RODRIGUEZ (2002) – seção 4.3.2 – ou visualizar os diagramas até um nível de profundidade de mensagens determinado (2 níveis de chamadas, 3 níveis etc.).

Um dos objetivos, neste trabalho, é recuperar algumas visões arquiteturais dinâmicas, seguindo o modelo de visões proposto por KRUCHTEN (1995) – veja seção 2.2.1. A nossa hipótese é que somente uma visão estática da arquitetura não é suficiente para a sua compreensão. Neste contexto, o diagrama de seqüência é utilizado para a remontagem das visões de processo e de cenário de KRUCHTEN (1995). A fim de que as visões dinâmicas reflitam de fato a arquitetura da aplicação e estejam sincronizadas com a visão estática, pretende-se mapear os elementos arquiteturais reconstruídos na visão estrutural (lógica) para os diagramas de seqüência, assim como no trabalho de RIVA e RODRIGUEZ (2002).

5.2.5 Definição dos Elementos Arquiteturais

A reconstrução dos elementos arquiteturais se dá, prioritariamente, com base na detecção de padrões de interação, associados a cenários de casos de uso. Duas hipóteses são formuladas neste trabalho para a reconstrução dos elementos arquiteturais:

- **Hipótese 2**: os padrões de interação ajudam a identificar objetos que colaboram com grande freqüência e que, portanto, podem ser agrupados para formar elementos arquiteturais.
- **Hipótese 3**: os padrões de interação ajudam a identificar casos de uso que implementam funcionalidades similares e que, portanto, devem ser agrupados para formar subsistemas.

Nas seções subseqüentes, a reconstrução dos elementos arquiteturais é explorada.

Os artefatos gerados nesta atividade são os **elementos arquiteturais do sistema** legado e suas **conexões**.

5.2.5.1 Utilização de Padrões de Interação e Métricas

Os tipos de objetos que compõem os padrões de interação que possuem um número de transações de suporte grande, ou seja, padrões de interação que aparecem em um grande número de rastros de execução (ou casos de uso), tendem a representar classes centrais na arquitetura. Estas classes oferecem serviços requeridos por diversas funcionalidades do software. A abordagem proposta apresenta, neste caso, a sugestão de agrupamento destas classes em um elemento arquitetural central na arquitetura (elemento do *kernel* da aplicação), desempenhando algum serviço essencial, e para o qual diversos outros elementos arquiteturais apresentarão uma dependência.

O número mínimo de transações de suporte para guiar esta tomada de decisão precisa ser definido com base na execução do processo em sistemas reais. A realização de estudos de caso está prevista no planejamento deste trabalho e será discutida no capítulo 7.

Os padrões de interação, por sua vez, que aparecem em um número menor de cenários, correspondendo a um pequeno conjunto de casos de uso, podem indicar que estes casos de uso implementam funcionalidades similares e que, portanto, devem ser agrupados em subsistemas. O agrupamento dos casos de uso resulta no agrupamento das classes que implementam estes casos de uso, levando à definição de um novo elemento arquitetural.

Durante a aplicação destas regras para o agrupamento de classes em elementos arquiteturais, um problema pode se apresentar: a mesma classe pode constar de mais de um padrão de interação e, conseqüentemente, de mais de um elemento arquitetural. Nestes

casos, para se determinar se a classe deve fazer parte de um elemento arquitetural ou de outro, métricas de acoplamento estáticas e dinâmicas para sistemas orientados a objetos devem ser utilizadas. Quanto maior for o acoplamento de uma determinada classe (ou dos seus objetos) com as classes de um elemento arquitetural, maior a tendência de que esta classe faça parte deste elemento na arquitetura. Considerações sobre métricas de acoplamento estáticas e dinâmicas para sistemas OO foram feitas na seção 3.3.2.1.

5.2.5.2 Manipulação de Elementos na Arquitetura

Além dos elementos arquiteturais sugeridos com base nos padrões de interação e na aplicação de métricas, o desenvolvedor pode gerar ou re-agrupar elementos na arquitetura de forma interativa, manipulando o modelo diretamente ou através da aplicação de consultas (*queries*).

Com o conhecimento que o desenvolvedor vem adquirindo sobre a aplicação ao longo do processo de reconstrução da arquitetura, ele pode decidir pela montagem ou reorganização manual de alguns elementos arquiteturais. Nesta atividade de manipulação do modelo, o desenvolvedor aplica *queries* para a seleção e movimentação das classes na arquitetura. A figura 18 ilustra o esquema de manipulação interativa do modelo.

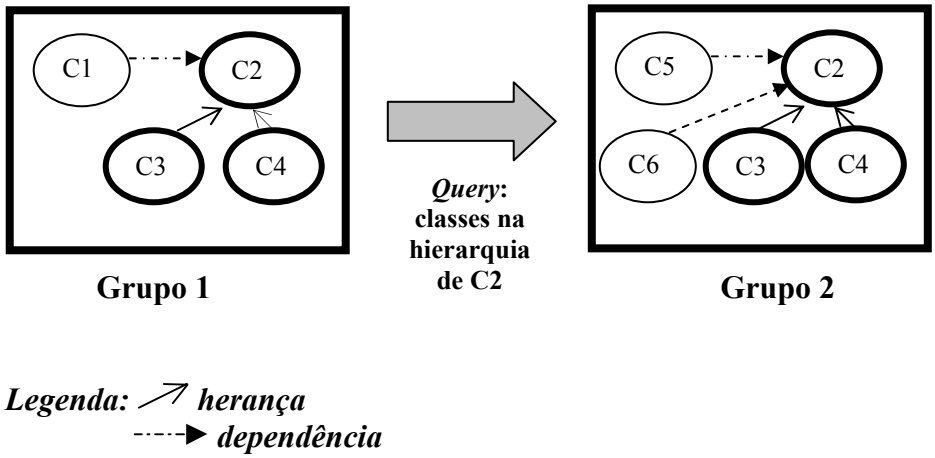

Figura 18. Manipulação de Classes entre *Clusters* no Modelo Estático.

Na figura 18, as classes C2, C3 e C4 foram movidas para o grupo 2 na arquitetura. Como no grupo 2, as classes C5 e C6 apresentam dependência para a classe C2, e no grupo 1, somente a classe C1 apresenta uma dependência, o desenvolvedor julgou mais adequado

que a hierarquia de classes iniciada por C2 ficasse neste 2º grupo. Como na sugestão de agrupamentos oferecida pela abordagem proposta a hierarquia de classes iniciada por C2 tinha ficado no grupo 1, o desenvolvedor aplica uma *query* sobre o modelo e movimenta as classes para o grupo 2, obtendo a reorganização desejada da arquitetura.

Neste momento, o desenvolvedor também pode fazer uso das métricas de acoplamento entre classes para apoiar a sua tomada de decisão.

5.2.5.3 Considerações sobre a Montagem dos Elementos Arquiteturais

Neste processo de remontagem dos elementos arquiteturais, o desenvolvedor deve estar atento aos elementos que já foram recuperados anteriormente. Uma vez que o processo de recuperação da arquitetura é iterativo e incremental, a geração de novos elementos arquiteturais pode levar à reorganização de elementos que já foram gerados. Por este motivo, os elementos arquiteturais não são remontados de forma automática, mas de maneira semi-automática, com a participação do desenvolvedor.

Além dos componentes arquiteturais provenientes de classes da própria aplicação, as bibliotecas externas acessadas pelo sistema legado também devem ser representadas na arquitetura. A compreensão da arquitetura interna destas bibliotecas não é de interesse nesta proposta, e, portanto, as mesmas são representadas como componentes caixa-preta na arquitetura. Para descobrir as bibliotecas acessadas pelo sistema, o desenvolvedor pode consultar arquivos de compilação e execução do software (arquivo de *build – build file*) e verificar, nos rastros de execução, as chamadas enviadas para bibliotecas externas.

5.2.6 Mapeamento de Padrões para a Arquitetura

Os trabalhos de HARRIS et al. (1997a) e de GALL e PINZGER (2002) – seção 4.2 - se apóiam na identificação de estilos e padrões arquiteturais no código para a reconstrução da arquitetura. Os autores se baseiam nos padrões de codificação da linguagem de programação para descobrir padrões arquiteturais presentes nos sistemas legados. Esta forma de detecção apresenta limitações com relação aos padrões que podem ser identificados. Nem todo padrão arquitetural pode ser detectado através de estruturas do código fonte, sobretudo os padrões caracterizados pelo fluxo de dados e de controle que estabelecem, como é o caso dos padrões camadas e tubos e filtros. A detecção de padrões deste tipo requer uma análise comportamental do software.

Neste contexto, a quarta hipótese que se formula nesta proposta (**hipótese 4**) é que a análise dinâmica do software pode apoiar a identificação semi-automática de padrões arquiteturais caracterizados pela hierarquia de controle e de fluxo de dados que estabelecem. Esta hipótese será investigada neste trabalho.

A identificação de padrões arquiteturais pode levar à reestruturação de elementos definidos na arquitetura ou à geração de novas visões arquiteturais. A cada padrão arquitetural identificado, os atributos de qualidade relacionados ao padrão devem ser registrados.

Além dos padrões arquiteturais, os padrões de projeto (GAMMA et al., 1995) também auxiliam na compreensão da arquitetura. Os padrões de projeto apresentam uma escala menor que os padrões arquiteturais, sendo úteis para o refinamento de componentes e conectores na arquitetura recuperada. A detecção de padrões de projeto também é considerada nesta proposta.

5.2.7 Avaliação da Arquitetura Recuperada

Ao final de cada ciclo de execução do processo, a arquitetura recuperada deve ser avaliada. Os trabalhos de recuperação de arquitetura apresentados no capítulo 4 apresentam critérios informais para a avaliação da arquitetura, com exceção do trabalho de HARRIS et al. (1997a), que faz uso de métricas de cobertura de código para verificar o percentual de código englobado por cada estilo arquitetural identificado.

Na realidade, a informalidade é um problema geral na avaliação de trabalhos de remodularização de software. Conforme afirmam ANQUETIL e LETHBRIDGE (2003), muitos critérios utilizados para a avaliação de uma organização modular de software são subjetivos, incluindo o suporte dado pela estrutura modular à compreensão do programa e manutenção do sistema, os conceitos representados pelos módulos recuperados etc.

Entretanto, ANQUETIL e LETHBRIDGE (2003) destacam dois critérios formais, alternativos, que podem apoiar a avaliação e ser obtidos de forma automática:

- **Critério de projeto**: agrupamentos devem refletir um bom projeto, no que diz respeito aos valores de métricas de coesão e acoplamento.
- **Critério de decomposição de referência**: agrupamentos devem estabelecer uma correspondência para um projeto de referência, definido por projetistas.

Nesta proposta, a arquitetura recuperada deve ser avaliada de três formas: manualmente, por especialistas da aplicação, assim como na maioria dos trabalhos de recuperação da arquitetura; e, automaticamente, através de métricas de cobertura de código, acoplamento e coesão.

A cobertura de código deve ser avaliada verificando-se o percentual de classes do sistema que compõem os elementos arquiteturais recuperados e o percentual de métodos destas classes cobertos pelos modelos dinâmicos analisados. Dessa forma, verifica-se a abrangência da arquitetura recuperada em relação ao sistema legado. As métricas de acoplamento e coesão visam avaliar a qualidade da arquitetura recuperada.

A avaliação de especialistas, assim como as métricas de acoplamento e coesão, verificam a qualidade da arquitetura recuperada. Porém, através de critérios subjetivos, que envolvem a análise do quanto a arquitetura recuperada de fato apóia a compreensão do sistema, refletindo seus conceitos e funcionalidades e o quanto ela reflete de fato os padrões arquiteturais originalmente empregados na aplicação.

5.3 Representação da Arquitetura

A arquitetura deve ser graficamente representada através de diagramas de pacotes e de classes da UML. Porém, a fim de prover uma maior semântica aos elementos arquiteturais recuperados, descrevendo os componentes, suas interfaces e conectores na arquitetura, a proposta é incorporar uma ADL ao ambiente de modelagem onde a arquitetura é recuperada. A xADL (DASHOFY et al., 2002) está sendo estudada como uma possível proposta. Ela prevê a descrição da arquitetura em XML e apresenta facilidades de extensão dos seus esquemas – veja seção 2.3.2.

Neste trabalho, assumimos a necessidade de representar diferentes perspectivas do sistema, a fim de facilitar a compreensão da sua arquitetura. Para contemplar as diferentes perspectivas arquiteturais, o modelo de visões adotado é o de KRUCHTEN (1995) (veja seção 2.2.1). Neste modelo, as seguintes visões são contempladas: lógica, de desenvolvimento, de processo e física. A visão lógica é representada graficamente através dos elementos da visão estrutural da UML, ou seja, diagramas de classes e de pacotes. A visão de cenários é demonstrada através dos diagramas de seqüência, recuperados e associados aos cenários de casos de uso do sistema.

A visão de desenvolvimento de KRUCHTEN (1995) demonstra as estruturas existentes na implementação do sistema. Estas estruturas são inicialmente recuperadas através da engenharia reversa estática. Porém, a organização original do sistema, em termos do seu código fonte, é perdida, uma vez que o modelo estrutural é reorganizado a fim de refletir a arquitetura da aplicação. Com o intuito de manter uma ligação entre a visão lógica (reconstruída) e a visão de desenvolvimento da arquitetura, duas alternativas são propostas neste trabalho: o registro da URL (onde estão localizados os arquivos originais do código fonte) na especificação da arquitetura em xADL (a xADL já prevê esta informação); a criação de mais uma visão no ambiente de modelagem onde a arquitetura é recuperada, contemplando a perspectiva de desenvolvimento.

As visões de processo e física de KRUCHTEN (1995) fazem mais sentido para sistemas distribuídos. Sistemas distribuídos não são o foco deste trabalho. Entretanto, como os estudos de caso serão realizados sobre sistemas escritos na linguagem de programação Java, múltiplas tarefas (*threads*) poderão compor os fluxos de execução recuperados do software. A representação de múltiplas *threads* pode ser feita através de diagramas de atividade da UML.

5.4 Apoio à Definição de uma Arquitetura de Referência para o Domínio

As arquiteturas recuperadas dos sistemas legados devem ser comparadas, gerando-se um mapa de semelhanças e diferenças entre os sistemas, o qual pode ser utilizado para apoiar a criação de uma arquitetura de referência para o domínio. O'BRIEN e STOERMER (2001) propõem o método MAP (*Mining Architectures for Product Line Evaluations*), que apresenta objetivos similares aos deste trabalho, buscando a construção de uma arquitetura de Linha de Produtos com base nas arquiteturas recuperadas de sistemas legados. O objetivo dos autores é migrar as aplicações de um mesmo domínio ou de domínios similares para uma abordagem de Linha de Produtos.

O'BRIEN e STOERMER (2001) selecionam sistemas candidatos em um domínio (ou em domínios similares) e extraem as suas arquiteturas, utilizando para a reconstrução das arquiteturas o ambiente Dali *Workbench* (KAZMAN e CARRIÈRE, 1997) – veja seção 4.3.1. Eles estendem a abordagem de reconstrução de arquitetura proposta no Dali *Workbench,* incorporando uma etapa de Qualificação da Arquitetura, na qual é previsto o mapeamento de estilos arquiteturais e atributos de qualidade para a arquitetura recuperada,

gerando diferentes visões arquiteturais. Esta etapa é similar à etapa de Mapeamento de Padrões para a Arquitetura proposta neste documento, porém em (O'BRIEN e STOERMER, 2001) não são estabelecidos critérios e técnicas formais para a detecção dos padrões e a qualificação da arquitetura é realizada manualmente.

Depois de extraídas as arquiteturas, O'BRIEN e STOERMER (2001) avaliam as semelhanças e diferenças entre elas, verificando a viabilidade de migração dos sistemas para uma Linha de Produtos. A comparação entre os sistemas em nível arquitetural leva em conta os seguintes aspectos:

- As características comuns e específicas dos sistemas e de seus clientes, em termos de funcionalidades, protocolos utilizados, sistema operacional e hardware;
- Vocabulário do domínio x vocabulários especializados;
- As estruturas dos sistemas recuperados, incluindo a sua topologia, a estrutura de componentes e conectores, as interfaces dos componentes etc.;
- Os dados acessados e atualizados dentro dos componentes e passados nas comunicações entre os componentes;
- Os estilos arquiteturais e atributos de qualidade identificados nas arquiteturas.

Dentro deste conjunto de critérios, os autores argumentam que os estilos arquiteturais e atributos de qualidade são fundamentais, uma vez que refletem decisões de projeto essenciais nos sistemas e que apresentam um grande impacto na sua estrutura. Em função disso, os estilos e atributos de qualidade deveriam variar o mínimo possível entre os sistemas candidatos. O'BRIEN e STOERMER (2001) afirmam ainda que os níveis de abstração abaixo do nível de componentes (e.g. o nível de implementação) não são adequados para uma comparação, uma vez que em sistemas distintos diferentes convenções de nomenclatura podem ser utilizadas e, possivelmente, diferentes formas de organização dos elementos do código são adotadas.

Após a comparação, um relatório das semelhanças e diferenças entre os sistemas é apresentado, demonstrando o potencial de migração das aplicações para uma Linha de

Produtos. Para facilitar a obtenção deste mapa de semelhanças e diferenças, O'BRIEN e STOERMER (2001) argumentam que o nível de granularidade dos componentes recuperados é muito importante. Os componentes têm que permitir a identificação das semelhanças e diferenças, escondendo detalhes de implementação. Componentes muito grandes podem esconder detalhes das variações entre os sistemas. Por outro lado, um grande número de componentes pequenos pode dificultar a identificação das semelhanças entre os sistemas.

Estabelecendo um paralelo entre a abordagem proposta nesta monografia com a de O'BRIEN e STOERMER (2001), percebe-se que o objetivo nesta proposta também é realizar uma comparação entre as arquiteturas recuperadas dos sistemas legados para apoiar a construção de uma arquitetura de referência para um domínio. Porém, os critérios definidos por O'BRIEN e STOERMER (2001) para comparar as arquiteturas, embora significativos, estão em um nível de abstração alto e não contemplam técnicas de suporte à sua automatização. Neste trabalho, serão definidos critérios mais formais e em um maior nível de detalhe, além de técnicas para a remontagem e comparação das arquiteturas, visando formalizar e automatizar estas atividades

No trabalho descrito nesta monografia, os critérios adotados para a comparação das arquiteturas incluem: a estrutura de componentes e conectores dos sistemas, em diferentes níveis de descrição arquitetural (pacotes e dependências nos primeiros níveis e classes e seus relacionamentos em níveis mais baixos); as interfaces dos componentes, levando em conta a sua semântica ou serviços oferecidos; os padrões arquiteturais identificados, relacionados aos atributos de qualidade privilegiados; e as funcionalidades especificadas para as aplicações através dos casos de uso, considerando, nesta comparação, as ligações entre os casos de uso e os elementos arquiteturais (ou componentes).

Com relação aos critérios para a comparação, convém ressaltar que as interfaces dos componentes podem ser derivadas dos métodos públicos declarados nas classes internas dos componentes (pacotes). Porém, métodos em diferentes sistemas possuirão diferentes assinaturas e a comparação das interfaces dos componentes nas diferentes arquiteturas pode ser inviável. Dessa forma, se o desenvolvedor realmente desejar utilizar as interfaces como um critério de comparação, ele precisará, primeiramente, mapear os métodos para serviços

em um nível mais alto de abstração. A ligação dos casos de uso com os componentes arquiteturais pode ajudar neste mapeamento.

Para automatizar a comparação, o padrão XML deve ser adotado na descrição das arquiteturas dos sistemas legados. Diversos algoritmos para computar automaticamente as diferenças entre dois arquivos XML (algoritmos de **diff**), que sigam o mesmo esquema, vêm sendo propostos na literatura (COBÉNA et al., 2002) (WANG et al., 2003). Através da linguagem de descrição arquitetural xADL (veja seção 2.3.2), que deve ser investigada neste trabalho, as arquiteturas poderão ser descritas em XML, propiciando a utilização (possivelmente com extensões) de um dos algoritmos de **diff** disponíveis para realizar as comparações. Além da xADL, outras abordagens possíveis para a descrição da arquitetura em XML serão investigadas, como a geração de arquivos XMI[16] para os modelos arquiteturais em UML, por exemplo.

Os algoritmos de **diff,** em geral, tomam um dos modelos (ou arquivo) como base para a comparação e cada elemento presente neste modelo é buscado no segundo modelo. Uma vez que o elemento seja encontrado, comparações de valores de dados e atributos são realizadas. Estes algoritmos se baseiam, normalmente, em comparação de árvores, sendo que no caso deste trabalho, cada sub-árvore em um arquivo XML corresponderia a um elemento arquitetural e suas sub-árvores à estrutura interna do elemento arquitetural.

Porém, um problema que se apresenta à utilização desta técnica é que os algoritmos de **diff** geralmente são utilizados para comparar duas versões de um mesmo documento (ou modelo). No caso da recuperação da arquitetura, pretende-se comparar modelos de sistemas diferentes, o que pode dificultar muito a aplicação de um algoritmo deste tipo. Se a equipe que estiver recuperando a arquitetura for a mesma, ela deve procurar padronizar a nomenclatura utilizada nos elementos arquiteturais e conectores recuperados. Caso contrário, a comparação ficará dificultada.

[16] XMI é um padrão de representação de informação, baseado em XML, que visa a descrição padronizada de modelos, cujo metamodelo siga o padrão MOF (*Meta Object Facility* – padrão OMG para metamodelos). Devido à padronização estabelecida pelo XMI, através dos seus DTD's, este formato de arquivo é utilizado para troca de informações entre ferramentas de modelagem que utilizem metamodelos baseados no MOF. UML é uma das linguagens de modelagem cujo metamodelo segue o padrão MOF e cujos modelos podem ser representados através de arquivos XMI (OMG, 2002).

Para apoiar a solução deste problema, será investigada a possibilidade de utilização de um modelo de Ontologia[17] do domínio. A utilização de um modelo de Ontologia pode apoiar a comparação da arquitetura em diferentes níveis de abstração.

5.5 Considerações Finais

A proposta apresentada está focada nas seguintes questões de pesquisa:

1. **Análise dinâmica de software para a reconstrução da arquitetura;**
2. **Estabelecimento de critérios de *clustering* para a remontagem de elementos arquiteturais;**
3. **Métricas de acoplamento e coesão para sistemas OO**;
4. **Detecção de padrões arquiteturais com base na análise dinâmica do software**;
5. **xADL para a descrição da arquitetura**;
6. **Geração de diff entre arquivos XML**;
7. **Utilização de ontologias para a unificação dos termos em um domínio.**

O objetivo desta proposta é a geração de artefatos para a Engenharia de Domínio. Além da arquitetura de referência, funcionalidades e componentes para o domínio podem ser gerados.

As funcionalidades são identificadas a partir dos casos de uso especificados para os sistemas legados e dos subsistemas identificados ao longo do processo de recuperação da arquitetura. Os subsistemas da aplicação resultam do agrupamento de casos de uso e representam funcionalidades da aplicação em um nível mais alto de abstração ou generalidade do que os casos de uso. Estes subsistemas podem indicar características do sistema legado, e, conseqüentemente, características do domínio. As características representam um artefato originado a partir do método de análise de domínio FODA (*Feature-Oriented Domain Analysis*) de KANG et al. (1990), que têm o objetivo de permitir a identificação de aspectos (funcionalidades, conceitos etc.) comuns e variáveis em

[17] O emprego de ontologias visa organizar o vocabulário de uma área, definindo a semântica dos termos e estabelecendo os seus relacionamentos. Através das ontologias, é possível verificar os termos que são sinônimos no domínio, homônimos etc.

uma família de aplicações similares ou domínio. Neste trabalho, assume-se que uma característica funcional da aplicação está em um nível de granularidade maior que um caso de uso e que pode ser representada pelo agrupamento de casos de uso ou subsistemas. Comparando-se as características identificadas para os diferentes sistemas legados em um domínio, é possível definir características obrigatórias, opcionais ou variáveis no domínio (classificação estabelecida pelo método FODA). As características obrigatórias são aquelas que constam da maioria dos sistemas legados; as opcionais são aquelas que constam de um ou poucos sistemas; e as variáveis são aquelas que aparecem em diversos sistemas, mas que apresentam diferenças no seu comportamento entre os diferentes sistemas.

A identificação e extração de componentes de sistemas legados, por sua vez, atividade possível de ser executada após a recuperação da arquitetura, representa um tema bastante explorado na literatura. CALDIERA e BASILI (1991) possuem um trabalho reconhecido nesta área, no qual eles ressaltam as características que devem ser avaliadas em componentes candidatos à reutilização. Dentre as características, os autores incluem: os custos de extração, empacotamento e integração do componente em um novo sistema; a utilidade do componente, a qual pode ser medida considerando-se a sua taxa de uso na aplicação, no domínio ou em aplicações em diferentes domínios; a corretude, facilidade de modificação e desempenho do componente. Para se obter valores para estas medidas, CALDIERA e BASILI (1991) propõem a utilização de métricas de complexidade, volume, regularidade e a freqüência de reuso. Eles acreditam que quanto menor, mais simples e com maior taxa de utilização o componente, maior a sua probabilidade de ser reutilizável.

O'BRIEN e SMITH (2002) consideram um outro conjunto de critérios para a extração de componentes dos sistemas legados, como: a adequação dos componentes à arquitetura de referência do domínio; os atributos de qualidade que os componentes satisfazem; o seu nível de acoplamento e coesão e, conseqüentemente, o esforço de reengenharia requerido para a sua extração e empacotamento; o custo de extração do componente versus o custo de desenvolvimento de um novo componente etc.

Nesta proposta, o objetivo não é propriamente suportar a extração de componentes dos sistemas legados, mas oferecer subsídios a uma atividade de extração posterior à atividade de recuperação da arquitetura. Como subsídios à extração de componentes, o trabalho proposto oferece as métricas de coesão e acoplamento sobre a arquitetura

recuperada, e, possivelmente, valores da taxa de utilização do componente na aplicação e no domínio.

Uma outra questão que se pretende investigar neste trabalho, além das questões de pesquisa mencionadas no início desta seção, é a comparação dos resultados obtidos com a Detecção de Padrões de Interação e a Técnica de Análise de Conceito Formal na reconstrução dos elementos arquiteturais. Ambas as técnicas podem ser utilizadas para indicar o compartilhamento de objetos entre as funcionalidades do sistema.

A técnica de análise de conceito formal, apresentada na seção 3.3.4, prevê a identificação de grupos de objetos que apresentam atributos compartilhados. A técnica é aplicada em um trabalho de recuperação de arquitetura para identificar métodos que são compartilhados por grupos de casos de uso (veja seção 4.4.1). Este compartilhamento é o critério para o agrupamento dos casos de uso e das classes em subsistemas na arquitetura.

Um dos objetivos da detecção de padrões de interação no trabalho descrito nesta monografia é encontrar áreas de implementação compartilhadas entre os casos de uso do sistema. A técnica de análise de conceito formal pode ser aplicada para este mesmo fim, embora ela gere uma hierarquia de conceitos com muitos níveis de aninhamento e redundâncias em seu resultado final, dificultando a tomada de decisão do desenvolvedor acerca dos elementos arquiteturais. De qualquer forma, os resultados gerados pelas duas técnicas e a sua contribuição para a tomada de decisão sobre a reconstrução dos elementos arquiteturais devem ser avaliados neste trabalho.

Este capítulo apresentou as técnicas e atividades que compõem a abordagem proposta para a recuperação e comparação das arquiteturas. No próximo capítulo, o suporte ferramental de apoio à abordagem é apresentado. Um dos objetivos deste trabalho é automatizar as atividades de recuperação de arquitetura de software e a sua comparação, através de critérios e técnicas propostos, e do desenvolvimento de ferramental de apoio. Dessa forma, a infra-estrutura apresentada no capítulo 6 é parte essencial desta proposta.

6. Infra-Estrutura de Suporte

Este capítulo descreve a infra-estrutura de suporte à abordagem descrita no capítulo 5. Ela terá como base o ambiente Odyssey (WERNER et al., 2004), o qual, conforme já mencionado, engloba tanto a Engenharia de Domínio quanto a Engenharia de Aplicação, sendo adequado à execução do processo proposto. Além do ambiente Odyssey, ferramentas isoladas e integradas ao Odyssey serão desenvolvidas.

O capítulo está organizado da seguinte forma: seção 6.1 apresenta uma visão geral da infra-estrutura e seção 6.2 descreve algumas ferramentas que já se encontram disponíveis.

6.1 Visão Geral da Infra-Estrutura

As avaliações da abordagem proposta serão realizadas no ambiente Odyssey (WERNER et al., 2004), o qual representa uma infra-estrutura de reutilização baseada em modelos de domínio. O Odyssey foi o ambiente escolhido para as avaliações por englobar tanto a Engenharia de Domínio (ED) quanto a Engenharia de Aplicação (EA). O processo proposto deverá ser executado no contexto da EA e os artefatos gerados para o domínio deverão ser armazenados nos repositórios do ambiente Odyssey para serem utilizados nas atividades de ED. Depois de incorporados aos modelos do domínio, os artefatos gerados pela abordagem proposta poderão ser reutilizados no desenvolvimento de novas aplicações no domínio.

Além de contemplar a Engenharia de Domínio, o Odyssey apresenta uma série de ferramentas e características que podem apoiar a execução das atividades do processo proposto. Em termos de ferramentas, aquelas que se destacam no apoio ao processo de recuperação de arquitetura são:

- A máquina de processos Charon (MURTA, 2002), na qual o processo de recuperação de arquitetura pode ser modelado, planejado, instanciado e acompanhado.
- A ferramenta de engenharia reversa estática Ares (VERONESE e NETTO, 2001), a qual é capaz de extrair um diagrama estrutural da UML a partir de código Java.

- A ferramenta de detecção de padrões de projeto (CORREA, 1999), que pode ser utilizada na etapa de mapeamento de padrões para a arquitetura, descrita na seção 5.2.6.
- As ferramentas de modelagem, que envolvem tanto os modelos da UML, necessários para suportar diversas etapas do processo, quanto o modelo de características do método FODA (*Feature-Oriented Domain Analysis*) de KANG et al. (1990). Conforme mencionado na seção 5.5, a abordagem proposta visa, além da geração de uma arquitetura de referência para o domínio, a identificação de características do domínio. Neste sentido, a utilização de um ambiente que aborde também este tipo de modelo é importante.

Além das ferramentas disponibilizadas pelo Odyssey, alguns protótipos vêm sendo desenvolvidos no contexto desta proposta, visando suportar funcionalidades não oferecidas pelo ambiente. A figura 19 ilustra um esquema de integração entre as ferramentas do Odyssey e os protótipos que vêm sendo desenvolvidos. Esta integração deve se dar, prioritariamente, através do compartilhamento de arquivos ou através de ferramentas intermediárias de integração.

Conforme representado na figura 19, a primeira atividade de recuperação de arquitetura propriamente é a modelagem de casos de uso. Em paralelo à modelagem de casos de uso, o modelo estrutural do sistema pode ser extraído através da ferramenta de engenharia reversa estática Ares. A terceira atividade do processo é a engenharia reversa dinâmica. Ela pode ser realizada com o apoio de um conjunto de ferramentas, incluindo um coletor de rastros de execução (o Tracer) e um Extrator de Diagrama de Seqüência.

Após a coleta dos rastros de execução pelo Tracer, a análise dos eventos coletados deve ser realizada pelo Detector de Padrões de Interação, ferramenta esta ainda não desenvolvida. A extração dos diagramas de seqüência, última tarefa da engenharia reversa dinâmica, é realizada pelo módulo Extrator de Diagramas de Seqüência. Ele lê os rastros de execução coletados pelo Tracer e os padrões de interação detectados pelo Detector de Padrões de Interação e gera os diagramas de seqüência no Odyssey. Cada diagrama de seqüência deve estar associado a um cenário de caso de uso no ambiente.

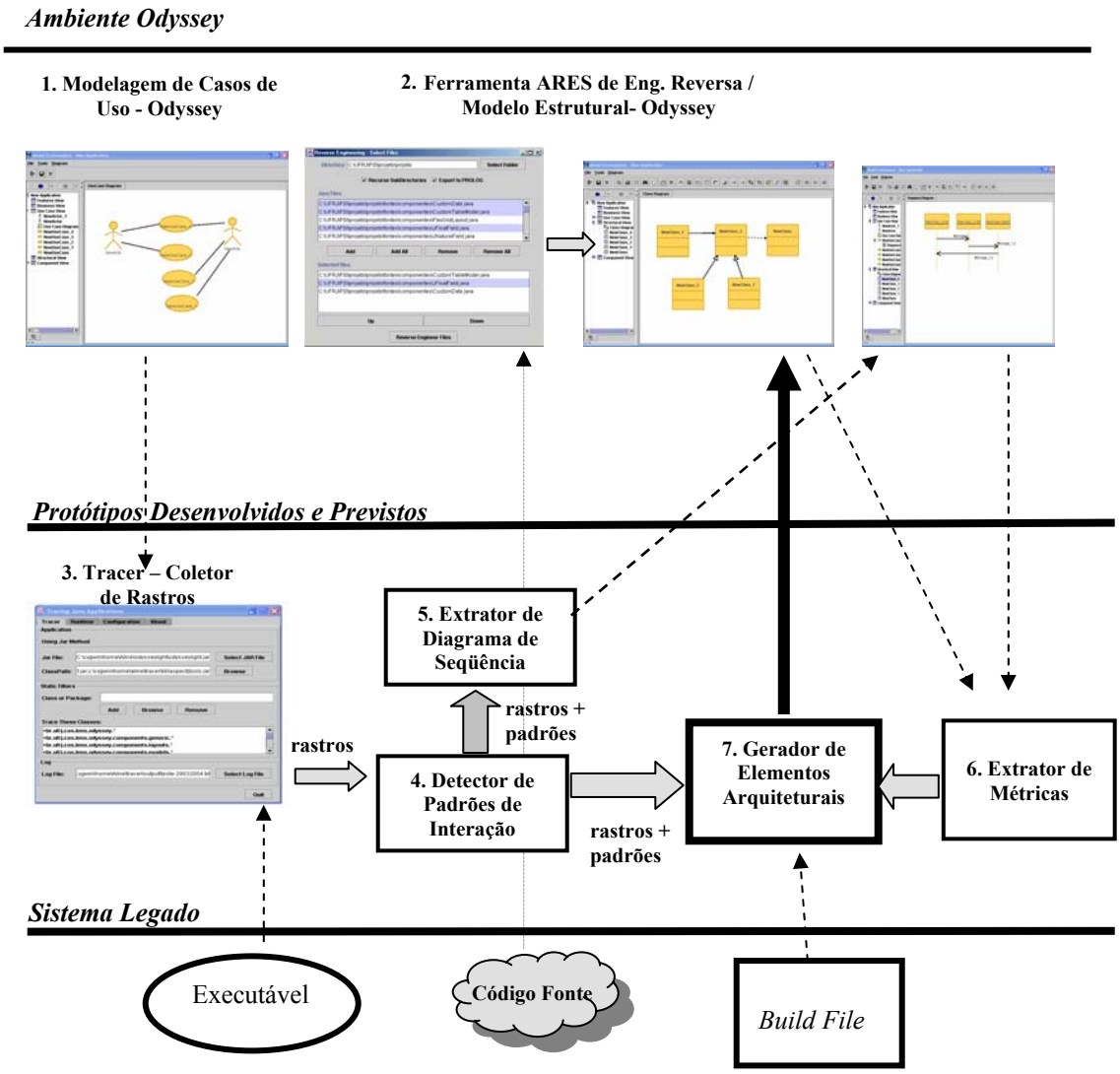

Figura 19. Esquema de Ferramentas Utilizadas para a Execução do Processo Proposto.

A definição dos elementos arquiteturais, atividade posterior à engenharia reversa dinâmica, deve ser realizada, prioritariamente, pelo módulo Gerador de Elementos Arquiteturais, a ser desenvolvido. Ele recebe como entradas os rastros de execução coletados pelo Tracer, associados a casos de uso, os padrões de interação detectados nos rastros e as métricas extraídas pelo módulo Extrator de Métricas, e gera, como saída, uma sugestão de elementos arquiteturais para a aplicação, os quais podem ser mapeados para o modelo estrutural do Odyssey. Além dessas informações de entrada, o módulo Gerador de

Elementos Arquiteturais pode ainda fazer uso das informações do *Build File* da aplicação, identificando bibliotecas externas que devem ser representadas na arquitetura.

Para apoiar a definição dos elementos arquiteturais, o desenvolvedor pode também fazer uso de um módulo de Manipulação de Classes (veja seção 6.4), módulo este cujo protótipo inicial jé se encontra disponível.

Além das ferramentas ilustradas na figura 19, a Charon (MURTA, 2002), máquina de processos do ambiente Odyssey, pode ser utilizada para suportar parcialmente o planejamento e o acompanhamento do processo. Na Charon, o gerente (ou desenvolvedor responsável pela execução do processo) define as atividades do processo, os papéis relacionados a cada atividade e os desenvolvedores envolvidos, o tempo previsto para a sua execução e os artefatos de entrada e saída. A máquina Charon (MURTA, 2002) oferece suporte parcial ao planejamento do processo porque ela apresenta algumas limitações, como: a falta de um mecanismo de apoio às estimativas de esforço, prazo e custo; a falta de um recurso para a elaboração de cronogramas; a impossibilidade de definição da duração da atividade para uma instanciação específica do processo.

A tabela 7 apresenta um resumo das atividades do processo, com o suporte ferramental previsto, nível de dependência com o Odyssey e o estágio atual de cada ferramenta.

Atividades	Ferramentas de Suporte	Dependência com o Odyssey	Estágio Atual
1. Planejamento do Processo	Charon – Suporte Parcial.	Sim.	Disponível.
2. Modelagem Casos de Uso	Ferramenta de Modelagem do Odyssey.	Sim.	Disponível.
3. Eng.Reversa Estática	Ferramenta Ares.	Sim.	Disponível.
4. Eng. Reversa Dinâmica			
4.1 Coleta Rastros	Tracer.	Não.	Protótipo disponível.
4.2 Análise de Rastros	Detector de Padrões de Interação.	Não.	A ser desenvolvida.

Tabela 7. Atividades do Processo Proposto e Ferramentas de Suporte.

Atividades	Ferramentas de Suporte	Dependência com o Odyssey	Estágio Atual
4.3 Extração Diagrama Seq.	Extrator de Diagrama de Seqüência.	Sim.	Protótipo disponível.
5. Definição de Elementos Arquiteturais			
5.1 Definição Semi-Automática	Gerador de Elementos Arquiteturais.	Parcial.	A ser desenvolvida.
5.2 Manipulação de Classes	Ferramenta de Manipulação de Classes.	Sim.	Protótipo disponível.
6. Mapeamento de Padrões	Ferramenta de Detecção de Padrões de Projeto e Ferramenta de Detecção de Padrões Arquiteturais.	Detecção de Padrões de Projeto - Sim. Detecção de Padrões Arquiteturais - Parcial.	Detecção de Padrões de Projeto – disponível. Detecção de Padrões Arquiteturais – a ser desenvolvida.
7. Avaliação da Arquitetura	Extrator de Métricas.	Sim.	A ser desenvolvida.

Tabela 7. Atividades do Processo e Ferramentas de Suporte (continuação).

As ferramentas dependentes do Odyssey são aquelas que por necessitarem conhecer os elementos do modelo semântico da aplicação, dependem do Odyssey para compilar e para serem executadas. As ferramentas com pouca ou nenhuma dependência para o modelo semântico do Odyssey são desenvolvidas de forma independente ao ambiente e os seus resultados são mapeados para o Odyssey, conforme necessário. Aquelas com dependência parcial são as ferramentas que executam de forma independente ao Odyssey, mas cujos resultados devem ser mapeados para o modelo semântico do Odyssey. Normalmente, este mapeamento vai se dar através de um módulo intermediário de mapeamento.

O ferramental de suporte necessário à segunda parte da abordagem proposta (veja seção 5.4), ou seja, o ferramental necessário para a comparação das arquiteturas recuperadas, ainda precisa ser planejado e desenvolvido. Será considerada, nesta etapa, a utilização e extensão, se possível, de ferramental já existente. Porém, para a implementação da segunda parte da abordagem, a integração da linguagem xADL com o Odyssey ou a geração de uma descrição da arquitetura em XML é essencial.

Apresenta-se, nas seções a seguir, a descrição detalhada de algumas ferramentas que já se encontram disponíveis.

6.2 Ferramenta de Engenharia Reversa Estática

Para a realização da engenharia reversa estática, deve ser utilizada a ferramenta de engenharia reversa Ares (VERONESE e NETTO, 2001), disponível no Odyssey. A Ares é capaz de recuperar o modelo estrutural da UML para uma aplicação implementada na linguagem de programação Java. Esta ferramenta apresenta algumas propriedades interessantes, as quais motivaram a sua escolha (além do fato de estar integrada ao Odyssey), como: a capacidade de exportação do modelo para Prolog (característica opcional) e a extração de um modelo estrutural mais completo do sistema do que os modelos extraídos por outras ferramentas comumente utilizadas para a engenharia reversa estática.

A exportação do modelo para Prolog é importante porque a ferramenta de detecção de padrões de projeto (DANTAS et al., 2002) do ambiente Odyssey, que também será utilizada nesta proposta, requer um modelo do sistema em Prolog para detectar os padrões.

Com relação ao modelo extraído, a Ares é capaz de detectar um conjunto maior de dependências entre os elementos do código fonte do que as ferramentas de engenharia reversa estática comumente utilizadas (veja seção 3.3.1.1). Ela recupera um modelo estrutural de classes mais completo do sistema, oferecendo um maior suporte à recuperação da arquitetura. Dentre as dependências recuperadas, estão aquelas obtidas diretamente do código Java, como as dependências resultantes de herança entre classes e de tipos de atributos declarados nas classes, e aquelas obtidas de uma forma indireta, como:

a) Dependências resultantes de tipos de parâmetro recebidos por um método, o que determina uma dependência da classe que possui o método para a classe que representa o tipo do parâmetro.
b) Dependências por tipos de retorno de métodos, o que determina uma dependência da classe que chama o método para a classe que representa o tipo retornado pelo método.
c) Instanciação de objetos, a qual determina uma dependência entre a classe instanciadora e a classe referente ao tipo instanciado.

d) Declaração de variáveis locais, a qual determina uma dependência da classe que possui o método, onde as variáveis são declaradas, para a classe que representa o tipo das variáveis.

e) Conversão forçada de tipos, a qual determina uma dependência da classe onde ocorre a conversão para a classe que representa o tipo resultante da conversão.

Além das dependências entre classes, a ferramenta recupera também as dependências entre os pacotes onde as classes se encontram. Se uma classe em um pacote A importa uma classe de outro pacote B, então o pacote A apresenta uma dependência de importação para o pacote B. Além disso, se uma classe em um pacote A referencia de forma qualificada uma classe do pacote B, o pacote A apresenta uma dependência de uso para B.

A Ares não é capaz de recuperar dependências entre classes resultantes do uso de coleções não-tipadas do Java. Por exemplo, se uma classe utiliza um *Vector* para guardar objetos e somente manipula determinado tipo de objeto através deste *Vector*, a Ares não será capaz de detectar a dependência entre esta classe e a classe relativa ao tipo de objeto manipulado na coleção. Uma possível solução para este problema deve ser investigada neste trabalho.

6.3 Ferramentas para a Engenharia Reversa Dinâmica

A coleta dos rastros de execução será realizada utilizando-se a ferramenta Tracer, desenvolvida no contexto do projeto Odyssey. Ela utiliza a tecnologia de aspectos, através da extensão AspectJ para a linguagem Java (ECLIPSE, 2003), para o monitoramento do programa e rastreamento das mensagens trocadas entre os objetos durante a execução. A vantagem do uso de aspectos está na possibilidade de monitorar a aplicação de uma forma não-intrusiva, ou seja, sem a alteração do código fonte original. A ferramenta coleta o objeto, seu tipo e método emissor da mensagem e o objeto, seu tipo e método receptor da mensagem, gravando as informações em um arquivo XML.

A ferramenta Tracer está ilustrada na figura 20. Ela solicita ao usuário que este informe o arquivo que representa o executável da aplicação (arquivo com extensão jar no Java, ou seja, o arquivo compactado contendo o código binário da aplicação), o caminho onde se

encontram as bibliotecas necessárias para executar o sistema (ClassPath), possivelmente alguns filtros e o arquivo de log, onde será gravada a saída da execução (ou seja, o conjunto de eventos coletados). O usuário pode informar ainda, para cada cenário de caso de uso que executa, uma etiqueta, indicando o cenário que está sendo executado no momento. O Tracer grava esta etiqueta no arquivo XML de saída, delimitando os eventos gerados por cenário de caso de uso.

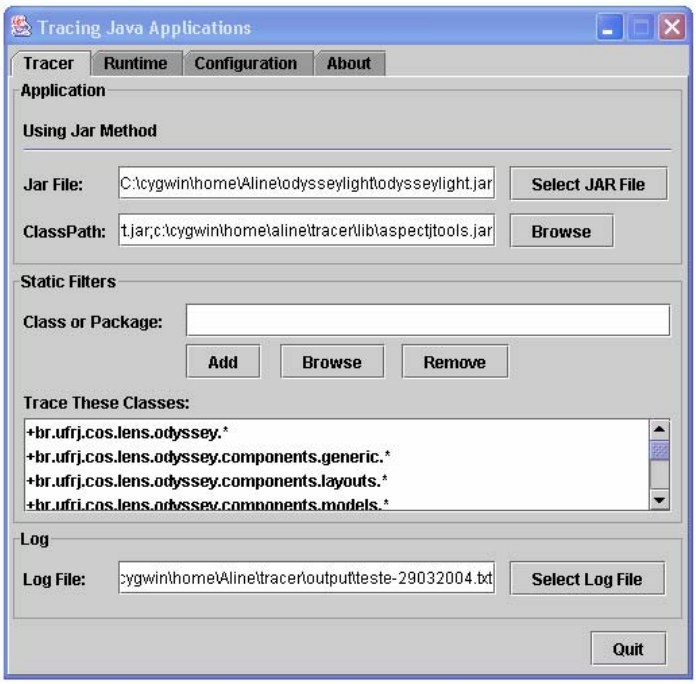

Figura 20. Ferramenta de Coleta de Rastros de Execução: Tracer.

O Tracer permite a informação de filtros estáticos, aplicados antes de se iniciar a execução da aplicação, e dinâmicos, aplicados durante a execução da aplicação. Os filtros estáticos permitem a seleção dos pacotes e classes que serão monitorados. Os dinâmicos permitem a exclusão de algumas classes monitoradas em tempo de execução e a seleção de um método origem, a partir do qual a coleta é executada.

Conforme já mencionado, a ferramenta Tracer grava os rastros de execução em arquivos XML. XML representa um formato padrão de arquivo texto para a descrição de dados de forma seqüencial ou estruturada. Cada arquivo XML possui um DTD (descritor dos dados do documento) para a descrição do seu esquema. Através do DTD, é possível compreender

as *tags* (ou rótulos) do arquivo. A ferramenta *Tracer* gera arquivos XML, cujo DTD define *tags* para as mensagens trocadas entre os objetos.

```
<trace label="Criar Domínio">
        <message id="1" sequence="1">
                <source file="<Unknown>" line="0"/>
                <timeinfo startTime="28 de Março de 2004 15h45min12s BRT"/>
                <caller class="odyssey.model.Dominio" method="[init]"
                        instance="@1699c75"/>
                <callee class="odyssey.components.generic.TransferableTreeNode"
                        method="[init]" instance="@9efc6a"/>
        </message>
        <message id="2" sequence="2">
                <source file="<Unknown>" line="0"/>
                <timeinfo startTime="28 de Março de 2004 15h45min12s BRT"/>
                <caller class="odyssey.model.Dominio" method="[init]"
                        instance="@1699c75"/>
                <callee class="odyssey.components.generic.TransferableTreeNode"
                        method="[init]" instance="@9efc6a"/>
        </message>
```

Figura 21. Trecho de Rastro de Eventos Extraído pela Ferramenta Tracer.

Um exemplo de rastro coletado pela ferramenta e armazenado no arquivo XML está ilustrado na figura 21. Neste rastro, o cenário de caso de uso executado é a criação de um domínio no ambiente Odyssey. Daí o rótulo "Criar Domínio" no *trace*. A aplicação monitorada foi o próprio Odyssey.

O objetivo da gravação dos rastros de eventos em XML é permitir o processamento automático dos mesmos e dar flexibilidade à análise dos *traces*, permitindo a sua leitura por diferentes ferramentas.

A ferramenta Tracer apresenta algumas características importantes para este trabalho, as quais justificam o seu desenvolvimento, diferenciando-a das demais ferramentas apresentadas na seção 3.3.1.2. Dentre elas, pode-se citar: a geração de um arquivo XML dos rastros, facilitando o seu processamento automático; a possibilidade de habilitar e desabilitar a coleta, permitindo a coleta de rastros para um cenário de caso de uso específico; a possibilidade de informação de filtros a serem aplicados na coleta; uma interface gráfica amigável e flexível; a capacidade de configuração, permitindo o monitoramento de qualquer aplicação Java.

6.4 Ferramenta para a Manipulação de Classes

Para apoiar a manipulação do modelo pelo desenvolvedor, tarefa esta necessária na definição dos elementos arquiteturais (veja seção 5.2.5.2), uma ferramenta de seleção e manipulação de classes foi desenvolvida no Odyssey: a *Class Manipulation Tool*. Ela permite que o desenvolvedor consulte o modelo estrutural do ambiente e selecione um conjunto de classes que devem ser agrupadas em um novo elemento arquitetural. A ferramenta está ilustrada na figura 22.

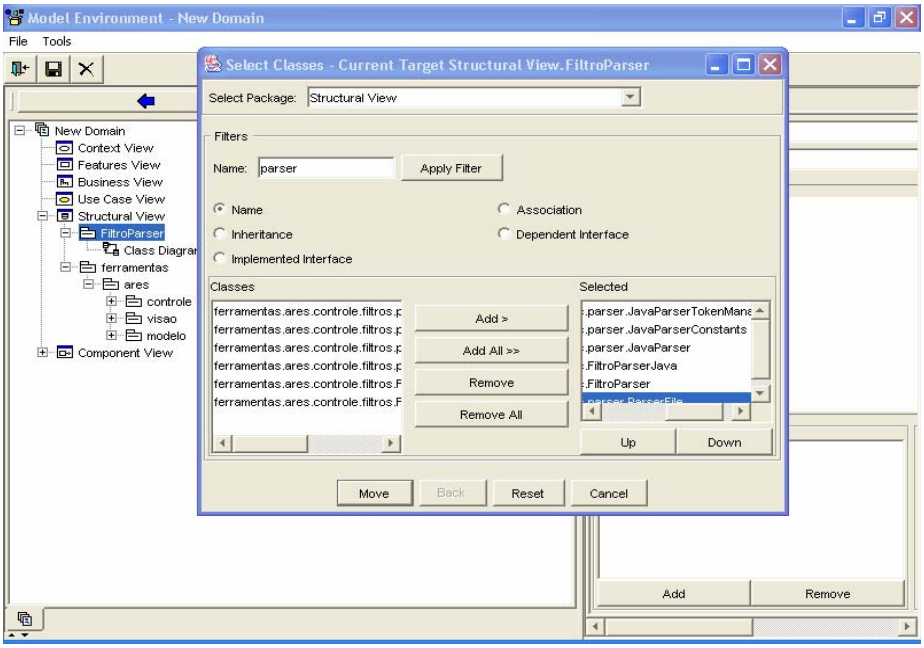

Figura 22. Ferramenta de Seleção e Manipulação de Classes.

O desenvolvedor seleciona o pacote raiz onde a busca de classes deve ser iniciada. Na figura 22, a busca é feita em todo o modelo estrutural do Odyssey, pois o pacote raiz selecionado é a própria *Strucutural View* (ou modelo estrutural).

A seguir, ele pode aplicar um critério de filtro para selecionar um conjunto de classes na hierarquia de pacotes informada. Até o momento, foram previstos os seguintes critérios de filtro na ferramenta: *Name* - uma *string* presente nos nomes das classes; *Inheritance* - a superclasse de uma hierarquia, levando à seleção das subclasses em todos os níveis da hierarquia; *Implemented Interface* - uma interface implementada, levando à seleção de todas as classes que implementam esta interface; *Association* – leva à seleção de

todas as classes associadas a uma classe especificada; *Dependent Interface* - uma interface dependente, levando à seleção de todas as classes que dependem desta. Depois de aplicado o filtro, as classes resultantes aparecem na caixa de listagem do lado esquerdo da tela. A partir deste conjunto de classes, o desenvolvedor pode ainda estreitar a seleção, movimentando as classes desejadas para a caixa de listagem que aparece do lado direito. O resultado final da aplicação da *query* é a movimentação das classes para o novo elemento arquitetural. No caso da figura 22, o elemento arquitetural é FiltroParser (cujo nome aparece no título da janela da ferramenta).

Uma característica interessante da ferramenta é a geração automática de dependências entre os pacotes ou elementos arquiteturais. A cada movimentação de classes, uma nova configuração de dependências se apresenta entre os elementos arquiteturais.

7. Considerações Finais

A fim de verificar a aplicabilidade prática da abordagem proposta, tanto em nível do processo proposto quanto em nível do ferramental de suporte, duas avaliações devem ser conduzidas: uma sobre o processo de recuperação da arquitetura proposto e uma sobre os critérios e técnicas para a geração de artefatos para a Engenharia de Domínio. As avaliações sobre o processo de recuperação de arquitetura e sobre os critérios e técnicas para a comparação das arquiteturas serão realizadas através de estudos de caso, envolvendo sistemas em um domínio selecionado. Deverão ser selecionados pelo menos três sistemas legados em um mesmo domínio. Os objetivos das avaliações envolvem a verificação da qualidade dos resultados gerados pela abordagem proposta e o apoio ao refinamento da proposta. Os resultados das avaliações, a cada ciclo de execução do processo de recuperação de arquitetura, serão utilizados como retro-alimentadores do próprio processo, servindo de base para o seu ajuste, aprimoramento e detalhamento.

As avaliações devem ser realizadas na infra-estrutura proposta no capítulo 6, sendo que, neste caso, a própria infra-estrutura também estará sendo testada e aprimorada ao longo da execução do processo.

Além das avaliações, o capítulo apresenta as contribuições esperadas com a pesquisa, o planejamento do trabalho e o cronograma de atividades previstas.

O capítulo está organizado da seguinte forma: seção 7.1 apresenta as avaliações que devem ser conduzidas; seção 7.2 apresenta as contribuições esperadas com esta pesquisa; e seção 7.3 finaliza o capítulo, apresentando as atividades necessárias para a realização da pesquisa, dispostas em um cronograma proposto

7.1 Avaliações

7.1.1 Avaliação do Processo de Recuperação de Arquitetura de Software

A fim de avaliar o processo de recuperação da arquitetura proposto, o mesmo deve ser executado para sistema reais. O objetivo da avaliação é verificar se a arquitetura recuperada reflete de fato uma possível organização arquitetural para o software.

WOHLIN et al. (2000) destacam três tipos de avaliações que podem ser utilizados para a verificação de uma nova abordagem na engenharia de software: *survey* (e.g.

entrevista, questionário), estudo de caso e experimento. A seleção de uma dentre estas três estratégias deve levar em conta o propósito da avaliação, as condições ambientais e o tipo de objeto a ser avaliado (técnica, ferramenta, processo ou método). Para a avaliação do processo proposto, serão utilizados estudos de caso e questionários.

Os estudos de caso envolverão pelo menos três sistemas em um mesmo domínio de aplicação. Consideramos, neste trabalho, uma dentre quatro possibilidades: a utilização de ambientes de engenharia de software, cujo código encontra-se disponível no próprio PESC (Programa de Engenharia de Sistemas e Computação), na COPPE UFRJ; a utilização de sistemas acadêmicos da UFRJ e de outras Universidades; a utilização de sistemas de código aberto, cujo código encontra-se disponível na Internet; a realização de experimento "em campo", em ambiente industrial. Os ambientes de engenharia de software do PESC ainda possuem especialistas disponíveis. Os sistemas de código aberto possuem, em geral, uma documentação de referência disponível**.** Em relação aos sistemas legados em ambiente acadêmico ou industrial, é necessário verificar a disponibilidade de alguma fonte de informação sobre os mesmos.

A avaliação do processo de recuperação de arquitetura deve ser feita com base em métricas sobre o produto final obtido, ou seja, a arquitetura reconstruída do sistema legado. As medidas a serem empregadas, mencionadas na seção 5.2.7, incluem métricas de coesão e acoplamento e métricas de cobertura de código. Além das métricas, a arquitetura recuperada deve ser avaliada por especialistas da aplicação, os quais irão julgar o quanto esta reflete de fato os conceitos e funcionalidades do sistema legado e o quanto ela apóia a compreensão do software. Para o julgamento dos especialistas, um questionário será elaborado, visando padronizar a avaliação.

7.1.2 Avaliação dos Critérios e Técnicas para a Geração de Artefatos para a Engenharia de Domínio

Na abordagem proposta, são apresentados critérios e técnicas para a geração de artefatos para a Engenharia de Domínio (ED), com base nos artefatos recuperados dos sistemas legados (veja seções 5.4 e 5.5). Para avaliar este conjunto de critérios e técnicas, as arquiteturas de pelo menos três sistemas legados em um mesmo domínio de aplicação deverão ser recuperadas. Os artefatos produzidos para a ED serão armazenados em um

repositório para posterior reutilização. Um especialista ou engenheiro do domínio deverá avaliar os artefatos gerados para a ED e verificar a sua adequação ao domínio em questão. Um questionário também poderá ser utilizado nesta avaliação.

7.1.3 Considerações sobre as Avaliações

A fim de estabelecer uma comparação entre as diferentes execuções do processo, as variáveis independentes (como o tamanho do sistema legado em linhas de código, a disponibilidade de fontes de informação sobre o sistema, o fato de o sistema possuir um projeto inicial etc.) que afetam as variáveis dependentes (esforço na execução do processo, qualidade da arquitetura recuperada etc.) devem ser medidas. Dessa forma, pode-se estabelecer, ao longo do tempo, uma linha de base (*baseline*) com os valores médios da execução do processo. Os valores da execução do processo incluem: o tamanho do sistema (em número de classes, linhas de código etc.), a disponibilidade de fontes de informação sobre o sistema legado, alguns atributos de qualidade do sistema, a duração (em horas) da execução do processo (por atividade), o número de desenvolvedores envolvidos, o esforço aplicado na execução do processo (por atividade), o custo (em dólares) etc. Com base nestes valores, o processo pode ser refinado e aperfeiçoado e estimativas futuras sobre a execução do processo, para outros projetos ou sistemas, podem ser realizadas.

É importante ressaltar que antes do início das avaliações, um planejamento e projeto dos estudos de caso deverão ser realizados.

Além das duas avaliações mencionadas, um pequeno estudo de caso para a comparação dos resultados obtidos com a detecção de padrões de interação e com a aplicação da técnica de análise de conceito formal, conforme mencionado na seção 5.5, deverá ser realizado. Este estudo de caso deverá se basear em um pequeno número de subsistemas de um dos sistemas legados selecionados para a recuperação da arquitetura.

7.2 Contribuições Esperadas

As contribuições esperadas nesta pesquisa envolvem:

- A definição de um processo de engenharia reversa para a recuperação da arquitetura de software apoiado na análise dinâmica do sistema. O processo proposto, juntamente com as técnicas empregadas e o ferramental de suporte, pretende oferecer um maior suporte à tomada de decisão na montagem dos elementos

arquiteturais, ser genérico do ponto de vista da aplicação ou do domínio e recuperar visões arquiteturais estáticas e dinâmicas.

- O apoio a processos de Engenharia de Domínio, através dos artefatos gerados para o domínio.
- A ampliação dos estilos e padrões arquiteturais que podem ser identificados em sistemas legados, com base na análise dinâmica do sistema.
- A utilização de critérios e técnicas mais formais para a comparação de arquiteturas.

Além dessas contribuições principais, algumas contribuições que também podem ser obtidas incluem:

- A integração de uma ADL ao ambiente de desenvolvimento Odyssey, permitindo a obtenção de uma descrição arquitetural mais completa e formal do software.
- A exploração de métricas dinâmicas para sistemas orientados a objetos, contribuindo com a proposição e avaliação deste tipo de métrica para software OO.
- O apoio à manutenção de software, através das ligações estabelecidas entre os cenários de casos de uso e os modelos dinâmicos extraídos do sistema.

7.3 Planejamento do Trabalho

Para a realização desta pesquisa, diversas atividades são necessárias. Elas envolvem o levantamento bibliográfico de trabalhos em áreas relacionadas, a elaboração e implementação da proposta, dentre outras. A lista das atividades é a seguinte:

1. **Cumprimento dos Créditos Obrigatórios**

   Foram cumpridas, no primeiro ano do curso, em 2002, as seis disciplinas necessárias à conclusão dos créditos exigidos para o Doutorado.

2. **Revisão da Literatura**

   A revisão da literatura foi realizada durante as disciplinas e ao longo do ano de 2003. Foram realizadas leituras nas áreas de Arquitetura de Software, Recuperação de Arquitetura de Software, Engenharia Reversa e Engenharia de Domínio. A revisão continuará ocorrendo durante todo o período da pesquisa para acompanhar os avanços

da área e para resgatar autores e trabalhos relacionados significativos que por ventura ainda não tenham sido mencionados.

3. Redação e Defesa do Exame de Qualificação

Esta atividade envolve a redação desta monografia e a defesa da proposta perante uma banca examinadora. O Exame foi elaborado ao longo do ano de 2003 e início do ano de 2004, para defesa em Junho de 2004.

4. Redação de Artigos

Ainda não foram publicados artigos referentes a este trabalho. Até o momento, foi produzido um relatório técnico publicado internamente na COPPE. A previsão é de dar encaminhamento a publicações significativas no decorrer dos dois próximos anos do curso, onde algumas conferências relacionadas ao tema do trabalho e selecionadas como meta, incluem: ICSM (*International Conference on Software Maintenance*), WCRE (*Working Conference on Reverse Engineering*), SBES (Simpósio Brasileiro de Engenharia de Software) e ICSE (*International Conference on Software Engineering*).

5. Desenvolvimento da Infra-Estrutura de Suporte

A infra-estrutura de suporte pode ser dividida em duas partes: infra-estrutura para apoiar a execução do processo de recuperação de arquitetura e infra-estrutura para a comparação das arquiteturas recuperadas e extração de artefatos para o domínio.

5.1 Infra-Estrutura de Suporte ao Processo de Recuperação de Arquitetura

Com relação à infra-estrutura apresentada na figura 19, durante o ano de 2003 foram desenvolvidos o Tracer (coletor de rastros de execução), o Extrator de Diagramas de Seqüência e o módulo de Manipulação de Classes. Para completar o ferramental de suporte à abordagem, além do aperfeiçoamento dos protótipos já desenvolvidos, os demais módulos deverão ser especificados e implementados, a saber: Extrator de Métricas de acoplamento estáticas e dinâmicas para sistemas OO, Detector de Padrões de Interação nos rastros coletados e Gerador de Elementos Arquiteturais.

5.2 Infra-Estrutura de Suporte à Extração de Artefatos para o Domínio

A fim de suportar a comparação das arquiteturas recuperadas e a geração de artefatos para o domínio, os seguintes módulos devem ser especificados e implementados: um módulo para integração da xADL com o Odyssey (ou de uma ferramenta para a descrição da arquitetura em XML); e um módulo de suporte à extração de **diff** entre arquivos XML que descrevem arquiteturas.

6. Avaliações

As avaliações previstas, brevemente apresentadas na seção 7.1, deverão ser planejadas e executadas assim que uma infra-estrutura mínima de suporte ao processo se encontre disponível.

7. Refinamento da Abordagem Proposta

O processo proposto deve ser refinado à medida que o ferramental de suporte seja desenvolvido e testado e à medida que os resultados das avaliações sejam obtidos. As atividades descritas no capítulo 5 devem ser reformuladas, se necessário, e descritas em um maior nível de detalhe.

8. Redação e Defesa da Tese

O texto da tese será composto principalmente dos resultados descritos nos artigos a serem publicados no decorrer do trabalho. Após a redação, a tese deverá ser defendida perante uma banca examinadora.

As atividades previstas encontram-se dispostas no cronograma a seguir:

Atividade/Período	2004				2005				2006			
	1º tri	2º tri	3º tri	4º tri	1º tri	2º tri	3º tri	4º tri	1º tri	2º tri	3º tri	4º tri
4												
5.1												
5.2												
6												
7												
8												

Tabela 8. Cronograma Previsto de Atividades para a Realização do Trabalho.

Referências Bibliográficas:

ALEXANDER, C., ISHIKAWA, S., SILVERSTEIN, M., LACOBSON, M., FIKSDAHL-KING, I., ANGEL, S., 1977, "A Pattern Language", *Oxford University Press,* Nova York.

ANQUETIL, N., LETHBRIDGE, T.C., 2003, "Comparative Study of Clustering Algorithms and Abstract Representations for Software Remodularisation", *In Software IEE Proceedings, vol. 150, issue 3, June,* pp. 185-201.

AOSD STEERING COMMITTEE, 2004, "Aspect-Oriented Software Development", disponível na Internet em http://aosd.net/, último acesso em 14/02/2004.

ARISHOLM, E., 2002, "Dynamic Coupling Measures for Object-Oriented Software", *Proceedings of the 8th IEEE Symposium on Software Metrics (Metrics'02),* Ottawa, Canadá, Junho, pp. 33-42.

BALL, T., EICK, S.G., 1994, "Visualizing Program Slices", *Proceedings of the 1994 IEEE Symposium on Visual Languages,* St. Louis, Missouri, pp.288-295.

BASILI, V., WEISS, D., 1984, "A Methodology for Collecting Valid Software Engineering Data", *IEEE Transactions on Software Engineering, 14 (6),* Junho, pp. 758-773.

BASS, L., CLEMENTS, P., KAZMAN, R., 1998, "Software Architecture in Practice", Addison-Wesley, Massachusetts.

BIGGERSTAFF, T. J., MITBANDER, B. G., WEBSTER, D. E., 1994, "Program Understanding and the Concept Assignment Problem", *ACM*, pp. 72-82, volume 37 nº 5.

BOJIC, D., VELASEVIC, D., 2000, "A Use-Case Driven Method of Architecture Recovery for Program Understanding and Reuse Reengineering", *Proceedings of the 4th European Software Maintenance and Reengineering Conference,* Zurique, Suíça, Fevereiro/Março, pp. 23-31.

BRAGA, R., 2000, "Busca e Recuperação de Componentes em Ambientes de Reutilização de Software", *Tese de Doutorado,* COPPE Sistemas, Rio de Janeiro, Brasil, Dezembro.

BURMEISTER, P., 1998, “Formal Concept Analysis with ConImp: Introduction to the Basic Features”, Arbeitsgruppe Allgemeine Algebra and Diskrete Mathematik, Technische Hochschule Darmstadt, Schloβgartenstr. 7, 64289, Darmstadt, Germany.

BUSCHMANN, F., MEUNIER, R., ROHNERT, H., 1996, “Pattern-Oriented Software Architecture: A System of Patterns.”, *1 ed. John Wiley & Sons.*

CALDIERA, G., BASILI, V.R., 1991, “Identifying and Qualifying Reusable Software Components”, *IEEE Computer, Vol. 24, Issue 2,* Fevereiro, pp. 61-70.

CHAN, S., LAMMERS, T., 1998, “OODE”, *5th International Conference on Software Reuse (ICSR-5) Tutorials,* Vitória, Canadá, Junho.

CHIDAMBER, S. R., KEMERER, C.F., 1994, “A Metrics Suite for Object-Oriented Design”, *IEEE Transactions on Software Engineering, Vol. 20, Nº 6,* Junho, pp. 476-493.

CHO, E. S., KIM, C. J., KIM, S. D., RHEW, S. Y., 1998, “Static and Dynamic Metrics for Effective Object Clustering”, *Proceedings of the Asia Pacific Software Engineering Conference*, Taipei, Taiwan, Dezembro, pp. 78-85.

CHUNG, L.K., NIXON, B., YU, E., 1994, “Using Quality Requirements to Drive Software Development”, *Workshop on Research Issues in the Intersection Between Software Engineering and Artificial Intelligence*, Sorrento, Itália, Maio, pp.16-17.

CLARKWARE CONSULTING INC., 2003, JDepend versão 2.7, disponível na Internet em http://www.clarkware.com/software/JDepend.html, último acesso em 10/04/2004.

CLEMENTS, P., 1994, "From Domain Models to Architecture", *USC Center for Software Engineering workshop on software architecture*, Los Angeles.

COBÉNA, G., ABITEBOUL, S., MARIAN, A., 2002, “Detecting Changes in XML Documents”, *Proceedings of the 18th International Conference on Data Engineering*, San Jose, CA USA, Fevereiro/Março, pp. 41-52.

CORREA, A. L., 1999, “Uma Arquitetura de Apoio para Análise de Modelos Orientados a Objetos”, Tese de M.Sc., COPPE/UFRJ, Rio de Janeiro, RJ, Brasil.

DANTAS, A.R., VERONESE, G.O., CORREA, A. L., XAVIER, J.R., WERNER, C. M. L., 2002, “Suporte a Padrões no Projeto de Software”, *XVI Simpósio Brasileiro de Engenharia de Software*, Gramado, Brasil, Outubro, pp. 450-455.

DASHOFY, E.M., HOEK, A. V., TAYLOR, R. N., 2001, "A Highly-Extensible, XML-Based Architecture Description Language", *In Proceedings of the Working IEEE/IFIP Conference on Software Architecture (WICSA 2001),* Amsterdam, Agosto, pp. 103-112.

DASHOFY, E.; HOEK, A.; TAYLOR, R., 2002, "An Infrastructure for the Rapid Development of XML-based Architecture Description Languages." *ICSE 2002. Proceedings of the 24th International Conference on Software Engineering*, Maio, pp. 266-276.

DEURSEN, A. V., 2001, "SWARM: Software Architecture Recovery and Modelling", *WCRE 2001 Discussion Forum Report.* Stuttgart, Alemanha, Outubro.

DI LUCCA, G. A., CANFORA, G., DI LUCIA, A., FASOLINO, A.R., 1994, "Recovering the Architectural Design for Software Comprehension". *IEEE, 1994. Proceedings of the 3 rd IEEE Workshop on Program Comprehension*, Washington, Estados Unidos, Novembro, pp. 30-38.

DING, L., MEDVIDOVIC, N., 2001, "Focus: A Light-Weight, Incremental Approach to Software Architecture Recovery and Evolution". *Working IEEE/IFIP Conference on Software Architecture (WICSA'01)*, Amsterdam, Holanda, Agosto, pp.191-200.

D'SOUZA, D., WILLS, A., 1999, "Objects, Components and Frameworks with UML", *Addison-Wesley*.

ECLIPSE ORG 2003a, AspectJ 1.1.1, disponível na Internet em http://eclipse.org/aspectj/, último acesso em 14/02/2004.

ECLIPSE ORG 2003b, Eclipse versão 2.1.3, disponível na Internet em http://www.eclipse.org/, último acesso em 10/04/2004.

EIXELSBERGER, W., OGRIS, M., GALL, H., BELLAY, B., 1998, "Software Architecture Recovery of a Program Family", *Proceedings of the International Conference on Software Engineering,* Kyoto, Japão, Abril, pp. 508-511.

EMAM, K.E., 2001, "A Primer on Object-Oriented Measurement", *Proceedings of 7th International Software Metrics Symposium,* London, UK, Abril, pp. 185-187.

FAYYAD, U. M., 1996, "Advances in Knowledge Discovery and Data Mining", *MIT Press,* Menlo Park, Califórnia, Estados Unidos.

GALL, H., JAZAYERI, M., KLÖSCH, R, 1996, "Architecture Recovery in ARES", *Proceedings of the International Software Architecture Workshop, ISAW96*, São Francisco, Califórnia, Estados Unidos, Outubro, pp. 111-115.

GALL, H., PINZGER, M., 2002, "Pattern-Supported Architecture Recovery*",* TU: Technical University of Vienna, Information Systems Institute, Distributed Systems Group, *In Proceedings of the 10th International Workshop on Program Comprehension*, Paris, França, pp. 53-61.

GAMMA, E., HELM, R., JOHNSON, R., VLISSIDES, J., 1995, "Design Patterns: Elements of Reusable Object-Oriented Software", *Addison-Wesley Longman, Inc.*

GANNOD, G., 1999, "Automated Reverse Engineering and Design Recovery*", 14th IEEE International Conference on Automated Software Engineering*, Flórida, Estados Unidos, Outubro, Tutorial 3.

GARLAN, D., PERRY, D., 1995, "Introduction to the Special Issue on Software Architecture", *IEEE Transactions on Software Engineering*, Abril, pp. 269-274.

GRISS, M., FAVARO, J., D'ALESSANDRO, M., 1998, "Integrating Feature Modeling with the RSEB", *In Proceedings of the 5th International Conference on Software Reuse*, Vitória, Canadá, Junho, pp. 76-85.

GUO, G., ATLEE, J., KAZMAN, R., 1999, "A Software Architecture Reconstruction Method*", Proceedings of the 1st Working International Federation for Information Processing (IFIP) Conference on Software Architecture*, Santo Antônio, Texas, Estados Unidos, Fevereiro, pp. 225-243.

HARRIS, D. R., YEH, A., REUBENSTEIN, H.B, YEH, A.S., 1995, "Reverse Engineering to the Architectural Level", *In Proceedings of the 17th International Conference on Software Engineering*, Scattle, Washington, Estados Unidos, Abril, pp. 186-195.

HARRIS, D. R., YEH, A., REUBENSTEIN, H.B., 1997a, "*Extracting Architectural Features from Source Code"*, *The Mitre Journal*, 1997.

HARRIS, D. R., YEH, A.S., CHASE, M.P., 1997b, "Manipulating Recovered Software Architecture Views", *In Proceedings of the 19th International Conference on Software Engineering*, Massachusetts, USA, Maio, pp.184-194.

HAYES, R., 1994, "Architecture-Based Acquisition and Development of Software Guidelines and Recommendations from the ARPA Domain-Specific Software Architecture (DSSA) Program", *Technical Report, Tecknowledge Federal System*, Outubro.

HITZ, M., MONTAZERI, B., 1996, "Chidamber and Kemerer's Metrics Suite: A Measurement Theory Perspective", *IEEE Transactions on Software Engineering, Vol. 22, Nº 4,* Abril, pp. 267-271.

HOFMEISTER, C., NORD, R.L., SONI, D., 1999, "Describing Software Architecture with UML", *Proceedings of 1st Working IFIP Conference on Software Architecture*, Santo Antônio, Texas, Estados Unidos, Fevereiro, pp. 145-160.

IBM, 2003, "Rose 2001", disponível na Internet em http://www-306.ibm.com/software/rational/, último acesso em 05/03/2004.

IBM RESEARCH, 2001, "Jinsight 2.1", disponível na Internet em http://www.alphaworks.ibm.com/tech/jinsight, último acesso em 15/04/2004.

JERDING, D., RUGABER, S., 1997. "Using Visualization for Architectural Localization and Extraction", *Proceedings of the 4th Working Conference on Reverse Engineering (WCRE'97)*, Amsterdam, Holanda, Outubro, pp. 56-65.

JONES, T.C., 1986, "Programming Productivity", *1ed, McGraw-Hill Book Company.*

KANG, K.C., COHEN, S.G., HESS, J.A., NOVAK, W.E., PETERSON, A.S., 1990, "Feature-Oriented Domain Analysis (FODA): Feasibility Study", *SEI Technical Report, CMU/SEI-90-TR-21, ESD-90-TR-222.*

KANG, K., KIM, S., LEE, J., et al., 1998, "FORM: a Feature-Oriented Reuse Method with Domain-Specific Reference Architectures", *SEI Technical Report.*

KAZMAN, R., ABOWD, G., BASS, L., WEBB, M., 1994, "SAAM: A Method for Analyzing the Properties of Software Architectures", *Proceedings of the 16th International Conference on Software Engineering,* Sorrento, Itália, Maio, pp. 1981-1990.

KAZMAN, R., CARRIÈRE, S.J., 1997, "Playing Detective: Reconstructing Software Architecture from Available Evidence", *Technical Report CMU/SEI-97-TR-010*, Carnegie Mellon University, Estados Unidos, Outubro.

KAZMAN, R., KLEIN, M., CARRIERE, S.J., BARBACCI, M., 1999a, "Experience with Performing Architecture Tradeoff Analysis", *Proceedings of the 21st International Conference on Software Engineering,* ACM Press, Los Angeles, Califórnia, Maio, pp. 54-63.

KAZMAN, R., WOODS, S. G., CARRIÈRE, S. J., 1999b, "The Perils and Joys of Reconstructing Architectures", *International Software Architecture Workshop (ISAW-3)*, Orlando, Florida, Estados Unidos, Outubro.

KLEIN, M., KAZMAN, R., BASS, L., CARRIERE, S.J., BARBACCI, M., LIPSON, H., 1999, "Attribute-Based Architecture Styles", *Proceedings of the 1st Working IFIP Conference on Software Architecture (WICSAI),* Santo Antônio, Texas, Estados Unidos, Fevereiro, pp. 225-243.

KLOCWORK INSIGHT, 2002, disponível na Internet em http://www.klocwork.com/index.asp, último acesso em 05/03/2004.

KNOR, R., TRAUSMUTH, G., WEIDL, J., VAN DER LINDEN, F., 1998, "Reengineering C/C++ Source Code by Transforming State Machines", *In Development and Evolution of Software Architectures for Product Families, Second International ESPRIT ARES Workshop*, Las Palmas de Gran Canaria, Espanha, Fevereiro.

KRASNER, G., POPE, S., 1988, "A Cookbook for using the model view controller user interface paradigm in Smalltalk-80". *Journal of Object-Oriented Programming, 1(3),* Agosto\Setembro, pp. 26-49.

KRIKHAAR, R.L., 1999, "Software Architecture Reconstruction", Dissertação de Doutorado, Universidade de Amasterdam, Holanda.

KRUCHTEN, P.B., 1995, "The 4+1 View Model of Architecture", *IEEE Software, November*, *Vol. 12, Number 6*, pp. 42-50.

KRUCHTEN, P., 2000, "The Rational Unified Process: An Introduction", *Addison-Wesley, 2* nd ed., Outubro.

LAINE, P.K., 2001, "The Role of Software Architecture in Solving Fundamental Problems in Object-Oriented Development of Large Embedded Systems", *Proceedings of the Working IEEE/IFIP Conference on Software Architecture,* Amsterdam, Holanda, Agosto, pp. 14-23.

LORENZ, M., KIDD, J., 1994, “Object-Oriented Software Metrics”, *1 ed., New Jersey, Prentice-Hall.*

MANCORIDIS, S., MITCHELL, B.S., RORRES, C., CHEN, Y., GANSNER, E. R., 1998, "Using Automatic Clustering to Produce High-Level System Organizations of Source Code", *In Proceedings of the 6th International Workshop on Program Comprehension,* Philadelphia, PA, USA, Junho, pp. 45-52.

MEDVIDOVIC, N., TAYLOR, R., 2000, **“**A Classification and Comparison Framework for Software Architecture Description Languages”, *IEEE Transations on Software Engineering, Vol.26, no.1,* pp. 70-93.

MENDES, A., 2002, “Arquitetura de Software: Desenvolvimento Orientado para Arquitetura”, Rio de Janeiro, ed. Campus.

MENDONÇA, N.C., KRAMER, J., 1996, “Requirements for an Effective Architecture Recovery Framework*”, Proceedings of the Second International Software Architecture Workshop (ISAW-2*), pp. 101-105.

MENDONÇA, N.C., KRAMER, J., 2001, “Architecture Recovery for Distributed Systems*”, SWARM Forum at the Eighth Working Conference on Reverse Engineering*, Sttutgart, Alemanha, Outubro.

METTALA, E., GRAHAM, M.H., 1992, “The Domain-Specific Software Architecture”, *SEI Technical Report CMU/SEI-92-SR-009*, Carnegie Mellon University, Estados Unidos.

MONROE, R., KOMPANEK, A., MELTON, R., GARLAN, D., 1997, “Architectural Styles, Design Patterns and Objects”, *Software, IEEE, Volume 14, Issue 1,*Janeiro-Fevereiro, pp. 43-52.

O’BRIEN, L., SMITH, D., 2002, “MAP and OAR Methods: Techniques for Developing Core Assets for Software Product Lines from Existing Assets”, *Technical Note, CMU/SEI-2002-TN-007,* Estados Unidos.

O’BRIEN, L., STOERMER, C. 2001, “MAP: Mining Architectures for Product Line Evaluations”, *Proceedings of the Third Working IFIP Conference on Software Architecture (WICSA),* Amsterdam, Holanda, Agosto, pp. 35-44.

O'BRIEN, L., STOERMER, C., VERHOEF, C., 2002, "Software Architecture Reconstruction: Practice Needs and Current Approaches", Technical Report, CMU/SEI 2002 – TR – 024, Agosto.

ODYSSEY, 2003, projeto Odyssey, grupo de reutilização da COPPE Sistemas, www.cos.ufrj.br/~odyssey, útlimo acesso em 05/03/2004.

PASHOV, I., RIEBISCH, M., 2003, "Using Feature Modeling for Program Comprehension and Software Architecture Recovery", *Proc. of the 10th IEEE Symposium and Workshops on Engineering of Computer-Based Systems,* Huntsville, Alabama, Estados Unidos, Abril, IEEE Computer Society, pp. 297-304.

PENEDO, M., RIDDLE, W., 1993, "Process-Sensitive SEE Architecture (PSEEA) – Workshop Summary", *Software Engineering Notes, ACM SIGSOFT, Vol. 18, Nº 3,* Julho, pp. A78-A94.

PRESSMAN, R. S., 2001, "Software Engineering, A practitioner's Approach", 5 edição, *McGraw-Hill*.

REASONING SYSTEMS, Inc., 1992, "Refine/C User's Guide ", Palo Alto, Califórnia, Estados Unidos, Reasoning Systems, Inc.

RIVA, C., RODRIGUEZ, J. V., 2002, "Combining Static and Dynamic Views for Architecture Reconstruction", *Sixth European Conference on Software Maintenance and Reengineering (CSMR'02)*, Budapeste, Hungria, Março, pp. 47-56

SARTIPI, K., KONTOGIANNIS, K., MAVADDAT, F., 1999, "Software Architecture Recovery", Software Engineering Group, University of Waterloo, Waterloo, Ontario, Estados Unidos, Technical Report.

SARTIPI, K., KONTOGIANNIS, K., MAVADDAT, F., 2000, "A Pattern Matching Framework for Software Architecture Recovery and Restructuring", *In Proceedings of the 8th International Workshop on Program Comprehension*, Limerick, Irlanda, Junho, pp. 37-47.

SEI, 2004, "Product Line Approach to Software Development", *Software Engineering Institute, Carnegie Mellon University,* disponível na Internet em http://www.sei.cmu.edu/plp/plp_init.html, último acesso em 14/02/2004.

SHAW, M., 1989, “Larger Scale Systems Require Higher-Level Abstractions”, *Proceedings of the International Workshop on Software Specification and Design*, Pittsburgh, PA, Estados Unidos, Maio, pp. 143-146.

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

SHAW, M., CLEMENTS, P.,1997, "A Field Guide to Boxology: Preliminary Classification of Architectural Styles for Software Systems", *Proceedings of Computer Software and Applications Conference (COMPSAC '97), The Twenty-First Annual International ,* Washington, DC, Estados Unidos, Agosto, pp. 6-13.

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

SIMOS, M., 1996, "Organization Domain Modeling (ODM): Domain Engineering as a Co-Methodology to Object-Oriented Techniques", *Fusion Newsletter, vol.4, Hewlett-Packard Laboratories*, pp. 13-16.

SYSTÄ, T, 1999, “On the Relationships between Static and Dynamic Models in Reverse Engineering Java Software”, *Sixth Working Conference on Reverse Engineering,* Atlanta, Georgia, Estados Unidos, Outubro, pp. 304-313.

TAKAHASHI, S., ISHIOKA, T., 2000, “Trace Visualization and Analysis Tool for Supervisory Control Systems”, *In the Proceedings of IEEE International Conference on Systems, Man, and Cybernetics, vol. 2,* Nashville, TN, Estados Unidos, Outubro, pp.1198-1203.

TIGRIS, 2004, “ARGO UML”, disponível na Internet em http://argouml.tigris.org/, último acesso em 31/03/2004.

TOGETHER SOFT, 2003, "Together 6.01", disponível na Internet em http://www.borland.com/together/index.html, último acesso em 05/03/2004.

TRACZ, W., 1987, “Software Reuse: Motivations and Inhibitors”, In Proceedings of COMPCON’87, Santa Clara, Estados Unidos, Fevereiro, pp. 358-363.

VAUCLAIR, S., 2002, “Extensible Java Profiler”, disponível na Internet em http://ejp.sourceforge.net/, último acesso em 31/03/2004.

VERONESE, G. O., NETTO, F. J., 2001, "ARES: Uma Ferramenta de Auxílio à Recuperação de Modelos UML de Projeto a partir de Códigos Java", *Projeto Final de Curso de Bacharelado em Informática- Instituto de Matemática/UFRJ,* Outubro.

XAVIER, R., 2001, "Criação e Instanciação de Arquiteturas de Software Específicas de Domínio no Contexto de uma Infra-Estrutura de Reutilização", *Dissertação de Mestrado, COPPE UFRJ*, Rio de Janeiro, Brasil, Julho.

YACOUB, S.M., AMMAR, H. H., ROBINSON, T., 1999, "Dynamic Metrics for Object Oriented Designs", *In the Proceedings of the 6th International Software Metrics Symposium,* Boca Raton, FL USA, Novembro, pp. 50-61.

WALL, L., SCHWARTZ, R., 1991, "Programming Perl", *Sebastopol, CA: O'Reilly & Associates*

WANG, Y., DEWITT, D. J., CAI, J., 2003, "X-Diff: An Effective Change Detection Algorithm for XML Documents", *In the Proceedings of the 19th International Conference on Data Engineering,* Bangalore, India, Março, pp. 519-530.

WEISER, M., 1984, "Program Slicing", *IEEE Transactions on Software Engineering,* 10(4):352-357, Julho.

WENTZEL, K., 1994, "Software Reuse, Facts and Myths", *In Proceedings of 16th Annual International Conference on Software Engineering*, Sorrento, Itália, Maio, pp. 267-273.

WERNER, C. M. L., et al. 2004, "Odyssey: uma Infra-Estrutura de Reutilização Baseada em Modelos de Domínio", grupo de reutilização da COPPE Sistemas, disponível na Internet em www.cos.ufrj.br/~odyssey, útlimo acesso em 05/03/2004.

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

WONG, K., TILLEY, S., MÜLLER, H. STOREY, M.A., 1994, "Programmable Reverse Engineering". *International Journal of Software Engineering and Knowledge Engineering* 4, 4, Dezembro, pp. 501-620.