UMA ABORDAGEM DE APOIO À CRIAÇÃO DE ARQUITETURAS DE REFERÊNCIA DE DOMÍNIO BASEADA NA ANÁLISE DE SISTEMAS LEGADOS

ALINE PIRES VIEIRA DE VASCONCELOS

TESE SUBMETIDA AO CORPO DOCENTE DA COORDENAÇÃO DOS PROGRAMAS DE PÓS-GRADUAÇÃO DE ENGENHARIA DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE DOUTOR EM CIÊNCIAS EM ENGENHARIA DE SISTEMAS E COMPUTAÇÃO.

Aprovada por:

_____

Profª. Cláudia Maria Lima Werner, D.Sc.

_____

Prof. Guilherme Horta Travassos, D.Sc.

_____

Profª. Marta Lima de Queirós Mattoso, D.Sc.

_____

Prof. Nicolas Anquetil, Ph.D.

_____

Profª. Cecília Mary Fischer Rubira, Ph.D.

RIO DE JANEIRO, RJ – BRASIL
ABRIL DE 2007

VASCONCELOS, ALINE PIRES VIEIRA DE

Uma Abordagem de apoio à Criação de Arquiteturas de Referência de Domínio baseada na Análise de Sistemas Legados [Rio de Janeiro] 2007

XVII, 267 p. 29,7 cm (COPPE/UFRJ, D.Sc., Engenharia de Sistemas e Computação, 2007)

Tese – Universidade Federal do Rio de Janeiro, COPPE

1. Reutilização de Software
2. Arquiteturas de Referência de Domínio
3. Engenharia Reversa

I. COPPE/UFRJ II. Título (série)

À minha mãe Maria Rachel (*in memorian*).

Agradecimentos

Aos meus irmãos Alessandra, Isabella, Cristiana, Bárbara e Mauricinho, e ao meu sobrinho Pedro, pelas palavras de carinho, horas ao telefone, amor e amizade infindos. Eles são essenciais para que eu siga adiante!

Ao meu pai, Maurício, pela amizade, palavras de sabedoria, amor e apoio constantes. Ele é o meu porto seguro!

À professora, e agora minha amiga, Cláudia Werner, pelos ensinamentos transmitidos, paciência em vários momentos difíceis, palavras de incentivo e de crítica, quando necessário. A convivência com a Cláudia, a sua amizade e a sua sabedoria certamente me fazem hoje uma pessoa melhor. A minha admiração pela sua contribuição ao desenvolvimento da Engenharia de Software no Brasil.

À minha amiga, doutora em Letras, Analice Martins, por compreender profundamente a árdua tarefa de trilhar o desenvolvimento de uma tese, me apoiando incondicionalmente com sua amizade e seu carinho, a qualquer momento em que eu necessitasse.

Aos amigos do grupo de reutilização da COPPE/Sistemas, cujo apoio e amizade me ajudaram muito nessa caminhada. Em especial aos parceiros diretos desta pesquisa, Ana Maria Moura, Carlos Melo Jr., Eldanae Teixeira, Guilherme Kümmel, Rafael Cepêda e Regiane Oliveira, pela dedicação e amizade.

Um agradecimento especial à Ana Paula Bacelo e ao Leonardo Murta. Sem seu apoio e sua amizade, eu não teria chegado até aqui.

Ao professor Guilherme Travassos, com quem eu tanto aprendi sobre Engenharia de Software e pesquisa. Minha admiração pela sua seriedade e dedicação à pesquisa científica em Engenharia de Software, contribuindo para o seu desenvolvimento no Brasil.

À professora Karin Becker, pela boa vontade no apoio ao entendimento dos conceitos de Mineração de Dados.

Ao CEFET Campos, em especial aos diretores Luiz Augusto Caldas Pereira e Cibele Daher Botelho Monteiro, pelo apoio à realização do meu doutorado, através de constantes renovações do meu afastamento.

Às minhas amigas campistas Denize Teller, Soraya Dutra, Viviane Daher, Jane Simões, Ludmila Monteiro e Renata Britto, pelas horas de descontração nos almoços das sextas-feiras campistas e eternos carinho e incentivo.

Aos colegas e amigos que apoiaram os meus estudos experimentais Alexandre Correa, Alexandre Dantas, Carlos Melo Jr., João Gustavo Prudêncio, José Fortuna, Leonardo Murta, Marco Lopes, Paula Cibele Fernandes, Paula Mian, Rafael Barcelos, Rodrigo Spínola e Wladmir Chapetta.

À TecGraf, em especial ao Carlos Cassino, pelo apoio no meu último estudo experimental.

Aos professores Cecília Rubira, Guilherme Travassos, Marta Mattoso e Nicolas Anquetil por terem aceitado participar da minha banca de doutorado.

Resumo da Tese apresentada à COPPE/UFRJ como parte dos requisitos necessários para a obtenção do grau de Doutor em Ciências (D.Sc.)

UMA ABORDAGEM DE APOIO À CRIAÇÃO DE ARQUITETURAS DE REFERÊNCIA DE DOMÍNIO BASEADA NA ANÁLISE DE SISTEMAS LEGADOS

Aline Pires Vieira de Vasconcelos

Abril/2007

Orientadora: Cláudia Maria Lima Werner

Programa: Engenharia de Sistemas e Computação

Grandes empresas costumam possuir sistemas de software que representam esforço e recursos investidos, além de embutirem conhecimento sobre o negócio. É comum que elas desenvolvam software no mesmo domínio, a fim de atender a diferentes clientes em um mesmo ramo de negócio. Esse fato tem motivado a adoção de abordagens de reutilização como Engenharia de Domínio (ED) e Linha de Produtos (LP), nas quais a arquitetura de referência de domínio ou DSSA representa a base para a instanciação de aplicações. Embora esses sistemas existentes, comumente denominados sistemas legados, representem uma das fontes de informação essenciais para a ED e LP, eles, em geral, não possuem documentação atualizada para a sua compreensão. Nesse contexto, a Engenharia Reversa (ER) provê técnicas para a reconstrução de modelos para esses sistemas, partindo da análise estática ou dinâmica. Porém, não oferece apoio à análise desses modelos para a sua reutilização. Diante desse cenário, esta tese propõe uma abordagem de apoio à criação de DSSAs a partir de sistemas legados, que envolve: um processo de ER, com foco na recuperação de arquiteturas; e a comparação das arquiteturas recuperadas em um domínio, identificando as suas semelhanças e diferenças. A análise dinâmica é priorizada, embora a estática a complemente. As abordagens de ED e LP existentes não costumam oferecer esse apoio sistemático à criação de DSSAs a partir de sistemas legados, como a abordagem proposta, que define uma seqüência de atividades, apoiadas por critérios, técnicas e ferramental.

Abstract of Thesis presented to COPPE/UFRJ as a partial fulfillment of the requirements for the degree of Doctor of Science (D.Sc.)

AN APPROACH TO SUPPORT THE CREATION OF DOMAIN REFERENCE ARCHITECTURES BASED ON LEGACY SYSTEMS ANALYSIS

Aline Pires Vieira de Vasconcelos

April/2007

Advisor: Cláudia Maria Lima Werner

Department: Computer and Systems Engineering

Large organizations usually have software systems that represent effort and resources invested, besides encompassing business knowledge. Moreover, these companies frequently develop systems of the same domain to different clients in the same business area. It has been motivating the adoption of reuse approaches, such as Domain Engineering (DE) and Product Line (PL), in which the domain reference architecture or DSSA represents the basis for application instantiation. Although these existent systems, usually named legacy systems, represent one of the most meaningful domain information sources for the construction of DSSAs, in general, they do not have an up-to-date documentation that can help in their comprehension. In this context, Reverse Engineering (RE) provides a set of techniques that aids in reconstructing models for these systems, starting from static or dynamic analysis. However, RE does not support model analysis for reuse. Considering this scenario, this thesis proposes an approach to support the creation of DSSAs based on legacy systems analysis, involving: a RE process, focusing on architecture recovery; and the comparison of the recovered architectures in the same domain in order to detect their similarities and variability. Dynamic analysis is prioritized, although static analysis complements its results. Existent DE and PL approaches, generally do not provide a systematic support to DSSAs specification from legacy systems, like the proposed approach, that defines a set of activities with supporting criteria, techniques, and tool set.

Índice

Índice de Figuras

Índice de Tabelas

Lista de Acrônimos

Acrônimo	Termo Relacionado
ADL	*Architectural description language*. Representa uma linguagem para a descrição de modelos arquiteturais de software.
ADS	Ambiente de Desenvolvimento de Software
ArchMine	Abordagem proposta para a recuperação de arquitetura de software baseada na mineração, que representa a primeira etapa de LegaToDSSA.
ArchToDSSA	Abordagem proposta de comparação de arquiteturas e criação de DSSA, que representa a segunda etapa de LegaToDSSA.
ArchToDSSATool	Ferramenta para apoio às atividades de comparação de arquiteturas, detecção das suas semelhanças e diferenças, e criação de DSSAs, desenvolvida nesta tese para apoio às atividades de ArchToDSSA.
ArchTrace	Ferramenta para apoiar a evolução de ligações de rastreabilidade entre a arquitetura e a sua implementação, desenvolvida na tese de doutorado de MURTA (2006). Ela foi utilizada como objeto de um dos estudos de caso de ArchMine.
Ares	Ferramenta de extração da estrutura estática uitlizada na abordagem ArchMine (VERONESE e NETTO, 2001).
ArqCheck	Abordagem de avaliação de arquitetura desenvolvida na dissertação de mestrado de BARCELOS (2006) e utilizada na etapa de avaliação de arquitetura de ArchMine.
DBC	Desenvolvimento Baseado em Componentes
Diff	Algoritmo de comparação e cálculo de diferenças entre modelos.
DSSA	*Domain specific software architecture.* Representa uma arquitetura de referência de domínio.
EA	Engenharia de Aplicação
ED	Engenharia de Domínio
ER	Engenharia Reversa
LegaToDSSA	Abordagem proposta de criação de DSSA baseada na análise de sistemas legados.
LP	Linha de Produtos
MOF	*Meta Object Facility*
Odyssey-FEX	Notação para o modelo de características do ambiente Odyssey.
OO	Orientação a Objetos
Phoenix	Ferramenta para a reconstrução de modelos dinâmicos, desenvolvida nesta tese para apoiar essa atividade do processo da abordagem ArchMine.
TraceMining	Ferramenta para apoio à mineração e reconstrução de elementos arquiteturais, desenvolvida nesta tese para apoiar essas atividades do processo da abordagem ArchMine.
Tracer	Ferramenta de coleta de rastros de execução, desenvolvida nesta tese para apoiar essa atividade do processo da abordagem ArchMine.
UML	Linguagem de modelagem unificada.
VP	Ponto de Variação
XMI	*XML Metadata Interchange*
XML	*eXtensible Markup Language*

Capítulo 1 – Introdução

1.1 – Referencial Teórico

Com o passar dos anos, o desenvolvimento de software vem lidando com software cada vez mais complexo e, ao mesmo tempo, com uma demanda cada vez maior por software de qualidade e entregue em menor tempo. Nesse contexto, a reutilização de software representa uma área da Engenharia de Software que vem ganhando força. Ela pode ser definida como o processo de criação de sistemas de software a partir de software preexistente (KRUEGER, 1992), trazendo consigo a promessa de aumentar a produtividade e diminuir custos do desenvolvimento, além de melhorar a qualidade do produto, através da reutilização de artefatos de software já testados e utilizados em outros contextos (FRAKES e KANG, 2005).

Inicialmente, a reutilização de software era mais focada em código fonte e em um baixo nível de granularidade em relação à unidade reutilizável, sendo realizada através de bibliotecas de rotinas. Essas bibliotecas costumavam ser alimentadas por procedimentos e funções utilitárias, de uso geral em diferentes projetos de software. Com a crescente adoção do paradigma de orientação a objetos (OO), a unidade reutilizável aumentou seu nível de granularidade, através de classes que encapsulam dados e funções. Entretanto, através do paradigma de Desenvolvimento Baseado em Componentes (DBC), esse nível de granularidade da unidade reutilizável pôde se tornar ainda maior, por meio de componentes de software, que representam artefatos auto-contidos, claramente identificáveis, que descrevem ou realizam uma função específica e possuem interfaces claras, documentação apropriada e um grau de reutilização definido (SAMETINGER, 1997).

Entretanto, não só o nível de granularidade da unidade reutilizável é um fator importante para uma abordagem de reutilização, mas também os níveis de abstração considerados em um processo de reutilização de software. Conforme afirmam WERNER e BRAGA (2005), para que a reutilização possa ser realmente efetiva, deve-se considerá-la em todas as fases do desenvolvimento de aplicações, desde a fase de análise até a fase de implementação. Abordagens de reutilização de software, como Engenharia de Domínio (ED) e Linha de Produtos (LP) apresentam essa preocupação, estabelecendo o desenvolvimento de artefatos de software reutilizáveis desde o nível de

abstração de análise até o nível de implementação. A ED e a LP apresentam muitas semelhanças entre si, sendo que a ED pode ser definida como: "o processo de identificação e organização do conhecimento sobre uma classe de problemas, i.e. o domínio do problema, para apoiar a sua descrição e solução" (PRIETO-DIAZ e ARANGO, 1991). A LP pode ser vista como uma vertente da ED, cujo foco foi transferido para o âmbito empresarial (LEE *et al.*, 2002).

Tanto na ED quanto na LP, as arquiteturas de referência de domínio representam o elemento-chave para a instanciação de aplicações e reutilização dos artefatos produzidos. Essas arquiteturas representam o arcabouço onde elementos arquiteturais específicos de aplicações podem ser conectados e onde elementos arquiteturais de domínio podem ser adaptados ou estendidos para novas aplicações. Convém ressaltar que as arquiteturas de software são artefatos importantes em qualquer processo de desenvolvimento por abstraírem detalhes de implementação, permitindo a compreensão da solução em um nível mais alto de abstração. Além disso, é consenso que elas devem ser representadas através de diferentes perspectivas (KRUCHTEN, 1995; HOFMEISTER et al., 1999; BARCELOS, 2006; GORTON, 2006), destacando aspectos estáticos e dinâmicos em diferentes e consistentes representações. Cada representação interessa a um grupo de *stakeholders*, onde *stakeholders* representam grupos ou indivíduos envolvidos, de forma direta ou indireta, em um projeto de software ou que possuem algum interesse no resultado obtido por esse projeto (BARCELOS, 2006).

As arquiteturas de referência de domínio representam o elemento central das DSSAs (*Domain Specific Software Architectures*), que, segundo XAVIER (2001), são mais do que uma arquitetura para um determinado domínio de aplicações, mas sim uma coleção de componentes, especializados para um determinado tipo de tarefa (domínio) e generalizados para que seja possível seu uso efetivo através do domínio.

Em LP, as arquiteturas de referência são conhecidas como *Product Line Architectures* (PLA) (GORTON, 2006). Tanto na ED quanto na LP, as arquiteturas de referência de domínio devem representar os elementos arquiteturais do domínio, seus relacionamentos, semelhanças e diferenças.

A fim de apoiar a criação dessas arquiteturas de referência, sistemas legados disponíveis para o domínio podem ser analisados, pois, segundo KANG (1990), eles representam uma das fontes de informação essenciais para a ED. O termo "sistemas legados" é um eufemismo para sistemas existentes nas empresas, geralmente antigos, mal documentados e mal projetados que devem ser mantidos por muitos anos, por

serem críticos para o negócio de uma organização (PRESSMAN, 2001). Grandes empresas costumam possuir uma base de sistemas legados que representam esforço e recursos investidos no passado, além de embutirem conhecimento sobre o negócio que muitas vezes não pode ser obtido a partir de nenhuma outra fonte de informação, o que motiva a sua evolução contínua e reutilização em novos esforços de desenvolvimento.

Nesse contexto, a Engenharia Reversa (ER) se apresenta como um processo de análise de sistemas existentes que pode auxiliar na recuperação de uma documentação atualizada e, conseqüentemente, no entendimento dos sistemas legados, tanto para fins de manutenção, como para fins de reutilização. A ER, normalmente, objetiva a extração de documentação de projeto a partir de artefatos disponíveis do sistema existente, envolvendo um conjunto de técnicas, que podem se basear na análise estática do software (i.e. do código) ou dinâmica (i.e. do sistema em execução).

Segundo CHIKOFSKY e CROSS II (1990), a ER pode capturar ou recriar o projeto de um software. Por isso, diversas abordagens de recuperação de projeto (*Design Recovery*) (SEEMANN e GUDENBERG, 1998) e remodularização (*Software Remodularization*) (ANQUETIL *et al.*, 1999) são propostas na literatura da área. Com a crescente importância que a arquitetura de software vem ganhando, a ER de projeto no nível de abstração arquitetural passou a ganhar destaque na comunidade, podendo ser definida como uma subárea da ER que visa à obtenção de uma arquitetura documentada para um sistema existente (DEURSEN *et al.*, 2004).

1.2 – Motivação

A maioria das organizações constroem sistemas de software em um domínio de aplicação particular, repetidamente entregando variantes de produtos pela adição de novas características (SUGUMARAN *et al.*, 2006). Dessa forma, é comum que essas organizações estejam migrando para abordagens de ED e LP. Uma vez que as arquiteturas de referência de domínio representam o elemento-chave para uma abordagem de ED ou LP de sucesso, a construção desses artefatos merece atenção nesse processo de migração.

Embora essas organizações costumem possuir uma grande base de sistemas instalados, as abordagens de ED (ex: FORM (KANG *et al.*, 2002), Odyssey-DE (BRAGA, 2000), CBD-Arch-DE (BLOIS, 2006)) e LP (ex: KobrA (ATKINSON *et al.*, 2002), PLUS (GOMAA, 2004)) pesquisadas oferecem um maior apoio à especificação de arquiteturas de referência através da engenharia progressiva, em um direcionamento

de cima para baixo (*top-down*). Apesar de algumas abordagens de LP, como as descritas em (STOERMER e O'BRIEN, 2001; GOMAA, 2004; ALVES *et al.*, 2005; GANESAN e KNODEL, 2005), tratarem da análise de sistemas legados, elas não oferecem um apoio sistemático a essa análise, propondo critérios e técnicas que possam guiar todas as etapas de um processo de baixo para cima (*bottom-up*) para a criação de arquiteturas de referência, desde a extração de modelos dos sistemas legados, passando pela sua comparação para a detecção de semelhanças e diferenças, até a geração de modelos de domínio. A falta desse apoio gera dificuldades e grande esforço nessa análise, em função das grandes dimensões dos sistemas legados e da degradação que a sua estrutura sofre ao longo do tempo. Além disso, quando a ER é voltada à reutilização, ela pode se tornar ainda mais complexa devido a necessidade da recuperação de modelos de diferentes sistemas que possam ser consistidos em um modelo de domínio.

Diversas abordagens de ER são propostas na literatura, entretanto, em geral, o foco não costuma ser na reutilização de software, mas sim no apoio à manutenção (RIVA e RODRIGUEZ, 2002; SARTIPI, 2003; MITCHELL e MANCORIDIS, 2006). Dessa forma, existe uma dificuldade em consistir os modelos arquiteturais recuperados em um modelo de domínio, uma vez que o tipo e a nomenclatura dos elementos arquiteturais recuperados podem divergir. Além disso, as abordagens existentes, em geral, apresentam critérios para a reconstrução dos elementos arquiteturais, a partir de elementos do código, que são específicos para determinadas linguagens de programação (ex: (HARRIS *et al.*, 1997b; GALL e PINZGER, 2002; SCHMERL *et al.*, 2006)) ou para determinadas aplicações ou domínios (ex: (RIVA e RODRIGUEZ, 2002), (KAZMAN e CARRIÈRE, 1997)). Por fim, as abordagens de ER realizam a avaliação das arquiteturas recuperadas de forma aleatória (*ad-hoc*). Essa forma de avaliação pode não levar à detecção de inconsistências no modelo arquitetural recuperado, incorrendo-se no risco dessas inconsistências se propagarem para novas aplicações quando as arquiteturas são recuperadas para a sua reutilização.

Em relação à comparação dos modelos para a detecção de semelhanças e diferenças, elementos equivalentes entre modelos de diferentes aplicações podem estar descritos de forma diferente, possuindo, por exemplo, nomes diferentes, embora sendo semanticamente equivalentes. Isso porque apesar dos sistemas analisados fazerem parte de um mesmo domínio, eles não foram necessariamente desenvolvidos por uma mesma equipe, em um mesmo período de tempo ou dentro de um mesmo departamento. Abordagens de comparação de modelos para a detecção de diferenças costumam

considerar diferentes versões de um modelo na comparação, onde os elementos de diferentes modelos possuem o mesmo nome ou identificador (CHEN *et al.*, 2003; MEHRA *et al.*, 2005; OLIVEIRA *et al.*, 2005a), sendo essas informações utilizadas como base para estabelecer a comparação. Como essas premissas não podem ser assumidas na comparação de modelos de diferentes sistemas em um domínio, a comparação se torna uma tarefa mais árdua, devendo levar em conta a grande variedade de elementos equivalentes com nomes e estruturas distintos encontrados nas diferentes aplicações analisadas.

Finalmente, com o crescente volume de desenvolvimento de sistemas segundo o paradigma OO nas últimas décadas, abordagens que apóiem a criação de arquiteturas de referência através da análise de sistemas desenvolvidos segundo esse paradigma se tornam necessárias.

1.3 – Objetivos

Diante das motivações expostas, esta tese apresenta como objetivo central propor uma abordagem sistemática de apoio à criação de arquiteturas de referência de domínio através da análise de sistemas legados OO, denominada LegaToDSSA. A fim de atingir esse objetivo central, a abordagem LegaToDSSA, proposta nesta tese, envolve uma etapa de ER, voltada à recuperação de arquiteturas, realizada através da abordagem ArchMine, e uma etapa de comparação de arquiteturas e geração de arquiteturas de referência parcialmente especificadas, realizada através da abordagem ArchToDSSA.

A fim de apoiar a reutilização, ArchMine recupera elementos arquiteturais que representam conceitos do domínio, representando possíveis candidatos a componentes de software do domínio. Para tal, a hipótese de pesquisa estabelecida é que a recuperação de elementos arquiteturais através do agrupamento de classes que implementam juntas um conjunto comum de funcionalidades do domínio leva à identificação de elementos arquiteturais que possam representar conceitos do domínio.

Uma vez que arquiteturas de software devem ser representadas através de diferentes perspectivas, a fim de permitir a compreensão do software por diferentes *stakeholders*, ArchMine recupera diferentes visões arquiteturais, separando aspectos estáticos e dinâmicos. Dessa forma, ArchMine visa contribuir também com a manutenção de sistemas. ArchMine se baseia em critérios genéricos para a reconstrução de elementos arquiteturais, a fim de que todo o esforço de definição da abordagem de

recuperação de arquitetura possa ser reaproveitado para diferentes domínios e linguagens de programação.

Finalmente, ArchMine adota uma abordagem de avaliação arquitetural, i.e. a ArqCheck de BARCELOS (2006), que permite que a avaliação das arquiteturas recuperadas não seja realizada de forma aleatória. ArqCheck é estendida nesta tese para a avaliação das arquiteturas quanto ao atendimento do atributo de qualidade Reusabilidade. A hipótese estabelecida nesse ponto é que, com a adoção de uma abordagem de avaliação arquitetural, é possível guiar a avaliação das arquiteturas recuperadas para atender aos objetivos estabelecidos para ArchMine.

A fim de verificar se esses objetivos foram atendidos, estudos experimentais foram conduzidos sobre a abordagem de recuperação de arquitetura ArchMine. Através desses estudos, foi possível obter indícios da viabilidade de ArchMine na recuperação de modelos arquiteturais que podem ser reutilizados na construção de uma arquitetura de referência de domínio.

Em relação à comparação de arquiteturas para a criação de arquiteturas de referência, ArchToDSSA apóia a comparação de modelos arquiteturais de diferentes sistemas em um mesmo domínio. Dessa forma, ArchToDSSA assume que ocorram diferenças de nomenclatura entre elementos arquiteturais equivalentes, além de diferenças estruturais entre as diferentes arquiteturas comparadas. A fim de tratar diferenças de nomenclatura, ArchToDSSA incorpora um dicionário de termos sinônimos no domínio.

Em relação às diferenças estruturais, ArchToDSSA considera questões estruturais durante a comparação, verificando o tipo dos elementos comparados nos diferentes modelos. Além disso, ArchToDSSA gera arquiteturas de referência parcialmente especificadas, exigindo que o Engenheiro de Domínio complemente a sua especificação.

O maior objetivo de ArchToDSSA é detectar as semelhanças e diferenças entre as arquiteturas comparadas. Para tal, a abordagem envolve a verificação de elementos que indicam variabilidade no domínio e em quantas arquiteturas cada elemento aparece, indicando a sua opcionalidade. A opcionalidade é uma propriedade que indica que o elemento pode ou não estar presente nas aplicações do domínio e a variabilidade se refere a pontos do domínio onde as aplicações podem ser configuradas. ArchToDSSA faz parte do trabalho de mestrado de KÜMMEL (2007), em fase de conclusão. Sua especificação e desenvolvimento foram apoiados pelo trabalho de pesquisa desta tese.

Por fim, dentre os objetivos desta tese, se inclui o apoio ferramental às atividades de ArchMine e ArchToDSSA, a fim de reduzir o esforço exigido na recuperação e comparação de arquiteturas de sistemas legados, que costumam possuir grandes dimensões. Esse apoio ferramental está integrado a um ambiente de desenvolvimento que permite a reutilização dos modelos reconstruídos em abordagens de ED e/ou LP.

1.4 – Contexto

Esta pesquisa se insere no contexto do Projeto Odyssey (WERNER *et al.*, 1999), que visa o desenvolvimento do ambiente Odyssey, e do Projeto Reuse (WERNER, 2005), cujo objetivo principal é explorar técnicas e ferramentas que apóiem a reutilização de software, permitindo a evolução do ferramental desenvolvido pelo grupo de reutilização da COPPE/UFRJ.

O Odyssey é um ambiente de reutilização baseado em modelos de domínio, provendo tecnologias de apoio à ED, LP e DBC. O Odyssey cobre tanto o desenvolvimento para reutilização, através de processos de ED (BRAGA, 2000; BLOIS, 2006), quanto o desenvolvimento com reutilização, através da Engenharia de Aplicação (EA) (MILER, 2000). Os modelos criados na ED são instanciados através de um processo de EA, que tem por objetivo a construção de aplicações em um determinado domínio. O ambiente Odyssey utiliza o modelo de características (*features*)[1] (KANG *et al.*, 1990), através da notação Odyssey-FEX (OLIVEIRA, 2006b), e modelos UML (*Unified Modeling Language*) (OMG, 2007b) para a modelagem de domínios.

1.5 – Organização da Tese

Partindo desta Introdução, esta tese está organizada em mais 6 capítulos, da seguinte forma:

No **Capítulo 2,** é feita uma revisão da literatura acerca de arquiteturas de software e reutilização. São abordados temas essenciais sobre arquiteturas de software, como Modelos de Descrição Arquitetural, Estilos e Padrões Arquiteturais e Abordagens de Avaliação de Arquitetura. Em relação à reutilização de software, o capítulo apresenta

[1] Características representam aspectos do domínio perceptíveis ao usuário, que definem similaridades e diferenças entre os sistemas existentes nesse domínio (KANG *et al.*, 1990).

abordagens de ED e LP e suas contribuições à criação de arquiteturas de referência de domínio, além de pontos em aberto nesse contexto.

O **Capítulo 3** apresenta uma revisão da literatura sobre técnicas de ER, técnicas de mineração de dados no contexto da ER e técnicas de Comparação de Modelos de Software, demonstrando trabalhos relacionados que aplicam técnicas similares àquelas utilizadas pela abordagem proposta nesta tese. São ressaltados pontos em aberto nesses trabalhos relacionados, apontando áreas onde contribuições ainda são necessárias.

No **Capítulo 4**, é apresentada a abordagem proposta nesta tese, i.e. a LegaToDSSA, sendo descritas detalhadamente as suas 2 etapas, a saber: recuperação de arquiteturas de sistemas legados, realizada através da abordagem ArchMine, e comparação de arquiteturas para apoio à criação de arquiteturas de referência de domínio, realizada através da abordagem ArchToDSSA.

O **Capítulo 5** apresenta o ferramental de apoio à execução das atividades de ArchMine e ArchToDSSA, descrevendo a arquitetura, tecnologias empregadas, funcionalidades e as contribuições de cada um dos protótipos desenvolvidos.

No **Capítulo 6**, são descritos os estudos experimentais conduzidos para avaliar a viabilidade da abordagem de recuperação de arquitetura ArchMine. Estudos experimentais para avaliar a viabilidade de ArchToDSSA fazem parte do trabalho de mestrado de KÜMMEL (2007).

Finalmente, o **Capítulo 7** descreve as contribuições alcançadas nesta tese em função dos objetivos estabelecidos, limitações identificadas em cada etapa da abordagem proposta, além de perspectivas para trabalhos futuros.

Os Anexos desta tese constam ao final deste documento, após as referências bibliográficas. O **Anexo A** apresenta a instrumentação do primeiro estudo experimental que avalia a viabilidade da abordagem de recuperação de arquitetura ArchMine. O **Anexo B** apresenta a instrumentação do estudo experimental comparativo de ArchMine, em que a sua viabilidade é avaliada através da comparação com uma outra abordagem de recuperação de arquitetura. O **Anexo C** apresenta a instrumentação do estudo de viabilidade das extensões realizadas na abordagem de avaliação de arquitetura adotada. O **Anexo D** apresenta a instrumentação do último estudo experimental, realizado nesta tese, para a verificação da viabilidade de avaliação das arquiteturas recuperadas com ArchMine, através da abordagem de avaliação de arquitetura estendida, avaliada no estudo anterior. O **Anexo E** descreve sucintamente a taxonomia da abordagem de modelagem de variabilidades Odyssey-FEX, adotada nesta tese para a representação das

semelhanças e diferenças entre as aplicações do domínio, e o **Anexo F** descreve heurísticas de mapeamento de classes para características definidas, nesta tese, como extensões na Odyssey-FEX. Finalmente, o **Anexo G** apresenta a versão original do *checklist* de ArqCheck, a abordagem de avaliação arquitetural adotada nesta tese, que foi estendido para a avaliação de Reusabilidade.

Capítulo 2 – Arquiteturas de Software e Reutilização

2.1 – Introdução

À medida que o tamanho e a complexidade dos sistemas de software aumentam, o projeto e a especificação da estrutura global do sistema se tornam questões mais importantes do que a escolha de algoritmos e estruturas de dados de computação (SHAW e GARLAN, 1996). Essa estrutura global representa a arquitetura do software e trata de questões, como: a organização do sistema como uma composição de componentes; a topologia dessa organização, estabelecida pelos protocolos de comunicação, controle, sincronização e acesso a dados; o mapeamento de requisitos funcionais para os componentes; a distribuição física na rede; a escalabilidade e o desempenho. Vale ressaltar que o termo "componentes", nesse contexto, é utilizado de forma flexível para representar um componente arquitetural qualquer, diferente do termo "componentes de software" que, nesta tese, é utilizado para representar os componentes auto-contidos de DBC.

A especificação de uma arquitetura adequada é essencial para o sucesso de um projeto de software. Ela representa a base da solução computacional para o problema descrito pelos requisitos do software, sendo o artefato que vai guiar as etapas subseqüentes do desenvolvimento, como o projeto detalhado e a implementação. É por isso que a maior parte do tempo de um arquiteto é dedicada ao particionamento adequado do sistema em um conjunto de componentes, módulos ou objetos inter-relacionados (GORTON, 2006). Diferentes requisitos e restrições vão direcionar esse particionamento, sendo que a escolha de um modelo arquitetural adequado é impactada pelos requisitos funcionais levantados para a aplicação, sendo importante uma adequada atribuição de responsabilidades aos elementos arquiteturais definidos. Além dos requisitos funcionais, a arquitetura sofre ainda influência dos requisitos não-funcionais, como desempenho, escalabilidade, portabilidade, interoperabilidade, disponibilidade, adaptabilidade, dentre outros.

Apesar do reconhecimento da importância da arquitetura de software na última década e do crescimento do número de conferências e periódicos dedicados ao tema (ex: WICSA – *Working IEEE/IFIP Conference on Software Architecture (WICSA, 2007)*; QOSA – *International Conference on Quality of Software Architectures* (QOSA, 2007);

JSA – *Journal of Systems Architecture* (JSA, 2007)), ainda não existe uma definição de arquitetura amplamente aceita pela indústria (GORTON, 2006). Arquitetura de software ainda é uma disciplina um tanto nebulosa. De fato existe uma falta de consenso na comunidade em relação tanto aos conceitos e definições básicas quanto à forma de se representar uma arquitetura de software (BUSCHMANN *et al.*, 1996; CLEMENTS *et al.*, 2004).

Dentre as definições de arquitetura de software que se encontram na literatura, algumas amplamente referenciadas são:

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

2) "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 dessas partes (módulos com interfaces, unidades de hardware, objetos), e dos relacionamentos e restrições entre elas" (D'SOUZA e WILLS, 1999).

3) "A arquitetura de software de um programa ou sistema de computador consiste da estrutura ou estruturas do sistema, que compreende elementos do software, as propriedades externamente visíveis desses elementos e os relacionamentos entre eles" (BASS *et al.*, 2003).

Dentre essas definições, a definição de BASS *et al*. (2003), ou seja, a terceira definição, é a adotada nesta tese, uma vez que ela enfatiza os conceitos essenciais acerca de arquitetura de software, como: elementos que compõem o software, os relacionamentos entre eles, a abstração (i.e. propriedades externamente visíveis) e a necessidade de representação da arquitetura a partir de diferentes perspectivas (i.e. estruturas de sistema).

Considerando a necessidade de se documentar arquiteturas de software, para apoio ao desenvolvimento e manutenção dos sistemas, DIAS e VIEIRA (2000) ressaltam três tipos de elementos básicos que devem ser usados para representar essas estruturas:

- **Elementos de software (i.e. módulo ou componente)**: abstrações responsáveis por representar as entidades que implementam as funcionalidades especificadas. Eles representam o local onde uma computação é realizada, um estado é estabelecido e possuem interfaces,

que especificam os seus serviços providos e requeridos, podendo ser definidas através de portas.

- **Conectores (i.e. relacionamentos ou interfaces)**: abstrações responsáveis por representar as entidades que facilitam a comunicação entre os elementos de software. Diferentemente dos componentes, conectores podem não corresponder a unidades de compilação no sistema implementado, i.e. podem ser implícitos. Quando explícitos, os conectores também possuem interfaces, as quais são definidas através de um conjunto de papéis esperados dos participantes (componentes) em uma interação.
- **Configuração (i.e. organização)**: consiste na forma como os elementos de software e conectores estão organizados.

De acordo com BASS *et al.* (2003), um elemento arquitetural representa tanto um elemento de software quanto um conector. Entretanto, nesta tese esse termo é utilizado referindo-se a elementos de software, uma vez que esses são priorizados na abordagem de recuperação de arquitetura proposta, i.e. ArchMine.

Em abordagens de reutilização de software, como Engenharia de Domínio (ED) e Linha de Produtos (LP), a arquitetura desempenha um papel ainda mais importante, pois ela estabelece a estrutura na qual os elementos arquiteturais do domínio ou de uma linha de produtos são conectados e instanciados, e para a qual os novos elementos das aplicações devem apresentar compatibilidade.

No contexto da ED, por exemplo, surgem as arquiteturas de referência de domínio, que representam arquiteturas para famílias de aplicações (ou domínios), que devem atender a um conjunto de requisitos funcionais e não-funcionais do domínio (SHAW e GARLAN, 1996). Essas arquiteturas se diferenciam das arquiteturas de software, as quais atendem apenas aos requisitos de uma única aplicação. Quando uma arquitetura de referência de domínio é instanciada, muitas vezes ela precisa ser adaptada ou estendida para atender aos requisitos específicos de uma aplicação.

No contexto da LP, as arquiteturas de referência são chamadas de *Product Line Architectures* (PLA)[2], ou seja, arquiteturas de linha de produtos que representam arquiteturas orientadas à reutilização dos artefatos-chave da linha de produtos

[2] Nesta tese, arquiteturas no contexto da ED e da LP serão indistintamente chamadas de arquiteturas de referência ou arquiteturas de referência de domínio.

(GORTON, 2006). Ambas representam o artefato-central para a instanciação de aplicações, sendo que a sua qualidade ou inconsistências se propagam para todas as aplicações instanciadas. Nesse contexto, tanto na ED quanto na LP, uma maior preocupação com a qualidade das arquiteturas é exigida.

Visto que o objetivo desta tese é apoiar a criação de arquiteturas de referência para domínios ou linhas de produtos, através da Engenharia Reversa (ER) de arquiteturas de aplicações, este capítulo aborda temas inerentes às áreas de arquitetura e reutilização de software, ressaltando seus aspectos essenciais e as relações existentes entre elas. Nesse contexto, partindo desta Introdução, o restante do capítulo está organizado da seguinte forma: a seção 2.2 aborda a área de arquitetura de software, descrevendo conceitos-chave e questões relacionadas à descrição e avaliação arquitetural; a seção 2.3 descreve abordagens de reutilização de software, i.e. ED e LP, e o seu apoio à especificação de arquiteturas de referência; a seção 2.4 conclui o capítulo, apresentando considerações acerca do apoio existente à especificação de arquiteturas de referência de domínio e pontos que suscitam contribuições na área.

2.2 – Arquiteturas de Software

Nesta seção, são abordados aspectos-chave na área de arquitetura de software, como os modelos de descrição de arquitetura, estilos e padrões arquiteturais, linguagens e notações para a representação de arquiteturas e métodos de avaliação de arquiteturas.

2.2.1 – Modelos de Descrição de Arquiteturas

Uma arquitetura de software representa um artefato de projeto complexo, havendo uma variedade de formas de "olhar" para ela e compreendê-la (GORTON, 2006). Uma arquitetura descreve diferentes propriedades do software, como a sua divisão em elementos arquiteturais, protocolos de comunicação e controle, sua distribuição física, sincronização, acesso a dados etc. Segundo PENEDO e RIDDLE (1993), uma arquitetura de software deve ser vista e descrita sob diferentes perspectivas (ou visões) e deve identificar seus componentes, relacionamentos estáticos, interações dinâmicas, propriedades, características e restrições. Cada perspectiva ou visão arquitetural descreve um conjunto de propriedades da arquitetura e abstrai características que não são inerentes àquela visão, sendo de interesse para um grupo específico de *stakeholders*.

Com base na idéia de separação de aspectos da arquitetura em diferentes

perspectivas, alguns modelos de visões arquiteturais são propostos na literatura (KRUCHTEN, 1995; HOFMEISTER *et al.*, 1999; CLEMENTS *et al.*, 2004).

O modelo de visões arquiteturais 4+1 (*The 4+1 View Model*) de KRUCHTEN (1995) foi o que tornou notório o termo "visões arquiteturais" (*architectural views*). Ele propõe a organização da descrição arquitetural em 4 visões concorrentes, as quais são amarradas através da 5ª visão, a de cenários, como se segue:

- **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 OO, 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 modelo 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 software de prateleira (COTS – *Commercial-Off-The-Shelf software components)*, de bibliotecas do ambiente operacional ou de componentes de software de terceiros.
- **Visão de Cenários**: ou de casos de uso, é 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.

A Figura 2.1 demonstra que cada visão interessa a um grupo de *stakeholders* e que existe um mapeamento de elementos entre as diferentes visões.

A visão de cenários é central, amarrando as demais visões. KRUCHTEN (1995) ressalta em seu trabalho que nem toda arquitetura de software requer representação em todas as 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.

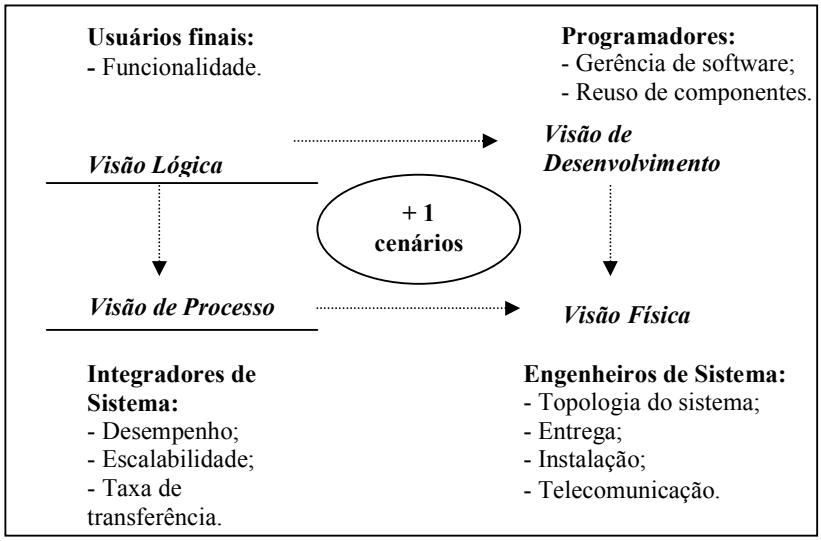

Figura 2.1: Visões do modelo 4+1 de KRUCHTEN (1995).

O modelo 4+1 é amplamente utilizado, tendo sofrido algumas evoluções conceituais sutis desde a sua criação até a sua incorporação ao RUP (*Rational Unified Process* (KRUCHTEN, 2000)), onde as visões são representadas através de modelos da UML[3] (OMG, 2007b) e a visão de cenários passa a se chamar visão de casos de uso.

Além das diferentes visões arquiteturais, uma outra questão a ser considerada é o nível de abstração de uma descrição arquitetural. Segundo GORTON (2006), um dos mecanismos mais poderosos para descrever uma arquitetura é a decomposição hierárquica. Componentes que aparecem em um nível de descrição vão sendo decompostos em mais detalhes durante a especificação do projeto, conforme mostra a Figura 2.2. Diferentes níveis de descrição na hierarquia tendem a ser de interesse a diferentes *stakeholders*. Por exemplo, a equipe de desenvolvimento trabalhando no Componente A, apresentado na Figura 2.2, não tem interesse no detalhamento dos componentes B e C. Por outro lado, Gerentes tendem a estar mais interessados na arquitetura de alto nível, que é a que melhor permite guiar o acompanhamento da produtividade do desenvolvimento.

[3] A UML - Linguagem de Modelagem Unificada (ou *Unified Modeling Language*) é o padrão de linguagem da indústria para especificar, visualizar, construir e documentar artefatos de análise e projeto de sistemas de software.

Além de interessar a diferentes *stakeholders*, os diferentes níveis de abstração arquitetural caracterizam diferentes etapas do desenvolvimento do software. Segundo BARCELOS (2006), nos estágios iniciais do desenvolvimento uma arquitetura em alto nível de abstração, denominada arquitetura inicial, começa a ser construída. Ao longo do processo de desenvolvimento, entretanto, a arquitetura vai sofrendo refinamentos que diminuem o seu nível de abstração, permitindo, por exemplo, a representação dos relacionamentos entre os elementos arquiteturais e os arquivos de código fonte responsáveis por implementá-los (CLEMENTS *et al.*, 2004). O fato da arquitetura representar informações em diferentes níveis de abstração leva à falta de consenso na comunidade acerca da granularidade adequada para se descrever esse artefato (BARCELOS, 2006).

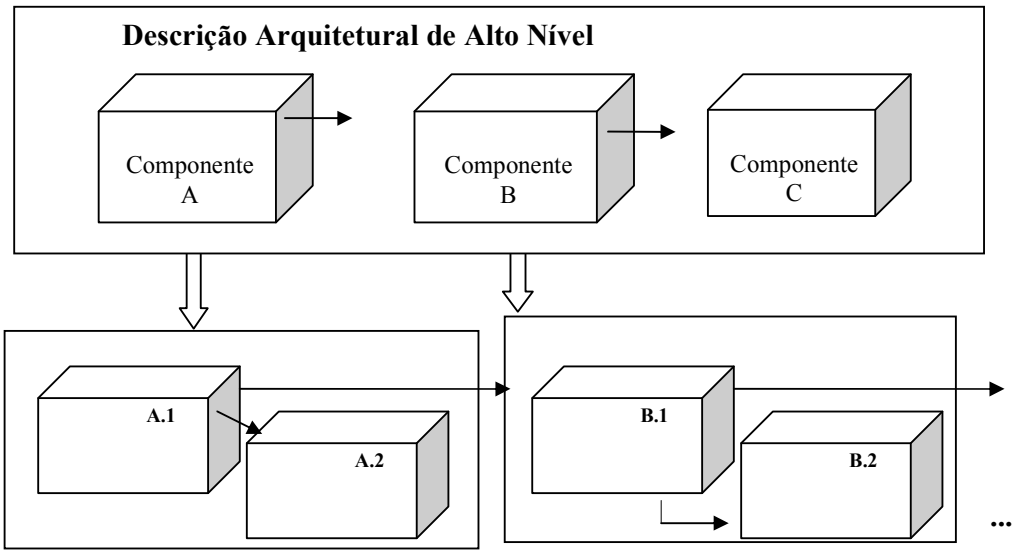

Figura 2.2: Descrevendo arquiteturas hierarquicamente. Adaptado de GORTON (2006).

2.2.1.1 – Representação de Arquiteturas

Uma descrição arquitetural de software, conforme ressaltado na seção anterior, se compõe de diferentes perspectivas. A fim de representar essas perspectivas, uma forma de representação arquitetural deve ser empregada. A forma de representação mais comumente encontrada nos trabalhos da literatura é o uso de caixas e setas representadas sem nenhuma formalidade em diagramas arquiteturais. Essa 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 que estariam representando? O fluxo de controle ou de dados? Ou simplesmente uma dependência de compilação? Essas são perguntas comuns a leitores tentando interpretar os diagramas arquiteturais e a resposta a essas perguntas seria: "esses símbolos representam um pouco disso tudo" (KRUCHTEN, 1995).

Dentre as abordagens para a representação de arquitetura com maior semântica, se encontram: os estilos e padrões arquiteturais, linguagens de descrição de arquitetura (ADLs – *Architecture Description Languages*), arquiteturas específicas de domínio (DSSAs – *Domain Specific Software Architectures*), *frameworks* e a utilização de linguagens de modelagem, como a UML (*Unified Modeling Language*). Esta seção aborda os estilos e padrões arquiteturais e linguagens que podem ser empregadas para a descrição de arquiteturas. As DSSAs e os *frameworks* são abordados na seção 2.3, que descreve questões relacionadas à reutilização de software.

2.2.1.2 – Estilos e Padrões Arquiteturais

Os estilos e padrões arquiteturais traduzem organizações estruturais conhecidas para sistemas de software, estabelecendo uma topologia para a arquitetura e restrições sobre os componentes e conectores. Diversos autores catalogaram estilos arquiteturais que caracterizam famílias de sistemas de software em termos da sua organização estrutural (SHAW e GARLAN, 1996; MENDES, 2002). A Tabela 2.1 apresenta um resumo de alguns estilos arquiteturais catalogados que são mencionados nesta tese, apontando, para cada um deles, benefícios e contrapartidas na sua escolha como direcionadores para uma dada organização arquitetural.

Os estilos arquiteturais definem quatro importantes propriedades 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.

Tabela 2.1: Estilos arquiteturais e suas principais características.

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 à reutilização, manutenção e evolução do esquema.	Degradação do desempenho e complexidade de implemen-tação.
Dutos e Filtros	Filtros: transformadores de dados.	Dutos (ou *pipes*): transmissores de cadeias de dados (*streams*).	Filtros são independentes.	Os dados e o controle fluem de um filtro a outro através dos dutos, seguindo uma topologia linear. Em alguns casos pode apresentar ciclos.	Suporte à reutilização, à evolução e manutenção do esquema.	Inadequado para sistemas interativos.
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 represen-tação 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. Decomposição de problemas em uma coleção de agentes colaboradores.	Objetos precisam conhecer a identidade dos outros. Modificações na interface dos objetos afetam seus clientes.
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.
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 de 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.

Tabela 2.1: Estilos arquiteturais e suas principais características (continuação).

Estilo	Componentes	Conectores	Regras	Questões de Controle e de Dados	Benefícios	Contrapar-tidas
Interpretadores	Máquina de interpretação, memória contendo o pseudo-programa a ser interpretado, representação do estado de controle do interpretador, representação do estado corrente do pseudo-programa.	Dados de acesso direto e chamada de procedimento.	Uso de transição de estados para a execução da máquina e	Os dados do pseudo-programa sendo interpretado ficam em memória. Orientada à entrada para a seleção do que interpretar.	Aumentam a portabilidade do programa; facilitam a descrição de funcionalidades.	Necessidade de configurar o interpretador (i.e. máquina virtual) a cada nova linguagem ou plataforma; dificuldade de construção de novos componentes; problemas de desempenho.
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.

Os padrões arquiteturais estão catalogados em (BUSCHMANN *et al.*, 1996), sendo muito semelhantes aos estilos arquiteturais no sentido de proporem uma organização global para o sistema de software e restrições estruturais. Alguns autores argumentam que não existe diferença, e que estilos e padrões arquiteturais são essencialmente a mesma coisa (GORTON, 2006).

Dentre os padrões arquiteturais catalogados em (BUSCHMANN *et al.*, 1996), se encontram:

- **MVC (*Model-View-Controller*):** a arquitetura é dividida em três tipos de componentes, que possuem responsabilidades bem definidas, a saber: modelo, visão e controlador. O modelo encapsula os dados e a funcionalidade do negócio, sendo independente de representações de saída e tratamento das interações dos usuários com a aplicação; a visão mostra as informações para os usuários, sendo responsável pelos formatos de saída específicos e obtendo seus dados a partir do modelo; e o controlador trata os eventos de entrada dos usuários disparados nas interfaces, que são traduzidos para requisições de serviços ao modelo ou à visão.
- ***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.

- ***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.

Um aspecto importante a ser ressaltado em relação aos estilos e padrões arquiteturais, é que normalmente os estilos e padrões são utilizados de forma combinada, formando arquiteturas heterogêneas (SHAW e GARLAN, 1996). Essa combinação exerce forte impacto em abordagens de recuperação de arquitetura baseadas na detecção de estilos e padrões arquiteturais (HARRIS *et al.*, 1995; HARRIS *et al.*, 1997a; GALL e PINZGER, 2002), que serão apresentadas no capítulo 3.

2.2.1.3 – Linguagens para a Descrição de Arquiteturas

Uma forma de representar arquiteturas de software 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 ADLs visam buscar uma representação mais formal para as arquiteturas de software do que os diagramas informais comumente usados.

Uma ADL deve descrever os elementos arquiteturais e os conectores em uma arquitetura. Além de descrever os elementos arquiteturais e conectores, uma ADL deve permitir descrever o sistema como uma composição de componentes e conexões independentes, descrevendo os componentes e os conectores em um alto nível de abstração, sem se deter a detalhes de implementação, permitindo a reutilização de padrões de relacionamento componente-conector (como uma estrutura cliente-servidor) em descrições arquiteturais diferentes daquela para a qual foram originalmente especificados, além de outros aspectos de uma descrição arquitetural (SHAW e GARLAN, 1996). MEDVIDOVIC e TAYLOR (2000) apresentam um *framework* para a classificação e comparação de ADLs.

Um dos maiores objetivos do uso de uma ADL é a possibilidade de simular e avaliar as especificações arquiteturais e detectar erros o quanto antes no ciclo de vida.

Além das ADLs, linguagens de modelagem como a UML podem ser empregadas para a representação de arquiteturas. Embora a UML apresente um menor formalismo de especificação do que as ADLs, dificultando a realização de simulações e avaliações automatizadas, ela apresenta como vantagem o fato de representar um padrão acadêmico e industrial para a modelagem de software OO, facilitando a comunicação das decisões arquiteturais entre os diferentes *stakeholders*. Além disso, diversas ferramentas e ambientes são compatíveis com a UML. Em função desses fatores, a UML é adotada para a representação das arquiteturas recuperadas nesta tese.

Finalmente, segundo GORTON (2006), há uma série de vantagens em se documentar arquiteturas de software, como: outros desenvolvedores podem compreender e avaliar o seu projeto; arquitetos podem compreender o seu projeto quando retornam a ele depois de um período de tempo; outros desenvolvedores da organização podem aprender com os projetos arquiteturais; análises podem ser realizadas sobre o projeto, como análise de desempenho. Porém, GORTON (2006) também argumenta que documentar arquiteturas pode se tornar problemático pelos seguintes fatores: não há um padrão de documentação arquitetural aceito universalmente; uma arquitetura pode ser complexa e documentá-la de forma compreensível pode exigir grande esforço, além de não ser tarefa trivial; uma arquitetura possui muitas visões possíveis; uma arquitetura evolui à medida que o sistema é incrementalmente desenvolvido e o domínio do problema se torna mais compreendido e manter essa documentação atualizada diante de pressões de cronograma e orçamento acaba não sendo prioridade nos projetos. Nesse contexto, a ER pode ajudar a manter uma documentação arquitetural atualizada.

2.2.2 – Avaliação de Arquiteturas

Segundo BARCELOS (2006), o principal objetivo de uma avaliação arquitetural consiste em verificar se as informações contidas no documento que descreve a arquitetura estão consistentes e se a arquitetura nele representada atende aos requisitos especificados para o software. Conforme mencionado no início do capítulo, em abordagens de reutilização de software a preocupação com a qualidade da arquitetura deve ser ainda maior, pois os seus defeitos ou qualidade são repassados para as diversas aplicações instanciadas a partir do domínio.

BARCELOS (2006) estabelece um *framework* para a caracterização das abordagens de avaliação arquitetural e realiza uma revisão sistemática

(KITCHENHAM, 2004) acerca dessas abordagens. Dentre as características utilizadas para avaliar as abordagens, o autor destaca a importância da simplicidade e custo de aplicação do método, a extensibilidade e adaptabilidade da abordagem e a sua generalidade. Entretanto, durante a sua pesquisa não foi encontrada nenhuma abordagem que atendesse a todos esses requisitos. Abordagens baseadas em cenários, como ATAM (KAZMAN *et al.*, 2000) e SAAM (KAZMAN *et al.*, 1994), apresentam alto custo de aplicação, outras abordagens como SAR (ERICKSON *et al.*, 1993) e PASA (WILLIAMS e SMITH, 1998) são específicas para a avaliação de determinados atributos de qualidade, e abordagens como SAEM (DUENAS *et al.*, 1998) e FAV (LICHTNER *et al.*, 2000) são específicas para determinadas representações arquiteturais.

Dessa forma, a fim de suprir as carências encontradas nas abordagens de avaliação de arquitetura segundo o *framework* de critérios estabelecido, a abordagem de avaliação arquitetural ArqCheck foi desenvolvida (BARCELOS, 2006). ArqCheck se constitui em uma abordagem de avaliação de arquiteturas através da inspeção de documentos arquiteturais, utilizando como técnica de detecção de defeitos na arquitetura um *checklist*. Através da técnica de *checklist*, busca-se atingir simplicidade e generalidade no método de avaliação.

Os itens de detecção de defeitos arquiteturais que compõem o *checklist* de ArqCheck estão divididos em 3 categorias, a saber: itens que identificam defeitos na consistência da representação dos elementos arquiteturais entre os diferentes modelos que compõem a arquitetura; itens que identificam defeitos referentes ao atendimento a requisitos funcionais; e itens que identificam defeitos referentes ao atendimento a requisitos não-funcionais (ou atributos de qualidade). ArqCheck considera os atributos de qualidade que, segundo BASS *et al*. (2003), são relevantes a nível arquitetural, i.e. Desempenho, Disponibilidade, Modificabilidade, Usabilidade, Segurança e Testabilidade.

Os defeitos identificados com ArqCheck, dentro das categorias mencionadas, devem ainda ser classificados pelos inspetores segundo a taxonomia definida por SHULL (1998) e adaptada para o contexto arquitetural, envolvendo os seguintes tipos de defeitos: omissão, ambigüidade, inconsistência, fato incorreto e informação estranha. Essa taxonomia é não ortogonal, onde o mesmo defeito pode pertencer a mais de um tipo. Uma descrição desses tipos de defeitos no contexto de ArqCheck pode ser encontrada no Anexo C/C.4.

A fim de evitar a subjetividade e tornar as questões do *checklist* de ArqCheck configuráveis a diferentes representações arquiteturais, projetos ou organizações, os itens que compõem o *checklist* foram definidos com base em conhecimentos obtidos dos principais conceitos e técnicas de projeto arquitetural (SHAW, 1996; BOSCH e MOLIN, 1999; BASS *et al.*, 2003) e com base em abordagens de documentação arquitetural descritas na literatura (IEEE, 2000), descrevendo informações que devem estar presentes em um documento arquitetural, independente da representação adotada.

Os itens de avaliação do atendimento a requisitos não-funcionais, por exemplo, estão baseados no conhecimento presente em táticas arquiteturais. As táticas arquiteturais consistem em um conjunto de diretrizes arquiteturais baseadas em boas práticas, que auxiliam no atendimento a atributos de qualidade que influenciam o projeto de uma arquitetura (TVEDT *et al.*, 2002). As táticas utilizadas em ArqCheck estão descritas em (BASS *et al.*, 2003).

Uma versão simplificada do *checklist* de ArqCheck é apresentada no Anexo G. Essa versão foi adaptada à documentação arquitetural do sistema utilizado em um dos estudos de viabilidade de ArqCheck realizado em (BARCELOS, 2006). Convém ressaltar que a fim de atingir generalidade em relação à documentação arquitetural, o *checklist* de ArqCheck deve ser configurado (adaptado) à documentação arquitetural avaliada, através da alteração dos termos arquiteturais utilizados, classificação das questões em aplicáveis ou não (coluna NA no *checklist*) e instanciação da descrição e cenário de cada requisito de qualidade de interesse.

Por ser uma abordagem de avaliação arquitetural de simples utilização e configurável a diferentes representações arquiteturais, a abordagem ArqCheck é adotada nesta tese para a avaliação das arquiteturas recuperadas, como será visto no capítulo 4.

2.3 – Abordagens de Reutilização de Software

A reutilização de software caracteriza-se pela utilização de produtos de software, construídos ao longo do processo de desenvolvimento, em uma situação diferente daquela para a qual foram originalmente produzidos (FREEMAN, 1980). Ou, pode-se dizer que a reutilização representa o uso de qualquer informação já disponível que um desenvolvedor pode necessitar no processo de criação de software (FREEMAN, 1987). Entende-se por "qualquer informação já disponível", produtos do desenvolvimento de software, como componentes de código, artefatos de análise e projeto, planos de teste,

manuais etc., ou conhecimento armazenado e catalogado na área de Engenharia de Software, como os padrões de projeto e *frameworks* (GAMMA *et al.*, 1995).

Uma vez que a reutilização envolve o uso de conhecimento catalogado ou de artefatos que já foram testados e utilizados previamente, acredita-se que ela possa melhorar a qualidade do software desenvolvido. Além da qualidade, a reutilização traz consigo a perspectiva de aumento da produtividade, uma vez que o software não começa a ser desenvolvido "do zero" e que a correção de defeitos em um artefato pode ser benéfica a uma família de aplicações e não a um único produto.

A reutilização pode ser obtida através de diferentes paradigmas de desenvolvimento (ex: OO e DBC) e em diferentes fases do ciclo de vida do software. Abordagens como ED e LP visam alavancar a reutilização de software desde o início do seu ciclo de vida. Em todas as abordagens, a arquitetura é a mola mestra, i.e. a base para propiciar a reutilização dos artefatos, que, de uma forma geral, devem estar compatíveis com a estrutura estabelecida para o sistema ou para a família de sistemas.

Nesta seção, são apresentadas as abordagens de ED e LP e o seu apoio ao estabelecimento de uma arquitetura de referência de domínio.

2.3.1 – Engenharia de Domínio

A ED representa o desenvolvimento de artefatos de software para um domínio de aplicação ou família de sistemas, que estejam compatíveis com os requisitos do domínio. Enquanto a ED cobre o desenvolvimento para reutilização, através da especificação de artefatos para uma família de aplicações, a Engenharia de Aplicação (EA) constrói aplicações com base na instanciação dos artefatos produzidos pela ED, caracterizando o desenvolvimento com reutilização. Segundo o SEI/CMU (2005), "a EA desenvolve produtos de software baseados nos artefatos gerados pelo processo de ED".

A ED produz artefatos reutilizáveis, em diferentes níveis de abstração, a fim de apoiar a construção de aplicações em um dado domínio. O objetivo da ED é que a reutilização seja atingida desde o início do ciclo de vida do software e, nesse sentido, os métodos de ED, em geral, cobrem as etapas de **Planejamento**, **Análise**, **Projeto** e **Implementação** do domínio. Alguns dos métodos de ED conhecidos na literatura são: FODA (KANG *et al.*, 1990), FORM (KANG *et al.*, 2002), FODACom (VICI e ARGENTIERI, 1998), FeatuRSEB (GRISS *et al.*, 1998), ODM (SIMOS e ANTHONY, 1998), Odyssey-DE (BRAGA, 2000) e CBD-Arch-DE (BLOIS, 2006).

A etapa de **Planejamento do Domínio** tem como objetivo prover um conjunto de critérios que permitam ao Gerente de Domínio fazer estimativas razoáveis em termos de recursos a serem utilizados na ED, custos e cronograma (BRAGA, 2000). Além disso, é realizada a análise de viabilidade do domínio, que é uma atividade relacionada à decisão efetiva sobre a realização da ED naquele domínio.

Durante a **Análise de Domínio**, a ED se preocupa principalmente em produzir modelos de requisitos que reflitam as semelhanças e diferenças entre as aplicações do domínio. As semelhanças e diferenças podem ser representadas através de opcionalidades e variabilidades nos artefatos do domínio (OLIVEIRA, 2006b). O modelo mais comumente utilizado para a modelagem de variabilidades e opcionalidades é o modelo de características (*features*).

Segundo a notação Odyssey-FEX (OLIVEIRA, 2006b) para a modelagem de opcionalidades e variabilidades, adotada nesta tese, uma característica do domínio pode representar, sob o ponto de vista da sua variabilidade: um **ponto de variação** (VP), que reflete a parametrização do domínio através da seleção de uma ou mais variantes; uma **variante**, que reflete funções ou conceitos ligados a um ponto de variação, atuando como suas alternativas de configuração; ou **invariantes**, elementos fixos, não configuráveis no domínio. Sob o ponto de vista da sua opcionalidade, as características na Odyssey-FEX podem ser classificadas como: **mandatórias**, elementos que necessariamente devem ser instanciados para todas as aplicações do domínio; ou **opcionais**, elementos que podem ou não compor uma aplicação derivada do domínio. Essas classificações são ortogonais, ou seja, uma característica variante, invariante ou ponto de variação pode ser ao mesmo tempo opcional ou mandatória.

A escolha da Odyssey-FEX, nesta tese, se deve ao fato dela ter sido desenvolvida com base na análise de notações de variabilidade existentes, tentando suprir os pontos falhos nas mesmas (OLIVEIRA, 2006b). Além disso, a Odyssey-FEX prevê a modelagem de variabilidades nos diferentes modelos de domínio. Finalmente, a Odyssey-FEX foi desenvolvida no mesmo contexto desta tese, i.e. no contexto do projeto Odyssey, e o seu desenvolvimento também foi apoiado pelo trabalho de pesquisa desenvolvido nesta tese (OLIVEIRA *et al.*, 2005b; OLIVEIRA *et al.*, 2005c).

A etapa de **Projeto de Domínio** visa, principalmente, a especificação de arquiteturas de software específicas de domínio (DSSAs – *Domain Specific Software Architectures*) com base nos requisitos, variabilidades e opcionalidades levantados para o domínio. Uma DSSA representa uma coleção de componentes especializados para um

determinado tipo de tarefa (domínio), generalizados para que seja possível seu uso efetivo através do domínio e compostos em uma estrutura padronizada (topologia) efetiva para a construção de aplicações (HAYES, 1994). Uma DSSA[4] compreende vários elementos, como os requisitos de referência, i.e. funcionais e não-funcionais do domínio, sendo que a arquitetura de referência de domínio é o seu elemento principal. Arquiteturas de referência são criadas para serem configuradas e estendidas durante o processo de instanciação de aplicações, representando um *framework* para o domínio.

A última etapa da ED é a etapa de **Implementação do Domínio**. Essa etapa envolve a transformação das oportunidades de reutilização, identificadas na análise, e das soluções de projeto em um modelo implementacional, envolvendo a identificação, construção ou extração de componentes reutilizáveis no domínio.

Nesta tese, o objetivo é apoiar mais fortemente a etapa de Projeto de Domínio, através da análise das arquiteturas de um conjunto de sistemas legados no domínio para a criação de arquiteturas de referência. As abordagens de ED pesquisadas, embora considerando os sistemas legados como uma das fontes de informação para a Análise de Domínio, não oferecem apoio sistemático à sua análise. A especificação de DSSAs é apoiada através de métodos de engenharia progressiva, i.e. de cima para baixo (*top-down*). Além disso, alguns métodos de ED pesquisados dão maior apoio à etapa de Análise de Domínio do que à etapa de Projeto de Domínio (i.e. o FODA (KANG *et al.*, 1990), o FeatuRSEB (GRISS *et al.*, 1998), o ODM (SIMOS e ANTHONY, 1998) e o FODACom (VICI e ARGENTIERI, 1998)).

BLOIS (2006), por exemplo, propõe um método de ED, o CBD-Arch-DE, que visa oferecer um maior apoio à etapa de Projeto de Domínio do que outros métodos de ED pesquisados. No CBD-Arch-DE, o modelo de características do domínio é mapeado para os modelos de mais baixo nível de abstração, como modelos de casos de uso e modelos de tipos de negócio, propiciando a geração de componentes de processo, de negócio, utilitários e de infra-estrutura do domínio, conforme mostra a Figura 2.3. As variabilidades são propagadas do modelo de características para os demais modelos, sendo representadas inclusive no modelo de componentes de software. O objetivo do método é apoiar a criação de arquiteturas de referência do domínio através de um processo de ED com suporte a DBC.

[4] A partir deste ponto da tese, os termos arquitetura de referência de domínio e DSSA serão utilizados indiscriminadamente como termos sinônimos.

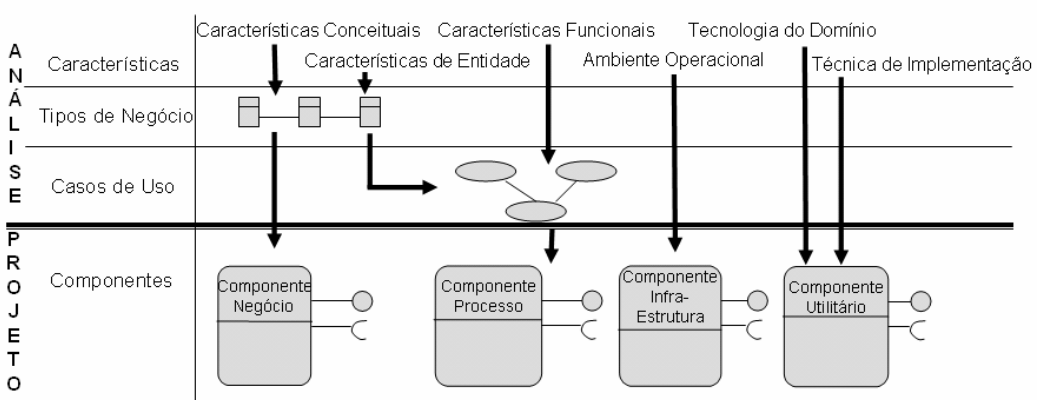

Figura 2.3: Artefatos do domínio e seus mapeamentos no CBD-Arch-DE. Extraída de (BLOIS, 2006).

Outros métodos de criação e instanciação de DSSA propostos na literatura, que visam suprir a falta desse apoio em alguns métodos de ED, são o projeto ARPA (METTALA e GRAHAM, 1992) e a abordagem implementada na ferramenta MENTOR (XAVIER, 2001), integrada ao ambiente de reutilização Odyssey (ODYSSEY, 2007).

Em (HAYES, 1994; TRACZ e HAYES, 1994) são descritos os processos de criação e instanciação de DSSA resultantes do projeto de pesquisa ARPA (METTALA e GRAHAM, 1992). A criação da DSSA envolve 3 elementos de informação principais: o modelo do domínio, os requisitos de referência e a arquitetura de referência. Os requisitos de referência são utilizados pelo Engenheiro de Domínio para a criação da arquitetura de referência, sendo divididos em requisitos funcionais, que definem o espaço do problema, e não-funcionais, que restringem o espaço-solução. Pelo menos uma arquitetura de referência que satisfaça às capacidades funcionais e não-funcionais do domínio deve ser especificada. Entretanto, os autores ressaltam que nem todos os requisitos são satisfeitos por uma única arquitetura, sendo possível a criação de mais de uma arquitetura de referência para o domínio.

Assim como na especificação da arquitetura de uma aplicação específica, a elaboração da arquitetura de referência também pode ser guiada pela seleção de um estilo ou padrão arquitetural. XAVIER (2001), por exemplo, leva em conta os requisitos não-funcionais privilegiados por um estilo ou padrão arquitetural (veja Tabela 2.1) e os requisitos não-funcionais priorizados para o domínio para sugerir padrões arquiteturais adequados à criação da arquitetura de referência. Através da sua ferramenta MENTOR, integrada ao ambiente Odyssey, ele calcula a distância entre cada padrão arquitetural e a solução computacional adequada ao domínio, estabelecendo um espaço vetorial com os

requisitos não-funcionais do domínio, em ordem de prioridade, e os requisitos priorizados pelos padrões arquiteturais. Com base nesse cálculo, a ferramenta indica os padrões mais adequados, em ordem de prioridade, para guiar a solução computacional para o domínio.

Em (IVKOVIC e GODFREY, 2003), os autores propõem a organização de uma base de conhecimento sobre o domínio, denominada ADAS (*Architectural Domain Assets Set*), baseada na combinação de conhecimento obtido com engenharia reversa e engenharia progressiva. Uma ADAS deve incluir informações sobre as características do domínio e suas variabilidades, tecnologias e teorias sobre o domínio, escopo, atributos de qualidade e histórico da evolução do domínio. Os autores argumentam que uma abordagem de recuperação de arquitetura para ser eficiente e justificar seu custo-benefício deve ser voltada às características específicas do domínio (ex: sistemas distribuídos), apoiada pela base de conhecimento organizada para o domínio e ser reutilizada para diversas aplicações do domínio ou várias versões de uma mesma aplicação. Entretanto, os autores não propõem um método de recuperação de arquitetura, mas apenas oferecem diretrizes para apoiá-la.

Dessa forma, percebe-se que no contexto das abordagens de ED, um apoio mais efetivo à análise de sistemas legados para apoiar a modelagem de domínio pode ser de grande utilidade, visto que as abordagens, em geral, oferecem um maior apoio no sentido da engenharia progressiva do que da engenharia reversa.

2.3.2 – Linha de Produtos

Assim como a ED, a **Linha de Produtos de Software** (LP) define um paradigma de desenvolvimento baseado em reutilização, cujo propósito é guiar organizações no desenvolvimento para e com reutilização. Normalmente, as LPs nascem dos produtos de software similares existentes na empresa, referentes a uma área comum de negócio, que podem se beneficiar da migração para uma abordagem de desenvolvimento de software baseada na reutilização. Segundo o SEI (*Software Engineering Institute*) (SEI/CMU, 2005), uma LP representa sistemas de software que compartilham um conjunto de características comuns e controladas, que satisfazem necessidades de um segmento de mercado em particular, e são desenvolvidos a partir de artefatos-chave (*core assets*), de forma predefinida.

Enquanto na ED, tem-se as arquiteturas de referência, elementos-chave das DSSAs, nas LPs tem-se as *Product Line Architectures*. Os princípios de construção são

os mesmos, uma vez que ambas representam arquiteturas genéricas ou *frameworks* para uma família de aplicações, construídas com base nos requisitos e variabilidades de uma linha de produtos ou domínio. A construção da PLA compõe a atividade de **Desenvolvimento do Núcleo de Artefatos** das LPs, que além dessa atividade, engloba também o **Desenvolvimento do Produto** e o **Gerenciamento da Linha de Produtos,** conforme apresentado na Figura 2.4. Estabelecendo um paralelo com a ED, o **Desenvolvimento do Núcleo de Artefatos** engloba as atividades de Análise, Projeto e Implementação da ED e o **Desenvolvimento do Produto** corresponde à instanciação de aplicações da EA, existindo uma retro-alimentação entre essas atividades, numa abordagem evolutiva, conforme mostra a Figura 2.4. Maiores detalhes sobre processos de LP podem ser obtidos em (GIMENES e TRAVASSOS, 2002; OLIVEIRA, 2006b).

A fim de verificar o apoio existente ao desenvolvimento do núcleo de artefatos das LPs e, conseqüentemente, à criação de arquiteturas de linhas de produtos, algumas abordagens de LP da literatura foram pesquisadas.

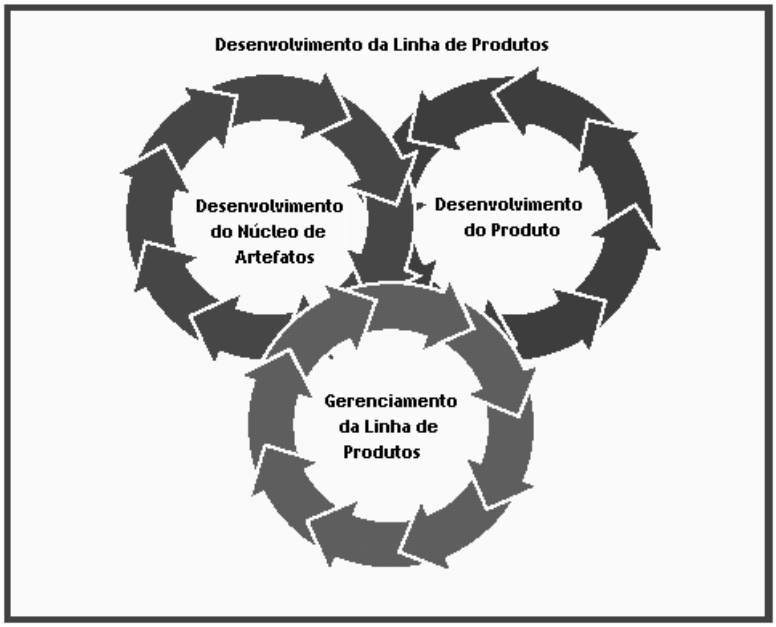

Figura 2.4: Atividades Essenciais da Linha de Produtos. Adaptada de (NORTHROP, 2002).

Nesse contexto, em (GOMAA, 2004), é proposto o método PLUS de desenvolvimento de linhas de produtos que parte da especificação dos casos de uso de uma LP, passando pelo modelo de características, modelagem dinâmica através de diagramas de interação e de máquinas de estado, modelagem de classes, modelagem de contexto (i.e. objetos externos ao sistema e a sua comunicação com objetos da LP) e modelagem dos componentes de software na arquitetura, onde um padrão arquitetural

deve ser adotado. A determinação das variabilidades no PLUS se inicia nos casos de uso, sendo propagada para os demais modelos. O autor comenta que o desenvolvimento é iterativo e evolutivo e que pode ser adotada uma abordagem de engenharia progressiva (*forward evolutionary engineering*) ou reversa (*reverse evolutionary engineering*). Entretanto, no que diz respeito à ER, embora o autor mencione algumas diretrizes que podem ser adotadas a fim de realizar a consistência entre os modelos e detecção de variabilidades, a necessidade de um suporte mais efetivo fica evidente. Por exemplo, o autor não indica como os modelos podem ser extraídos dos sistemas existentes e não leva em conta questões como diferenças de nomenclaturas entre os diferentes sistemas ou diferenças estruturais entre os modelos.

Em (ALVES *et al.*, 2005; ALVES *et al.*, 2007), uma abordagem extrativa (i.e. *extractive*) e evolucionária (i.e. *reactive*) para a criação de LPs é desenvolvida. As LPs surgem a partir de um conjunto de produtos existentes (i.e. abordagem extrativa) e vai evoluindo à medida que novos produtos são acrescidos à linha (i.e. abordagem evolucionária). O foco da abordagem está na refatoração dos produtos através do uso de aspectos (AOP – *Aspect Oriented Programming* (KICZALES *et al.*, 1997), que representam as características variáveis dos produtos da linha, as quais refletem prioritariamente requisitos não-funcionais. A detecção das partes variáveis das LPs é feita através da elaboração de um gráfico de conceitos, que demonstra a sua relação com elementos do código fonte. Através desse gráfico, partes variáveis do código fonte, a serem refatoradas para aspectos, devem ser identificadas. Entretanto, as guias oferecidas para a construção do gráfico de conceitos são superficiais e não oferecem subsídios adequados à sua aplicação, além de não ser oferecido suporte à detecção das variabilidades entre os elementos do código que suportam esses conceitos. O foco do trabalho está de fato na refatoração e representação das variabilidades através de aspectos e não na sua detecção. Por fim, vale ressaltar que o tratamento de variabilidades nesse trabalho é em nível de código, ao passo que na abordagem desenvolvida nesta tese o seu tratamento é em nível arquitetural.

Em (STOERMER e O'BRIEN, 2001), é proposto o método MAP (*Mining Architectures for Product Lines*), para a migração de produtos individuais de um mesmo domínio ou de domínios similares para LPs. O método é composto de 5 etapas, envolvendo: a Preparação, i.e. o planejamento para a migração, através da análise de disponibilidade de recursos na empresa, questões organizacionais para suporte a LPs e seleção dos produtos candidatos; a Extração, i.e. a recuperação de modelos estáticos e

dinâmicos dos produtos a partir do código ou executável; a Composição e a Qualificação, responsáveis por elevar o nível de abstração dos modelos extraídos para o nível arquitetural; a Avaliação, onde variabilidades entre os produtos da linha são identificadas no nível arquitetural; e Atividades de Pós-Avaliação, que são conduzidas a fim de decidir sobre a criação de uma LP com base nas variabilidades identificadas. A Composição é realizada no ambiente de recuperação de arquitetura Dali (KAZMAN e CARRIÈRE, 1997).

No MAP, os autores argumentam que diferenças em nível de estilo arquitetural e atributos de qualidade podem comprometer a viabilidade de criação das LPs. São também consideradas na comparação dos produtos, características como funcionalidades específicas de clientes, protocolos, sistemas operacionais e plataforma de hardware. Estabelecendo um paralelo entre o MAP e a abordagem proposta nesta tese, percebe-se que o método MAP é mais completo, uma vez que cobre atividades de planejamento, análise de viabilidade e atividades de pós-avaliação. Entretanto, os critérios de composição para a recuperação da arquitetura são altamente dependentes de conhecimento humano acerca da estrutura do produto ou domínio e, embora a etapa de Avaliação estabeleça a partir de que características as arquiteturas devem ser comparadas, não é comentada a existência de qualquer suporte a essa comparação.

Em (CORNELISSEN *et al.*, 2005), é proposta a identificação de pontos de variação entre diferentes versões de um produto através da análise de rastros de execução (i.e. seqüências de métodos chamados em tempo de execução). A abordagem executa o mesmo cenário em diferentes versões de um produto - considerando as suas variações entre as diferentes versões - e extrai rastros de execução. A identificação de bifurcações (*forks*) nos rastros, ou seja, pontos onde eles se desviam, identifica a presença de características variantes entre os produtos. A identificação de pontos de variação e variantes é, portanto, realizada em nível de chamadas de métodos nos rastros de execução, ou seja, em nível funcional. Nesta tese, conforme será visto no capítulo 4, a identificação de variabilidades e opcionalidades entre os sistemas legados do domínio é realizada em nível estrutural na arquitetura.

Em (GANESAN e KNODEL, 2005), é apresentada uma abordagem para a identificação de componentes de software de domínio baseada em atributos de qualidade e métricas. A abordagem, aplicada a sistemas OO, é semi-automática e a proposta é reduzir a quantidade de candidatos a componentes de software reutilizáveis que devem ser avaliados pelos especialistas das aplicações. Os candidatos são

selecionados com base em um conjunto de atributos de qualidade desejáveis em um componente reutilizável (CALDIERA e BASILI, 1991), como utilidade funcional, corretude, testabilidade, custo de identificação e modificação etc., que são mapeados para métricas de software (ex: WMC – complexidade ciclomática, LCOM – coesão de classes, DIT – profundidade na hierarquia de herança), através do paradigma GQM (*Goal Question Metric*) de BASILI *et al.* (1994). Entretanto, não há estudos que comprovem se as métricas apresentam eficácia na indicação de valores para os atributos de qualidade, os quais não podem ser medidos diretamente, e como estabelecer limites máximos e mínimos para eliminação de candidatos com base nos valores das métricas. Se por um lado, a abordagem visa reduzir a quantidade de candidatos a componentes de software que o especialista deve avaliar, por outro lado, ela pode acabar eliminando, equivocadamente, candidatos que seriam de interesse. Além disso, a avaliação é feita em nível de classe e os autores indicam que os componentes de software devem ser compostos com base nas classes identificadas como "reutilizáveis". Entretanto, diretrizes para a montagem dos componentes não são oferecidas.

Em (KNODEL e MUTHIG, 2005), é analisada a adequação de componentes de sistemas existentes à sua reutilização em LPs. A análise de adequação é realizada empregando-se técnicas de ER para a avaliação dos seguintes itens: a qualidade interna dos componentes, a sua adequação à sua documentação e a sua conformidade à arquitetura de uma LP. Dentre as técnicas de ER empregadas, encontram-se a análise estática do código para a detecção de código repetido (i.e. *clone detection*) e a extração de métricas sobre o código. A detecção de código repetido e a extração de métricas (como acoplamento, tamanho e complexidade) visam verificar de uma forma indireta o grau de manutenibilidade dos componentes e, conseqüentemente, a sua qualidade interna. A qualidade dos componentes também é avaliada verificando-se se convenções de nomenclatura em identificadores são respeitadas e o percentual de linhas de código comentadas, questões que também impactam a manutenibilidade dos componentes. A análise de adequação à sua documentação é realizada através do emprego de ferramentas de análise estática que fazem análises reflexivas, através do mapeamento do modelo para o código. Uma vez que o componente seja julgado como adequado pelos engenheiros da LP, um trabalho de reengenharia deve ser conduzido para adaptá-lo à arquitetura e requisitos da LP.

O trabalho ressalta 2 questões importantes para a reutilização de artefatos extraídos de sistemas existentes, a saber: a necessidade de avaliação da qualidade desses

artefatos e do esforço em adaptá-los para a sua reutilização em uma arquitetura de referência; e a necessidade de refatoração, adaptação ou reengenharia de componentes existentes para a sua adequação a arquiteturas de referência de domínio ou de LPs. Entretanto, não contempla o apoio à especificação de arquiteturas de referência de domínio, uma vez que assume a existência da arquitetura para a extração pontual de componentes que possam preencher essa arquitetura, oferecendo, dessa forma, apoio à fase de implementação na construção do núcleo de artefatos de LPs.

Finalmente, o método KobrA de ATKINSON *et al*. (2002), envolve uma abordagem de DBC com LP baseada em modelos UML. O foco é na engenharia progressiva, havendo uma forte distinção entre os níveis de abstração dos artefatos produzidos. O processo de DBC previsto pelo KobrA envolve as seguintes fases para a especificação de componentes de software: modelagem estrutural das classes dos componentes; modelagem funcional, descrevendo os efeitos externamente visíveis das operações fornecidas pelo componente; e modelagem comportamental, demonstrando como o componente se comporta em resposta a estímulos externos. Componentes especificados em um nível abstrato podem ser implementados por componentes de software, que sejam adquiridos de terceiros (i.e. COTS), provindos de um outro projeto ou extraídos de sistemas legados. Entretanto, o KobrA não trata da questão de busca ou extração de componentes de software, focando apenas na avaliação da conformidade dos componentes para a sua utilização na estrutura do projeto atual. A especificação de um componente, através dos modelos determinados pelo KobrA, pode ser utilizada para apoiar essa busca.

Quando o modelo de componentes especificado com o KobrA visa atender a uma família de produtos ou LP, o método prevê a construção de um *framework* de componentes, que representa a arquitetura da LP. A variabilidade na arquitetura é especificada através de um modelo de decisão, que consiste em um conjunto de tabelas onde cada requisito é descrito, indicando quais produtos da linha compreendem tal requisito e que outros requisitos são necessários para a sua utilização. Entretanto, não existe um apoio, através de técnicas ou ferramentas, para a associação dos produtos da LP aos requisitos nessas tabelas, sendo essa associação realizada de forma aleatória.

Dessa forma, pela análise das abordagens de LP, observa-se que algumas conduzem à especificação de arquiteturas de referência seguindo, prioritariamente, um direcionamento de cima para baixo, i.e. da análise para a implementação, no sentido da engenharia progressiva (i.e. (ATKINSON *et al.*, 2002; GOMAA, 2004)). Outras

abordagens, como as apresentadas em (CORNELISSEN *et al.*, 2005; GANESAN e KNODEL, 2005; KNODEL e MUTHIG, 2005; ALVES *et al.*, 2007), embora pautadas na Engenharia Reversa, focam em algum aspecto específico da análise de sistemas legados, como a extração de componentes de código legado, avaliação da adequação dos componentes às LPs ou refatoração de variabilidades. Assim, elas não propõem um método sistemático de extração e análise de modelos arquiteturais dos sistemas legados para se chegar na especificação das arquiteturas de referência.

O método MAP (STOERMER e O'BRIEN, 2001) é o mais abrangente no sentido de focar na ER para a especificação de arquiteturas de referência de domínio. Entretanto, os critérios para a reconstrução de elementos arquiteturais a partir de sistemas legados são altamente dependentes de conhecimento humano sobre a arquitetura das aplicações e não existe um apoio efetivo à comparação das arquiteturas reconstruídas para a detecção de variabilidades e opcionalidades na arquitetura de referência. Percebe-se, então, que contribuições nessa área ainda são necessárias para a comunidade de Engenharia de Software.

2.4 – Considerações Finais

Este capítulo apresentou conceitos e questões essenciais acerca de arquiteturas de software, destacando aspectos relacionados à sua representação e avaliação. Uma relação entre arquitetura e reutilização de software foi estabelecida, demonstrando a importância da arquitetura nesse contexto e o apoio oferecido por abordagens de reutilização à especificação de arquiteturas de referência de domínio.

Em relação às arquiteturas de referência de domínio, por essas apresentarem uma característica de generalização, buscando atender a uma família de aplicações, o custo de sua criação 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 criada a partir de uma DSSA (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 reduzir esses custos, a criação de DSSAs pode se beneficiar de sistemas legados disponíveis no domínio. Dentre as abordagens de ED e LP pesquisadas, observou-se que a maioria delas segue um direcionamento de cima para

baixo, i.e. no sentido da engenharia progressiva, para apoiar a especificação de arquiteturas de referência de domínio (ex: (XAVIER, 2001; ATKINSON *et al.*, 2002; GOMAA, 2004; BLOIS, 2006)). As abordagens que focam na ER, em geral, não oferecem um apoio sistemático à extração e análise de modelos dos sistemas legados para apoiar a criação de arquiteturas de referência, estando focadas em algum aspecto específico, como a extração ou avaliação da adequação de componentes (GANESAN e KNODEL, 2005; KNODEL e MUTHIG, 2005).

Nesse contexto, a abordagem proposta nesta tese visa beneficiar a ED e a LP, através do apoio à criação de arquiteturas de referência a partir de sistemas legados no domínio, fazendo uso da base de conhecimento disponível nesses sistemas e suprindo as carências identificadas nos métodos existentes.

Capítulo 3 – Engenharia Reversa e Comparação de Modelos

3.1 – Introdução

A fim de apoiar a criação de arquiteturas de referência de domínio, técnicas de Engenharia Reversa (ER) e de Gerência de Configuração, referentes à comparação de modelos, são empregadas nesta tese.

A Engenharia Reversa pode ser definida como: "o processo de análise de um sistema, a fim de identificar os componentes desse sistema e seus relacionamentos e criar representações do sistema em uma outra forma ou em um nível mais alto de abstração" (CHIKOFSKY e CROSS II, 1990). A ER, normalmente, objetiva a extração de documentação de projeto a partir do código, representando uma atividade ou um conjunto de técnicas que visam facilitar a compreensão de um software existente.

Em relação aos sistemas legados, a ER é de grande utilidade, pois esses sistemas costumam se encontrar em operação há anos nas organizações, sendo modificados ao longo do tempo sem que a sua documentação seja atualizada em paralelo. Assim, o código fonte acaba se tornando a única fonte de informação atualizada. Porém, o código se encontra em um baixo nível de abstração, sendo direcionado a ferramentas do gênero dos compiladores e as dimensões de um software são geralmente grandes (milhares de linhas de código), o que dificulta o seu entendimento sem o apoio de ferramentas de ER (VERONESE e NETTO, 2001).

Embora a ER conduza à extração de representações em um nível mais alto de abstração, podendo envolver a reconstrução de modelos de projeto ou de requisitos a partir de artefatos disponíveis do software, a ER de representações de projeto é mais comumente encontrada em trabalhos da literatura. Principalmente porque, quanto mais alto o nível de abstração, mais difícil se torna a obtenção de representações confiáveis a partir do código ou executável.

Diversas abordagens de recuperação de projeto (*Design Recovery*) (ex: (SEEMANN e GUDENBERG, 1998)) e remodularização (*Software Remodularization*) (ex: (MITCHELL e MANCORIDIS, 2006)) são propostas na literatura da área. A remodularização pode ser definida como a decomposição de um conjunto de componentes em sub-partes (i.e. os módulos) (ANQUETIL, 2000). Nesse caso, os

algoritmos costumam impor uma estrutura ao sistema existente, se baseando em técnicas de agrupamento (*clustering*). *Clustering* representa uma atividade-chave na ER para a descoberta de projetos "de melhor qualidade" do sistema ou para a extração de conceitos significativos a partir do código (ANQUETIL *et al.*, 1999). A remodularização acaba inferindo uma forma de reestruturação, que, segundo CHIKOFSKY e CROSS II (1990) envolve a transformação de uma representação para outra no mesmo nível de abstração, mantendo as funcionalidades do sistema.

Com a crescente importância que a área de arquitetura de software vem ganhando nos últimos anos, conforme apresentado no capítulo 2, a ER de arquitetura passou a ganhar destaque na comunidade, podendo ser definida como:

1) "Um conjunto de técnicas e processos utilizados para abstrair uma representação de mais alto nível (i.e. a arquitetura do software) a partir de informação disponível, como artefatos existentes (ex: código fonte, rastros de execução, documentação de projeto) e conhecimento de especialistas (ex: arquitetos de software e mantenedores)." (GALL e PINZGER, 2002)

2) "Uma atividade de Engenharia Reversa que visa à obtenção de uma arquitetura documentada para um sistema existente" (DEURSEN *et al.*, 2004).

3) "A descrição arquitetural de um software comunica decisões de projeto essenciais, as quais em relação aos sistemas legados foram tomadas no passado. Recuperar a arquitetura requer, portanto, a recuperação de decisões passadas de projeto, levando o desenvolvedor a inferir informações arquiteturais que não estão imediatamente evidentes" (RIVA e RODRIGUEZ, 2002).

A abordagem de recuperação de arquitetura proposta nesta tese está em consonância com a primeira definição apresentada. A fim de apoiar a recuperação de arquitetura, técnicas de análise estática e dinâmica de software são empregadas. Essas técnicas são combinadas a técnicas de mineração de dados (*Data Mining*).

Mineração de dados, ou descoberta de conhecimento em bases de dados (KDD - *Knowledge Discovery in DataBases*), representa o processo de extração de relações implícitas, ou de alto nível, a partir de um conjunto de informações relevantes (CHEN *et al.*, 1996). Técnicas de mineração de dados são empregadas, na abordagem proposta, para derivar conhecimento a partir dos modelos extraídos com a ER.

A abordagem proposta nesta tese aplica ainda técnicas de comparação de modelos, que normalmente são empregadas em trabalhos sobre manutenção e evolução de software (KEIENBURG e RAUSCH, 2001) da área de Gerência de Configuração

(NIERE, 2004; KELTER *et al.*, 2005). Nesta tese, o objetivo é comparar modelos extraídos de diferentes sistemas legados em um domínio, a fim de detectar as suas opcionalidades e variabilidades.

Partindo desta Introdução, o restante do capítulo está organizado da seguinte forma: a seção 3.2 explora as técnicas, atividades e trabalhos relacionados aos desta tese em ER, destacando as técnicas de análise estática e dinâmica, a atividade de recuperação de arquitetura e o emprego de técnicas de mineração de dados no contexto da ER; a seção 3.3 examina trabalhos de comparação de modelos para a detecção das suas diferenças, enfatizando a detecção de diferenças entre modelos arquiteturais; por fim, a seção 3.4 conclui o capítulo, apresentando algumas considerações finais.

3.2 – Engenharia Reversa

A ER é uma área extremamente ampla e explorada na Engenharia de Software, envolvendo trabalhos com diferentes objetivos, como: redocumentação de software, recuperação de projeto, remodularização, recuperação de arquitetura, localização de funcionalidade no código, extração de componentes etc. Nesse contexto, a recuperação de arquitetura é uma atividade que vem recebendo destaque por parte da comunidade nos últimos anos. A recuperação de arquitetura pode ser realizada com o apoio de técnicas de análise estática e análise dinâmica, detecção de padrões, reconhecimento de *clichés* (i.e. planos de programação), agrupamento de elementos, mineração de dados etc.

Como as técnicas de análise estática e análise dinâmica representam o ponto de partida para a aplicação das outras técnicas, elas são exploradas inicialmente nesta seção. É dado maior enfoque à análise dinâmica, que é a técnica mais explorada nesta tese. Técnicas de mineração de dados e as abordagens de ER que as aplicam são especialmente exploradas nesta seção, uma vez que a abordagem de ER proposta nesta tese também faz uso de uma técnica de mineração de dados, i.e. detecção de regras de associação (AGRAWAL e SRIKANT, 1994). Por fim, como a abordagem de ER proposta envolve a recuperação da arquitetura de sistemas legados, abordagens de recuperação de arquitetura, aplicando as técnicas mencionadas, são apresentadas, sendo destacadas as suas contribuições e pontos em aberto.

3.2.1 – Análise Estática e Análise Dinâmica

A fim de reconstruir modelos de projeto a partir de sistemas existentes, dois tipos de análise podem ser realizados para a extração de um conjunto de informações iniciais ou "fatos" do sistema, i.e. análise estática e análise dinâmica. A análise estática pode ser definida como a análise do código fonte para a extração de informações básicas do software, como acesso a variáveis e chamadas de procedimento, sendo normalmente realizada com o uso de um analisador sintático (*parser*[5]). A análise dinâmica, por outro lado, consiste na execução de um programa e monitoramento dos valores das variáveis, funções chamadas etc. (ANQUETIL, 2000). Para se descobrir que funções uma outra função pode chamar, a análise estática é suficiente, mas saber quais ela realmente chama, pode depender dos dados de entrada utilizados ou caminho escolhido para executar o programa, sendo necessária a análise dinâmica. A análise dinâmica é menos geral do que a estática, no sentido de que ela fornece informações sobre a chamada do programa com um conjunto de parâmetros particular. O mesmo programa com outros parâmetros pode fornecer outras informações (ANQUETIL, 2000).

As análises estática e dinâmica podem ser empregadas para a reconstrução de modelos estáticos e dinâmicos de projeto OO. Modelos estáticos revelam a estrutura do programa (ex: diagramas de pacotes e de classes da UML), enquanto os dinâmicos revelam o comportamento dos objetos para a realização de uma tarefa ou ao longo do tempo (ex: diagramas de seqüência, de colaboração e de estado da UML). Em relação aos modelos dinâmicos, nesta tese o foco é em modelos que demonstrem as interações entre objetos para a realização de uma tarefa, como os diagramas de seqüência e de colaboração da UML. Isso se deve ao fato de que esse tipo de modelo permite descrever o comportamento do sistema em sua arquitetura para a realização das suas funcionalidades. Os diagramas de seqüência são adotados por enfatizarem a seqüência temporal das mensagens.

A análise dinâmica é bastante empregada para a reconstrução de modelos dinâmicos para sistemas OO, em função de características específicas desse paradigma,

---

[5] *Parsing* é o processo de analisar uma seqüência de entrada (lida de um arquivo ou do teclado, por exemplo) para determinar a sua estrutura gramatical segundo uma determinada gramática formal. Esse processo é formalmente chamado de análise sintática. Um *parser* é um programa de computador que realiza essa tarefa (WIKIPÉDIA, 2007).

como a vinculação tardia de uma chamada ao método correspondente (*late binding*) e o polimorfismo, que só podem ser resolvidas em tempo de execução. Diversos trabalhos são propostos na literatura para a reconstrução de modelos dinâmicos para sistemas OO a partir da análise dinâmica (JERDING e RUGABER, 1997; DE PAUW *et al.*, 1998; SYSTÄ, 2000; RIVA e RODRIGUEZ, 2002; BRIAND *et al.*, 2003; TANIGUCHI *et al.*, 2005), embora a reconstrução desse tipo de modelo com a análise estática também seja possível (TONELLA e POTRICH, 2003; MAGICDRAW, 2007; TOGETHER, 2007), com alguma perda de informação. Na realidade, cada tipo de análise apresenta as suas vantagens e desvantagens. Se por um lado, através da análise dinâmica é possível resolver questões como referências polimórficas e vinculação tardia, por outro, cada execução do software exercita um cenário de uso específico, com um determinado conjunto de valores de entrada, gerando um caminho específico de execução da aplicação. Dessa forma, se torna difícil generalizar os modelos extraídos. Porém, para localizar a implementação de uma funcionalidade no código e analisar o impacto de uma manutenção, a análise dinâmica pode se mostrar bastante adequada, revelando as estruturas que de fato são executadas em um cenário que deve ser modificado.

A fim de realizar a análise dinâmica, cenários de uso que representam funcionalidades da aplicação devem ser executados e uma técnica de monitoramento de programa deve ser utilizada para permitir a coleta dos respectivos rastros de execução (*execution traces*), i.e. seqüências de métodos chamados em tempo de execução. Essa pode se basear na instrumentação do código fonte, código binário ou ambiente de execução do software (ex: máquina virtual Java), dependendo do artefato de programa disponível para análise e do nível de "intrusão" desejado, sendo que a instrumentação do código representa a técnica mais intrusiva de monitoramento, uma vez que modifica o código fonte. O sistema pode ainda ser executado sob o controle de um depurador (*debugger*). Essa técnica apresenta a vantagem de não modificar o código fonte, binário ou o ambiente de execução, entretanto, pode prejudicar consideravelmente o desempenho do sistema.

Um problema encontrado com a análise dinâmica é o grande volume de informação nos rastros de execução. As abordagens existentes (JERDING e RUGABER, 1997; DE PAUW *et al.*, 1998; WALKER *et al.*, 1998; SYSTÄ, 2000; RIVA e RODRIGUEZ, 2002) aplicam um conjunto de técnicas para reduzir o volume de informação nos rastros de execução, como:

- **Filtragem**: envolve a seleção dos objetos, componentes ou métodos que devem ser monitorados ou visualizados nos diagramas.
- **Detecção de padrões de interação:** padrões de interação (*interaction patterns*) ou padrões comportamentais (*behavioral patterns*) representam seqüências de mensagens que aparecem repetidamente nos rastros de execução. Caso essas seqüências sejam contíguas, elas podem representar um laço (*loop*) no programa, e, caso sejam não contíguas, podem representar um subcenário. Seqüências de chamadas em laços costumam ser representadas uma única vez, indicando-se no diagrama que existe um laço. Subcenários, por outro lado, costumam ser representados através de caixas nos diagramas de interação (i.e. diagramas de trocas de mensagens, como diagramas de seqüência da UML) e detalhados em diagramas a parte ou sob demanda.
- **Partição dos rastros de execução**: a partição dos rastros envolve a quebra de uma grande seqüência de chamadas de métodos em subseqüências menores, podendo ter como critério o cenário de caso de uso (i.e. funcionalidade do sistema) que cada subseqüência representa.
- **Ocultação de informação**: envolve o agrupamento de objetos ou classes no elemento arquitetural ao qual pertencem, ocultando a troca de mensagens entre eles no diagrama. Pode envolver também o agrupamento de mensagens em uma mensagem de mais alto nível.

A Tabela 3.1 apresenta um resumo de abordagens que reconstroem diagramas de interação para sistemas OO e suas técnicas de redução do volume de informação, resultante da análise dinâmica. A representação adotada para os modelos se assemelha aos diagramas de seqüência da UML. Entretanto, pelo fato das ferramentas que apóiam essas abordagens não estarem em sua maioria integradas a um ambiente de desenvolvimento, as abordagens, em geral, utilizam diagramas informais, representando simplificações dos diagramas de seqüência da UML. As ferramentas Ovation (DE PAUW *et al.*, 1998) e AVID (WALKER *et al.*, 1998), entretanto, seguem uma estrutura própria, representando os rastros através de uma estrutura de árvore e diagramas de caixas e arcos, respectivamente.

A ferramenta AVID (WALKER *et al.*, 1998) apresenta, além das técnicas abordadas na Tabela 3.1, a técnica de amostragem para a redução do volume de

informação, onde um subconjunto do rastro de execução é apresentado em função de parâmetros informados pelo usuário, como, por exemplo, a data e hora a partir da qual as chamadas de método devem ser apresentadas.

Tabela 3.1: Abordagens de análise dinâmica para a reconstrução de modelos dinâmicos e suas técnicas para redução do volume de informação.

Abordagens de Análise Dinâmica/Técnicas de Redução do Volume de Informação	Filtragem	Detecção de Padrões de Interação	Partição dos Rastros de Execução	Ocultação de Informação	Representação
Shimba (SYSTÄ, 2000)	**Sim**. Seleção de componentes.	**Sim**. Detectados através do algoritmo Boyer-Moore *string matching*.	**Não.**	**Não.**	Diagramas de cenário.
ISVis (JERDING e RUGABER, 1997)	**Sim**. Instrumentação seletiva de componentes, i.e. classes, funções etc.	**Sim**. Usuário pode guiar a busca por padrões.	**Não**.	**Sim**. Classes de um elemento arquitetural podem ser agrupadas.	Mural de Informação e diagrama de fluxo de mensagens temporais.
Rigi estendida (RIVA e RODRIGUEZ, 2002)	**Não.**	**Não.**	**Sim**, por caso de uso.	**Sim.** Permite agrupamento de mensagens e de objetos.	Diagramas de seqüência de mensagens (MSC – *message sequence charts*).
Ovation (DE PAUW *et al.*, 1998)	**Sim**. Usuário pode mostrar mensagens para um único objeto.	**Sim**. Algoritmo não informado.	**Não**.	**Sim**. Sub-árvores podem ser ocultadas.	Visão em forma de árvore.
AVID (WALKER *et al.*, 1998)	**Não.**	**Não.**	**Sim.** Rastros de execução quebrados em seqüências de visões, i,e. células.	**Sim.** Classes são agrupadas em elementos arquiteturais	Diagrama de caixas e arcos. Visão reduzida contendo todo o rastro.

Quanto à representação, a técnica utilizada pela ferramenta ISVis merece destaque. JERDING e RUGABER (1997) apresentam um mural de informação que contém a miniatura de todo o rastro da execução de uma sessão do programa, destacando os padrões de interação. O usuário pode selecionar uma parte do rastro e detalhá-la em uma escala maior.

A avaliação de ferramentas de análise dinâmica, a partir de outros conjuntos de critérios, pode ser encontrada em (PACIONE et al., 2003; HAMOU-LHADJ e LETHBRIDGE, 2004). Quanto à compatibilidade com a UML, a abordagem de BRIAND et al. (2003) apresenta regras de mapeamento de informações dos rastros de

execução para elementos do diagrama de seqüência, representando, por exemplo, condições e iterações nos diagramas. A abordagem de ROUNTEV (2004) é a única, dentre as pesquisadas, capaz de recuperar elementos do metamodelo da última versão da UML até o momento, i.e. a versão 2.0.

3.2.2 – Técnicas de Mineração de Dados no Contexto da Engenharia Reversa

A mineração de dados ou descoberta de conhecimento em banco de dados se refere a uma coleção de algoritmos para descobrir ou verificar relações interessantes e não-triviais entre dados em grandes bancos de dados (FAYYAD, 1996). Pode ainda ser definida como a exploração e a análise, por meio automático ou semi-automático, de grandes quantidades de dados, com a finalidade de descobrir padrões, i.e. regras significativas (BERRY e LINOFF, 1997).

Um número substancial de abordagens de mineração de dados na literatura é baseado em extensões do algoritmo Apriori de AGRAWAL e SRIKANT (1994) (SARTIPI, 2003). Essas abordagens foram motivadas por problemas de suporte à decisão em programas de *marketing* em grandes organizações de venda, que armazenam consideráveis volumes de informação sobre vendas, através de transações em bases de dados. O algoritmo Apriori minera as transações armazenadas em bases de dados e descobre regras de associação, indicando, por exemplo, que 98% dos clientes que compram pneus e acessórios para carros, também solicitam serviços de manutenção em seus carros. Nesse caso, pneus, acessórios e serviços de manutenção representam itens de dados constantes das transações na base de dados. Ou, considerando um cenário de um supermercado, através da aplicação do algoritmo descobriu-se que clientes que compram fraldas, também costumam comprar cerveja. Com base nessas informações, as empresas podem direcionar os seus programas e estratégias de *marketing*.

Além do conceito de regras de associação, o algoritmo Apriori também se baseia nos conceitos de suporte e confiança. Nesse contexto, o suporte (s) de uma regra de associação pode ser definido como o percentual de transações da base de transações que contém os itens de dados que participam da regra. Ou seja, representa a probabilidade de que aquela regra ocorra na base de transações. A confiança (c) de uma regra de associação, por outro lado, pode ser definida como o percentual de transações onde um item A aparece, que também contém um item B. Dessa forma, representa uma probabilidade condicional.

O algoritmo Apriori pode ser formalmente definido da seguinte forma. Assuma que $I=\{i_1, i_2,..., i_n\}$ seja um conjunto de itens de dados; $T=\{t_1, t_2, ..., t_n\}$ seja um conjunto de transações, onde cada transação t_k agrega um subconjunto de itens tal que $t_k \in T$, $1 \leq k \leq |T|$ e $t_k \subseteq I$. Assuma que uma transação t_k contém um subconjunto X dos itens em I (i.e., $X \subseteq I$). Assuma também que uma transação t_j contém um subconjunto Y dos itens em I (i.e., $Y \subseteq I$), e que k pode ser igual a j. Uma **regra de associação** é então uma implicação na seguinte forma: $X \Rightarrow Y$, se $X \cap Y = \varnothing$, c% das transações em T que contêm X também contém Y, e s% das transações em T contém $X \cup Y$. No exemplo $X \Rightarrow Y$, X é o **antecedente** da regra e Y o seu **conseqüente**. Uma regra de associação pode conter um conjunto de itens no seu antecedente e um conjunto de itens no seu conseqüente. É possível, entretanto, descobrir regras contendo apenas um item como antecedente e um item como conseqüente, i.e. regras binárias. O antecedente pode ainda ter o seu valor especificado. Associado a cada transação pode se encontrar um identificador único, denominado TID (*transacion* ID), que indica as transações que suportam uma regra de associação.

O algoritmo Apriori descobre todas as regras de associação em uma determinada base de transações, que tenham confiança e suporte maiores que a confiança mínima e o suporte mínimo informados pelo usuário, respectivamente. Apriori descobre regras de associação em duas fases, a saber: (1) encontra todos os conjuntos de itens (*itemsets*) freqüentes, i.e. aqueles conjuntos de itens que juntos têm suporte maior que o suporte mínimo informado pelo usuário; (2) utiliza os conjuntos de itens freqüentes encontrados para derivar as regras de associação.

Os algoritmos de mineração de dados examinam os itens de dados em grandes bases de transações, a fim de descobrir padrões recorrentes e generalizações sobre os itens de dados. Nesse contexto, relações interessantes podem ser descobertas entre conjuntos de itens que ocorrem juntos em certo número de transações (i.e. regras de associação (AGRAWAL e SRIKANT, 1994)); entre seqüências de itens de dados que ocorrem em diferentes transações, numa dada ordem e com uma certa freqüência (i.e. padrões seqüenciais – *sequential patterns* - (AGRAWAL e SRIKANT, 1995)); e entre o tempo de ocorrência das transações (i.e. *time-series clustering* (AGRAWAL *et al.*, 1995)).

De acordo com KLEMETTINEN *et al.* (1997), um processo de mineração envolve as seguintes fases: pré-processamento e transformação, que transforma a base

de dados bruta num formato que possa ser interpretado pelo algoritmo de mineração; descoberta de padrões ou mineração; e apresentação e interpretação dos resultados, que pode requerer grande esforço em função do número de padrões descoberto, os quais podem ser filtrados levando-se em conta a sua validade estatística ou o conhecimento do domínio. A mineração de dados é, em geral, um processo iterativo e requer um humano e ferramentas de apoio que permitam guiar o processo de descoberta de padrões.

Na literatura de ER pesquisada, foram encontrados alguns trabalhos que utilizam técnicas de mineração de dados, entretanto com uma aplicação diferente da proposta nesta tese (veja capítulo 4, seção 4.2). MONTES DE OCA e CARVER (1998) utilizam regras de associação e um modelo de representação visual para apresentar graficamente os subsistemas que são identificados a partir da representação do banco de dados do sistema. SARTIPI (2003) utiliza um sub-produto de regras de associação, i.e. conjuntos de itens de dados freqüentes com suas respectivas transações, a fim de descobrir entidades do sistema similares em função das suas propriedades estruturais.

Em (MOHAMMAD *et al.*, 2002), uma técnica de mineração de padrões seqüenciais é utilizada para capturar requisitos de sistemas legados como cenários de uso. Rastros de execução, nesse trabalho, são modelados como interações do usuário com a aplicação, onde um conjunto de interações define um cenário de uso da aplicação. O objetivo é a recuperação de modelos comportamentais. TJORTJIS *et al*. (2003) empregam regras de associação sobre o código fonte a fim de recuperar a estrutura do programa e apoiar a sua compreensão. Entidades do código, i.e. funções, procedimentos ou classes, são agrupados em conjunto se seus atributos, i.e. itens de dados, participam de regras de associação comuns. As entidades representam as transações e os itens de dados representam variáveis acessadas e declaradas, tipos de dados e chamadas a outras funções.

Em (ZAIDMAN et al., 2005), o algoritmo HITS de mineração de dados Web (KLEINBERG, 1999) é utilizado para minerar rastros de execução e descobrir classes-chave da aplicação. Classes-chave são classes controladoras que correspondem ao conceito de páginas *hub*[6] na Web, ou seja, aquelas que não contêm informação e dependem de muitas outras páginas, sendo as que primeiro devem ser examinadas para a compreensão do programa. Nesse trabalho, o objetivo é apoiar a compreensão de

[6] Páginas *hub* são definidas em (KLEINBERG, 1999) como sendo páginas que se referem a outras páginas que contêm informação, i.e. *authority pages*, ao invés de serem informativas por elas mesmas.

programa, indicando as classes que primeiro devem ser investigadas em uma tarefa de manutenção. Nesta tese, o objetivo da mineração não é priorizar determinadas classes durante a análise do software, mas detectar grupos de classes fortemente relacionados que devem compor elementos arquiteturais.

3.2.3 – Recuperação de Arquitetura

Diversos trabalhos de recuperação de arquitetura de software são propostos na literatura, cada um extraindo uma forma de elemento arquitetural, que representa, em geral, um componente, e um determinado conjunto de visões arquiteturais, sendo que a maioria deles recupera apenas uma visão estática da arquitetura (KAZMAN e CARRIÈRE, 1997; ANQUETIL e LETHBRIDGE, 1999; BOJIC e VELASEVIC, 2000; DING e MEDVIDOVIC, 2001; MENDONÇA e KRAMER, 2001; SARTIPI, 2003; MITCHELL e MANCORIDIS, 2006; SCHMERL *et al.*, 2006).

A recuperação de arquitetura é motivada pela necessidade de compreensão de um sistema existente, que não possui documentação atualizada, para fins de manutenção, reutilização, migração para abordagens de LP ou ED etc. Ela envolve um processo de 3 fases, a saber (DEURSEN *et al.*, 2004): *extração* – extração de informação "bruta" a partir do código ou executável; *abstração* – agrupamentos de elementos do código em elementos de mais alto nível de abstração; e *apresentação* ou *visualização* – envolvendo a apresentação dos modelos arquiteturais extraídos. Na fase de abstração, diferentes técnicas podem ser empregadas, como: agrupamento de elementos, detecção de padrões, análise de conceito formal (*formal concept analysis*), técnicas de mineração de dados etc.

Nesta tese, as abordagens de recuperação de arquitetura são classificadas de acordo com a principal fonte de informação utilizada na recuperação, da seguinte forma: recuperação baseada em estilos e padrões arquiteturais; recuperação baseada na detecção de padrões de domínio; recuperação baseada em requisitos funcionais; e recuperação baseada em outras fontes de informação.

Elas são avaliadas segundo princípios da área de Arquitetura de Software, como as perspectivas arquiteturais reconstruídas e o tipo de elemento arquitetural recuperado. É levada em conta a sua generalidade em relação à linguagem de programação e domínio de aplicação, uma vez que abordagens muito específicas têm seu escopo de aplicação limitado. O apoio ferramental e a possibilidade de integração a um ambiente de desenvolvimento de software (ADS) também são investigados, a fim de reduzir o

esforço da ER, além da realização de estudos experimentais planejados visando avaliar a viabilidade da abordagem. Nesse contexto, as contribuições e os pontos em aberto de cada grupo de abordagens são ressaltados ao longo da seção, sendo apresentado um quadro comparativo entre as abordagens e possibilidades de contribuições na área.

3.2.3.1 – Recuperação baseada em Estilos e Padrões Arquiteturais

Algumas abordagens visam o reconhecimento de componentes e conectores de estilos ou padrões arquiteturais no sistema (HARRIS *et al.*, 1995; HARRIS *et al.*, 1997a; MENDONÇA e KRAMER, 2001; GALL e PINZGER, 2002; SCHMERL *et al.*, 2006), recuperando uma representação da arquitetura baseada nesses estilos.

Em (HARRIS *et al.*, 1995; HARRIS *et al.*, 1997a), a abordagem implementada na ferramenta ManSART recupera componentes e conectores de estilos arquiteturais através da análise de padrões de programação (i.e. *clichés*) reconhecidos na árvore sintática abstrata (AST) extraída com um *parser* para a linguagem de programação alvo. Os reconhecedores de estilos de ManSART percorrem a AST e realizam três tarefas de reconhecimento: (1) identificam estruturas sintáticas que indicam a presença de um componente ou conector (ex: um comando "*system*" no Unix que indica a existência de um conector entre dois componentes executáveis); (2) delimitam os componentes detectados, mapeando-os para estruturas do código (ex: no caso do comando "*system*", essa tarefa envolve delimitar o programa que contém o comando, encontrando a sua raiz, i.e. uma declaração "*main*", na AST); (3) realizam uma análise de fluxo de controle e fluxo de dados, percorrendo os nós da AST, a fim de determinar os valores dos argumentos em uma chamada, identificando o destino de uma conexão. Os estilos detectados se limitam àqueles codificados através de reconhecedores e, em geral, àqueles que são passíveis de ser reconhecidos por padrões de programação.

A arquitetura recuperada é representada através de grafos que demonstram os diferentes estilos arquiteturais reconhecidos em diferentes visões, embora ManSART também permita a manipulação de visões. Através de operadores aplicados sobre grafos, é possível combinar visões seguindo diferentes estilos ou gerar novos estilos. Por exemplo, o estilo de tipos abstratos de dados ou OO pode ser derivado a partir do agrupamento de partes de um grafo conectadas, que representem procedimentos ou funções acessando dados persistentes.

Em (MENDONÇA, 1999; MENDONÇA e KRAMER, 2001), é apresentada uma abordagem de recuperação de arquitetura para sistemas distribuídos, denominada

X-Ray. Três técnicas de análise estática são empregadas: (1) classificação de módulos de componentes (*component module classification*), que é utilizada para identificar que módulos do código fonte constituem exclusivamente ou são compartilhados por cada processo; (2) busca de padrões sintática (*syntatic pattern matching*), utilizada para a identificação de padrões de programação que implementam conectores entre processos, semelhante à detecção de padrões descrita em ManSART (HARRIS *et al.*, 1997a); e (3) análise de alcance estrutural (*structural reachability analysis*), utilizada para associar interconexões identificadas pela técnica de busca de padrões a componentes individuais.

A classificação de módulos analisa os relacionamentos entre nós de um grafo, identificando o módulo que inicia um processo e os módulos exclusivamente acessados ou compartilhados por ele, através da análise de dominância e de alcance nos relacionamentos no grafo Segundo os autores, no contexto de aplicações distribuídas, 3 visões devem ser recuperadas, a saber: visão de execução, que identifica os processos e suas possíveis interações; visão física, envolvendo a possível determinação de alocação dos processos na rede, inferida através dos seus tipos; e visão de desenvolvimento, que relaciona os processos e suas interações a artefatos do código fonte.

Em (GALL e PINZGER, 2002), o reconhecimento de padrões arquiteturais também se baseia na detecção de padrões de codificação. Através da ferramenta ESPaRT (KNOR *et al.*, 1998), os autores reconhecem blocos de texto do programa que contêm os elementos-chave (i.e. *hot-spots*) de um padrão, ou seja, aqueles que sempre devem aparecer no código quando o padrão é utilizado, independente das suas variações de implementação. Como exemplo, pode-se citar o uso de estruturas de soquete (*socket*) em C e Java, podendo indicar a implementação de uma arquitetura Cliente-Servidor. Independente das suas variações de implementação, a criação de um objeto do tipo *Socket* sempre indica, nas duas linguagens, que uma arquitetura Cliente-Servidor está sendo empregada. A busca por padrões é um processo iterativo, onde a definição de um padrão vai sendo refinada a cada *hot-spot* detectado. Por exemplo, partindo do *hot-spot "mySocket = socket(...)"* em C*,* a recuperação da arquitetura Cliente-Servidor é refinada através do reconhecimento de uma construção "*connect*(*asocket, address, ...*)" para a identificação do Cliente, e de uma estrutura "*bind(asocket, address,...) listen(asocket,....) accept(asocket, ...)*" para a identificação do Servidor.

Em (SCHMERL *et al.*, 2006), é apresentada uma abordagem de recuperação de arquitetura com base na análise dinâmica, denominada DiscoTect. Ela identifica estilos

arquiteturais da aplicação a partir da análise de eventos[7] disparados em tempo de execução. Conjuntos de eventos revelam protocolos de comunicação entre componentes ou estilos de implementação, que são mapeados, através de regras de mapeamento, para a arquitetura. Por exemplo, o evento de criação de um servidor em Java, "new ServerSocket()", indica a criação de um componente Servidor na arquitetura, e o evento de conexão de um cliente, "aServerSocket.accept()", leva à criação de um componente cliente e de uma conexão. Eventos de acesso a arquivos compartilhados podem indicar um estilo arquitetural baseado em repositório, bem como eventos que demonstram a comunicação via *stream* de dados podem indicar um estilo de dutos e filtros.

DiscoTect recupera uma visão estática da arquitetura. Embora os autores argumentem que existam regularidades na implementação dos sistemas, as regras para a interpretação dos eventos, de uma forma geral, devem ser adaptadas e/ou estendidas para cada aplicação. Essa adaptação requer conhecimento humano acerca do sistema sob análise e da linguagem utilizada em DiscoTect para a definição das regras, o que representa uma restrição da abordagem.

Conforme observado na análise das abordagens nesta seção, embora a identificação de estilos e padrões arquiteturais represente um possível caminho para a reconstrução da arquitetura, essa identificação é dificultada pela distância de vocabulário existente entre o domínio arquitetural e o domínio das linguagens de programação e pelas variações de implementação dos componentes e conectores em diferentes ambientes de implementação.

De acordo com HARRIS e REUBENSTEIN (1997), 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, como padrão de nomenclatura para os elementos de uma mesma camada e protocolos de comunicação. Além disso, diferentes linguagens de programação conduzem a diferentes formas de implementação dos estilos arquiteturais. Essa distância de vocabulário caracteriza um

[7] O termo evento é utilizado indiscriminadamente nesse trabalho de (SCHMERL *et al.*, 2006), referindo-se tanto a chamadas de métodos, como a instanciação de objetos ou disparo de eventos com invocação implícita, propriamente.

problema já conhecido na comunidade de ER, i.e. o "Problema da Associação de Conceitos" (*Concept Assignment Problem*) (BIGGERSTAFF *et al.*, 1994), que ocorre quando a origem real do conceito implementado em algoritmo não é possível de ser atingida. Cada componente ou conector arquitetural 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. Isso dificulta a recuperação do conceito arquitetural original a partir do programa.

A fim de superar essa distância entre o vocabulário arquitetural e o vocabulário de implementação, alguns trabalhos como a ArchJava (Connecting software architecture to implementation), de ALDRICH *et al.* (2002), embutem construtores de estilos arquiteturais na linguagem de programação. Entretanto, acabam gerando soluções proprietárias.

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 ou *clichés* 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. Entretanto, essas abordagens acabam se tornando dependentes de linguagens de programação, devendo ser estendidas e adaptadas a cada nova plataforma.

3.2.3.2 – Recuperação baseada na Detecção de Padrões de Domínio

Algumas abordagens utilizam o conhecimento do domínio da aplicação como fonte de informação prioritária para a reconstrução da arquitetura (KAZMAN e CARRIÈRE, 1997; RICHNER e DUCASSE, 1999; RIVA e RODRIGUEZ, 2002; O'BRIEN e STOERMER, 2003; SARTIPI, 2003; SARTIPI e KONTOGIANNIS, 2003).

Em (KAZMAN e CARRIÈRE, 1997), é apresentado um ambiente para a recuperação de arquitetura que integra diversas ferramentas para apoiar o processo, i.e. o Dali *workbench*. A idéia é que diversas ferramentas de análise estática e dinâmica, como *parsers*, analisadores léxicos, *profilers* etc. extraiam diferentes visões de baixo nível do sistema, as quais são combinadas e armazenadas em um repositório baseado em SQL (i.e. um banco de dados relacional). Com base nas informações do repositório, um grafo é construído e visualizado em um visualizador de grafos integrado ao ambiente, onde o modelo em baixo nível de abstração é manipulado diretamente ou indiretamente pelo usuário até se chegar ao nível de abstração arquitetural. As

manipulações indiretas se dão através de um mecanismo de aplicação de consultas (*queries*) seguida do agrupamento dos elementos retornados, permitindo o mapeamento de padrões de domínio para a arquitetura. A fim de especificar essas consultas, o usuário deve buscar conhecimento sobre o domínio em documentos disponíveis ou com especialistas. O ambiente Dali passou por evoluções, sendo conhecido atualmente como ARMIN (O'BRIEN e STOERMER, 2003), com maior capacidade de manipulação das visões arquiteturais.

RICHNER e DUCASSE (1999) apresentam um ambiente para a geração de visões arquiteturais customizáveis para sistemas OO, a partir de fatos extraídos do sistema com a análise estática e dinâmica. A recuperação da arquitetura é baseada em regras especificadas pelo usuário sobre os fatos representados em Prolog, que permitem a obtenção de abstrações de alto nível do sistema. Regras em Prolog são criadas com base em conhecimento do domínio. Visões podem projetar um determinado tipo de relacionamento extraído da aplicação, como, por exemplo, invocação de métodos entre classes, e apresentar os componentes em um maior ou menor nível de detalhamento.

Em (RIVA e RODRIGUEZ, 2002), os modelos recuperados do código, através da análise estática e dinâmica, são armazenados em uma base de fatos em Prolog e os agrupamentos de elementos são realizados através da definição de regras sobre os fatos. Essas regras de agrupamento são definidas para cada aplicação através do conhecimento de especialistas do domínio e podem ser reaproveitadas entre diferentes versões de uma mesma aplicação. Entretanto, diferente de RICHNER e DUCASSE (1999), o objetivo não é a geração de visões customizáveis, mas a separação de aspectos estáticos e dinâmicos em diferentes visões arquiteturais. O modelo estático é representado através de grafos hierárquicos na ferramenta Rigi (WONG *et al.*, 2004) e o dinâmico através de diagramas de seqüência de mensagens MSC (*Message Sequence Charts*) também na ferramenta Rigi, estendida para esse tipo de visualização.

A fim de lidar com o volume de mensagens da análise dinâmica, mecanismos de ocultação de informação são utilizados, através da abstração horizontal e vertical aplicadas nos diagramas MSC. A abstração horizontal leva ao agrupamento de entidades que pertencem a um mesmo elemento arquitetural, encapsulando as mensagens trocadas entre elas. A abstração vertical, por outro lado, leva ao agrupamento de mensagens contíguas em uma mensagem de mais alto nível. Os diagramas podem ainda ser recortados por cenário de caso de uso.

A abordagem apresentada em (SARTIPI, 2003; SARTIPI e KONTOGIANNIS, 2003), implementada no ambiente Alborz, propõe uma solução para a recuperação da arquitetura baseada em combinação de grafos (*graph matching*) e mineração de dados. Através de um ambiente de busca iterativa e incremental, o usuário elabora uma arquitetura conceitual de alto nível em uma linguagem arquitetural de consulta (AQL – *Architecture Query Language*), que permite estabelecer restrições a respeito da estrutura dos módulos e suas interconexões na arquitetura final (ex: número de funções em um módulo, função que deve necessariamente constar de um módulo), deixando pontos em aberto (*placeholders*). Para elaborar a arquitetura conceitual, o usuário deve buscar conhecimento do domínio em documentação disponível, especialistas etc. A arquitetura conceitual é traduzida para um grafo, que é comparado com o grafo extraído do código fonte em um mecanismo denominado *graph matching*, levando ao preenchimento dos *placeholders* para a geração da arquitetura final.

A fim de reduzir o espaço de busca no processo de *graph matching*, um algoritmo de mineração de dados baseado em regras de associação (AGRAWAL e SRIKANT, 1994) é empregado, decompondo o grafo extraído do código em regiões contendo entidades "similares" (ex: variáveis, tipos de dados, funções) em função do seu grau de associação estrutural (ex: variáveis e tipos de dados sendo acessados pelas mesmas funções, funções sendo chamadas pelo mesmo grupo de funções). Essas entidades similares são candidatas a permanecer juntas na arquitetura durante o preenchimento dos *placeholders*.

As abordagens apresentadas nesta seção dependem de conhecimento estrutural acerca do domínio da aplicação para gerar resultados eficazes. Embora as técnicas utilizadas sejam independentes de domínio, a sua aplicação depende de um modelo de alto nível da aplicação ou do conhecimento de padrões arquiteturais ou estruturas do domínio para se tornarem efetivas. Os critérios para a reconstrução dos elementos arquiteturais são dependentes de domínio, requerendo a sua redefinição a cada novo sistema ou domínio.

3.2.3.3 – Recuperação baseada em Requisitos Funcionais

Abordagens direcionadas por requisitos funcionais são aquelas que guiam a recuperação dos elementos arquiteturais pelos requisitos funcionais da aplicação (BOJIC e VELASEVIC, 2000; DING e MEDVIDOVIC, 2001). Normalmente, são mais

fortemente baseadas na análise dinâmica da aplicação através do monitoramento dos cenários de uso relacionados aos requisitos funcionais.

BOJIC e VELASEVIC (2000) partem da definição de casos de uso, e casos de teste que cubram os casos de uso definidos, para a análise dinâmica da aplicação e coleta de rastros de execução. Os rastros são analisados através da técnica de análise de conceito formal (*Formal Concept Analysis*), que provê um mecanismo para a detecção de conceitos através de grupos de objetos que compartilham grupos de valores de atributos (BURMEISTER, 1998). Os objetos considerados são as funções que implementam casos de uso, sendo os casos de uso os atributos desses objetos. Dessa forma, o objetivo é descobrir o maior conjunto possível de funções que implementam juntas um grupo de casos de uso em comum. Funções e casos de uso são dispostos em uma matriz, a partir da qual é construído um grafo hierárquico de conceitos (i.e. *concept lattice*). A partir desse grafo, subsistemas hierárquicos são derivados na arquitetura, cada subsistema correspondendo a um conceito do grafo. Porém, a análise de conceitos formal gera uma rede aninhada de conceitos, onde o mesmo objeto participa de vários conceitos, sendo difícil determinar a sua localização na arquitetura.

Os autores propõem o agrupamento de funções não-membro, i.e. de C++, em subsistemas de acordo com os casos de uso que implementam, sendo considerado o conceito de mais baixo nível na hierarquia que as contêm, por serem esses os que agrupam o conjunto dos seus casos de uso. Classes, por outro lado, são agrupadas no subsistema que representa o conceito que contém a maior parte das suas funções, gerando uma visão funcional da arquitetura. Entretanto, não foram conduzidos estudos experimentais para avaliar esses critérios de agrupamento.

Em (DING e MEDVIDOVIC, 2001), é apresentada a abordagem Focus, direcionada por requisitos de manutenção adaptativa e evolutiva. Os casos de uso impactados por cada ciclo de manutenção são exercitados e rastros de execução coletados. Porém, antes do início dos ciclos de manutenção, uma arquitetura de alto nível para a aplicação é elaborada segundo um estilo arquitetural e um modelo estático é extraído do código através de um *parser*, onde classes são agrupadas em elementos arquiteturais com base em seus acoplamentos estáticos. Com base nos rastros de execução coletados para o software, elementos arquiteturais impactados pelos requisitos de manutenção são alocados na arquitetura conceitual. Essa alocação se baseia principalmente na análise das comunicações entre esses elementos.

As abordagens apresentadas nesta seção são direcionadas por requisitos funcionais, onde a geração dos elementos arquiteturais ou das suas conexões se baseia na análise dinâmica do software, exercitando um conjunto de cenários de uso. Essas abordagens apresentam uma maior independência de linguagem de programação do que as abordagens baseadas no reconhecimento de estilos e padrões arquiteturais no código. Apresentam também uma maior independência de domínio do que as abordagens baseadas na detecção de padrões do domínio, apresentando critérios para a reconstrução dos elementos arquiteturais que são independentes de domínio. Entretanto, elas exigem algum conhecimento do comportamento do sistema alvo para permitir a sua execução, uma vez que são mais fortemente apoiadas na análise dinâmica. Além disso, conforme apresentado na seção 3.2.1, técnicas de redução do volume de informação são necessárias na análise dinâmica e essas não são propostas nas abordagens avaliadas.

3.2.3.4 – Recuperação baseada em Outras Fontes de Informação

Em (ANQUETIL e LETHBRIDGE, 1999), a recuperação da arquitetura é baseada na análise dos nomes dos arquivos do código fonte, que tendem a ser formados por abreviações ou *substrings* que encapsulam conceitos do domínio. Arquivos são agrupados em subsistemas em função da similaridade das partes de seus nomes. Partes de nomes de arquivos podem expressar, além de conceitos do domínio, conceitos referentes a tipos de dados manipulados (ex: bd - banco de dados, *flag*), estruturas de controle (ex: ctr - controlador), período de tempo em que o processamento ocorre (ex: init) etc. De acordo com o tipo de informação contida nos nomes dos arquivos, diferentes tipos de elementos arquiteturais podem ser recuperados.

O desafio da abordagem se encontra em sistemas legados executados sobre sistemas operacionais antigos que forçam a atribuição de nomes com tamanho limitado (em geral, de 8 a 12 caracteres), que acabam sendo extremamente abreviados, não possuindo delimitadores entre as suas partes, como sinal de sublinhado (*underscore*) ou letra maiúscula. Dessa forma, abreviações extraídas podem ser válidas ou não. Arquivos devem ser agrupados com base apenas na similaridade das abreviações válidas. Assim, as seguintes soluções são propostas para o filtro de abreviações contidas nos nomes desses arquivos, como: comparação de abreviações candidatas com abreviações em identificadores de rotinas no código ou palavras que podem ser encontradas no dicionário; utilização de uma abordagem estatística para considerar apenas as abreviações que apareçam em vários arquivos; filtro de abreviações que não apareçam

em comentários no código etc. Os autores conduziram 2 estudos experimentais que verificaram a viabilidade da abordagem.

Em (MANCORIDIS *et al.*, 1999; MITCHELL e MANCORIDIS, 2006), é apresentado um mecanismo de *clustering* hierárquico, que se baseia em princípios de projeto, como baixo acoplamento e alta coesão. *Clustering* representa o processo de classificação de entidades (ex: classes) em subconjuntos que têm significado no contexto de um problema particular (JAIN, 1988). O algoritmo implementado na ferramenta Bunch utiliza como entrada um grafo de entidades e relacionamentos extraído do código fonte e gera como saída uma partição desse grafo em grupos que representam elementos arquiteturais.

A fim de gerar uma partição do grafo de boa qualidade, a ferramenta Bunch tenta maximizar o valor de uma função objetiva, chamada qualidade da modularização (*Modularization Quality* – MQ). MQ determina a qualidade de uma partição do grafo como a solução de compromisso (i.e. *trade-off*) entre inter-conectividade (i.e. acoplamentos entre entidades de diferentes grupos ou elementos arquiteturais) e intra-conectividade (i.e. acoplamentos entre entidades do mesmo grupo ou elemento arquitetural). O objetivo é atingir baixos valores de inter-conectividade contra altos valores de intra-conectividade, buscando-se um "meio termo" ou solução de compromisso entre esses valores. Essa solução de compromisso é baseada na premissa de que sistemas de software bem projetados são organizados em elementos arquiteturais coesos e fracamente acoplados. Bunch suporta diferentes algoritmos, como *Hill-Climbing* (MITCHELL, 2002) e algoritmos genéticos (DOVAL *et al.*, 1999), dos próprios autores, os quais aplicam a função objetiva mencionada. Entretanto, pode ser estendida para novos algoritmos e funções objetivas. Os autores argumentam que o algoritmo de *Hill-Climbing* foi o que produziu os melhores resultados para os sistemas avaliados até o momento.

Como Bunch implementa um algoritmo de *clustering* hierárquico, o usuário pode selecionar o nível da hierarquia onde deseja fazer o "recorte", i.e. de onde deseja extrair os elementos arquiteturais. Quanto mais alto o recorte na hierarquia de subsistemas ou elementos arquiteturais, maior granularidade terão os grupos e menor será o número de grupos retornados. Quanto mais baixo na hierarquia o recorte, menor granularidade terão os grupos e maior será o número de grupos retornado. O ponto onde fazer o recorte para se obter uma arquitetura adequada representa um desafio para algoritmos hierárquicos. Finalmente, Bunch apresenta uma funcionalidade de "adoção

de órfãos" (*orphan adoption technique*) para fins de apoio à manutenção. Uma arquitetura já recuperada não precisa ser novamente analisada em Bunch, mas apenas as novas entidades criadas ou estruturalmente modificadas são alocadas na arquitetura, objetivando-se o melhor valor de MQ. Vários estudos experimentais para avaliar a viabilidade da abordagem implementada em Bunch já foram conduzidos e relatados na literatura (ANQUETIL *et al.*, 1999; MITCHELL e MANCORIDIS, 2006).

Nesta seção, foram descritas abordagens baseadas em fontes de informação mais genéricas, i.e. identificadores de arquivos e princípios de projeto, ao invés de conhecimentos mais específicos, como padrões arquiteturais, padrões de domínio ou requisitos funcionais. Observa-se que elas atingem um grau de generalidade maior do que as outras abordagens. As abordagens baseadas em estilos e padrões arquiteturais costumam se basear na detecção de padrões de codificação para o reconhecimento das abstrações arquiteturais. As abordagens baseadas em padrões ou requisitos do domínio requerem algum conhecimento do domínio para apoiar a recuperação. As abordagens apresentadas nesta seção requerem informações genéricas em relação a domínio ou linguagem de programação como entrada para a recuperação de arquitetura. Informações essas inerentes a todo sistema de software, como identificadores de arquivos e dependências entre arquivos.

3.2.4 – Considerações sobre a Engenharia Reversa

Conforme observado, a maioria das abordagens de recuperação de arquitetura é semi-automatizada, requerendo conhecimento humano. A exceção se encontra nas abordagens baseadas em *clustering*, como a implementada na ferramenta Bunch. PASHOV e RIEBISCH (2003) afirmam que, embora a maioria das abordagens se apóie no código como a fonte de informação mais disponível e confiável dos sistemas existentes, algum conhecimento sobre o domínio do problema é requerido para facilitar a extração de informações arquiteturais úteis sobre o software. A Figura 3.1 apresenta fontes de informação úteis para a recuperação de arquitetura.

De acordo com GALL e PINZGER (2002), a maior parte das abordagens de recuperação de arquitetura combina técnicas de análise do software de baixo para cima com técnicas de cima para baixo. As técnicas de baixo para cima extraem conhecimento de artefatos disponíveis, como o código fonte e executável, enquanto as técnicas de cima para baixo incorporam conhecimento humano ao processo, útil para se chegar a modelos em nível de abstração arquitetural, conforme mostra a Figura 3.2. De acordo

com a Figura 3.2, um desenvolvedor guiando o processo de recuperação de arquitetura vai embutir conhecimento humano ao processo, podendo apoiar a manipulação dos modelos de mais baixo nível até a geração dos elementos arquiteturais.

A Tabela 3.2 apresenta um resumo das abordagens de recuperação de arquitetura, segundo um conjunto de critérios estabelecidos para a sua análise, conforme apresentado no início desta seção. Como mostra a Tabela 3.2, a maior parte das abordagens pesquisadas extrai apenas uma visão estática da arquitetura, com exceção de (RICHNER e DUCASSE, 1999; DING e MEDVIDOVIC, 2001; RIVA e RODRIGUEZ, 2002), o que de fato se observa como tendência nas abordagens de recuperação de arquitetura da literatura. Além disso, a maioria possui ferramental de apoio, mas que não é integrado a um ADS, onde visões complementares de um sistema poderiam ser relacionadas, através da rastreabilidade entre elementos de modelo, e navegadas, facilitando a sua compreensão.

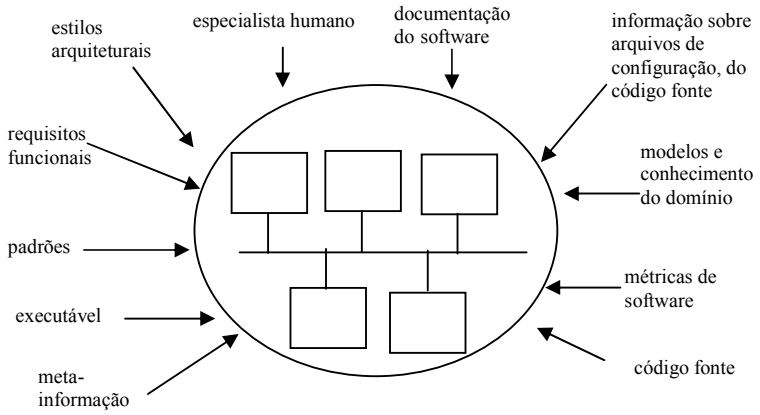

Figura 3.1: Fontes de informação para a recuperaração de arquitetura. Adaptada de GALL *et al.* (1996).

Em relação à generalidade das abordagens, ela foi avaliada levando-se em conta, mais fortemente, a fase de abstração de um processo de recuperação de arquitetura, ou seja, os critérios para a transformação de modelos "brutos", extraídos com a análise estática e dinâmica, em modelos arquiteturais. Nesse quesito, somente as abordagens apresentadas em (ANQUETIL e LETHBRIDGE, 1999; DING e MEDVIDOVIC, 2001; MITCHELL e MANCORIDIS, 2006) são de fato genéricas. Algumas abordagens têm a sua viabilidade avaliada através de estudos experimentais planejados, outras não. Destaca-se a importância desse tipo de estudo para oferecer indícios da viabilidade de uma abordagem, sem os quais fica difícil aferir a eficácia de seus resultados.

Percebe-se, então, que muitas oportunidades de pesquisa ainda se apresentam na área de recuperação de arquiteturas. Como possibilidades de contribuição, destacam-se o benefício de abordagens genéricas, que recuperem visões arquiteturais estáticas e dinâmicas do software, e que estejam integradas a um ADS, permitindo a manipulação e compreensão dessas visões recuperadas. A abordagem proposta nesta tese visa tratar essas questões de pesquisa.

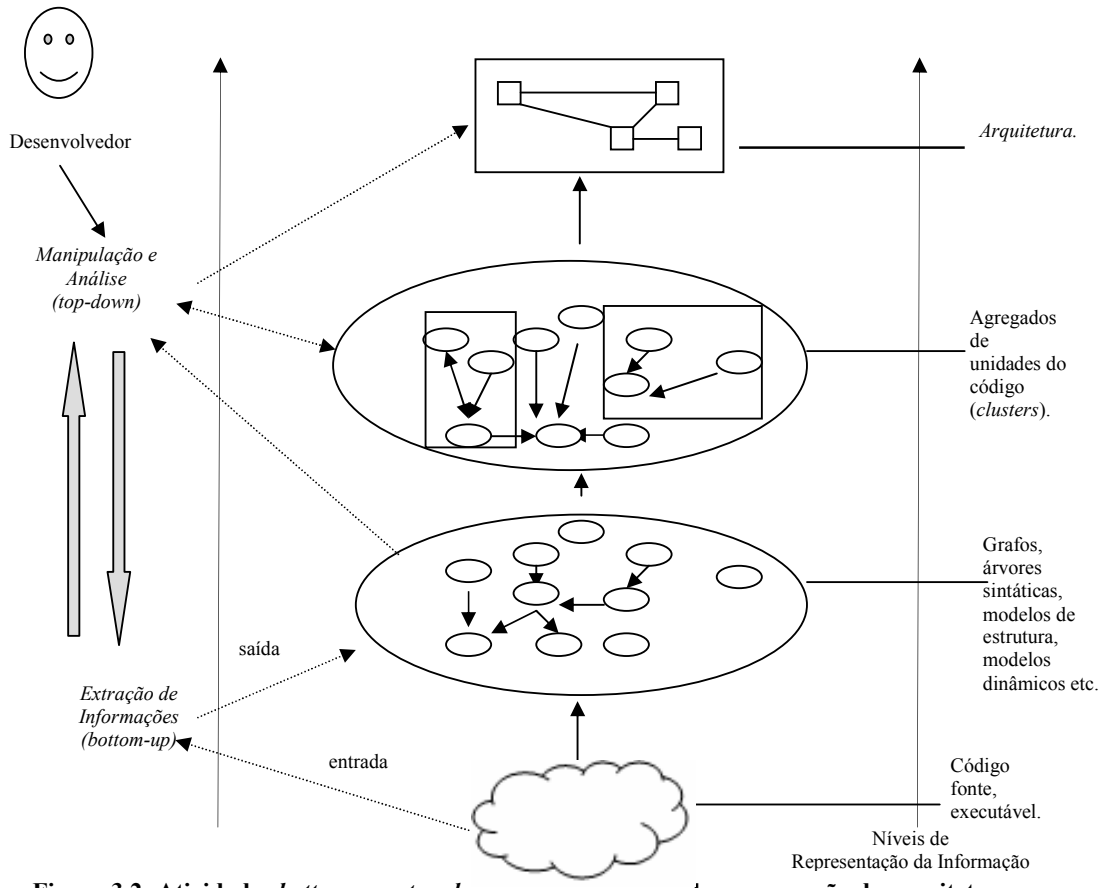

Figura 3.2: Atividades *bottom-up* e *top-down* em um processo de recuperação de arquitetura.

Entretanto, convém ressaltar que nenhuma abordagem de recuperação de arquitetura é adequada a todos os sistemas legados, estruturados segundo diferentes padrões arquiteturais ou de domínio, e nem a todos os objetivos pretendidos com a recuperação, como reutilização, reengenharia, compreensão de programa etc. Observa-se, por exemplo, que em relação ao tipo do elemento arquitetural recuperado, algumas abordagens recuperam abstrações que mapeiam componentes e conectores de estilos arquiteturais, outras recuperam abstrações que se referem a conceitos ou padrões do domínio, ou ainda a módulos de controle. De fato, cada abordagem de recuperação de arquitetura visa a recuperação da visão arquitetural que mais atende aos objetivos pretendidos com a recuperação. Além disso, não existe um único modelo arquitetural

válido para um sistema de software. Um sistema organizado em camadas, por exemplo, pode ter um modelo arquitetural válido representado em camadas e um outro modelo arquitetural válido também, representado através de módulos que reflitam conceitos do domínio, perpassando as camadas.

Tabela 3.2: Quadro comparativo das abordagens de recuperação de arquitetura.

Abordagem de Recuperação de Arquitetura	Perspectivas Arquiteturais	Tipo do Elemento Arquitetural	Generalidade	Ferramental	Integração ADS	Estudos Experimentais
ManSART (HARRIS *et al.*, 1995)	Visão estática baseada em estilos arquiteturais.	Componentes de Estilos Arquiteturais	Não. Reconhece *clichés* em C/Unix.	Sim.	Não.	Não. Estudos de caso informais.
X-Ray (MENDONÇA e KRAMER, 2001)	Estática.	Processo.	Não. Se baseia em padrões de programação.	Sim.	Não.	Sim.
Pattern-supported (GALL e PINZGER, 2002)	Estáticas baseadas em padrões arquiteturais.	Componentes de Padrões Arquiteturais.	Não. Se baseia no reconhecimento de padrões de programação.	Sim.	Não.	Não. Estudos de caso informais.
DiscoTect (SCHMERL *et al.*, 2006)	Visão estática.	Componentes e conectores de Estilos Arquiteturais	Parcial. Esforço de adaptação das regras de mapeamento	Sim.	Não.	Sim.
Dali (KAZMAN e CARRIÈRE, 1997)	Visão estática.	Elementos arquiteturais que representam conceitos do domínio.	Parcial. Requer conhecimento sobre padrões estruturais do domínio.	Sim.	Não.	Não. Estudos de caso informais.
High-Level Views for OO Applications (RICHNER e DUCASSE, 1999)	Visão estática e dinâmica, com base em regras definidas pelo usuário.	Elementos arquiteturais representam conceitos do domínio.	Parcial. Regras específicas para cada aplicação.	Sim.	Não.	Não.
Sincronização entre Visões Estática e Dinâmica (RIVA e RODRIGUEZ, 2002)	Visão estática e dinâmica (comportamental).	Elementos representam conceitos do domínio.	Parcial. Regras para agrupamento de elementos dependentes de aplicação.	Sim.	Não.	Sim.
Alborz (SARTIPI e KONTOGIANNIS, 2003)	Visão estática.	Elementos que representam conceitos do domínio.	Parcial. Requer modelo arquitetural de alto nível específico de domínio.	Sim.	Não.	Sim.

Tabela 3.2: Quadro comparativo das abordagens de recuperação de arquitetura (continuação).

Abordagem de Recuperação de Arquitetura	Perspectivas Arquiteturais	Tipo do Elemento Arquitetural	Generalidade	Ferramental	Integração ADS	Estudos Experimentais
Abordagem direcionado por Casos de Uso (BOJIC e VELASEVIC, 2000)	Visão estática.	Elementos representam conceitos do domínio.	Parcial. Ferramental e alguns critérios específicos para C++.	Sim.	Sim. Exportação para *Ratio-nal Rose*.	Não.
Focus (DING e MEDVIDOVIC, 2001)	Visão estática e dinâmica através de diagramas de seqüência.	Elementos de estilos arquiteturais combinados a conceitos do domínio.	Sim.	Não. Ferramentas existentes e atividades manuais	Não.	Sim.
File Names (ANQUETIL e LETHBRIDGE, 1999)	Visão estática.	Elementos podem representar conceitos do domínio, controladores etc.Depende do conteúdo de nomes de arquivos.	Sim.	Sim.	Não.	Sim.
Bunch (MITCHELL e MANCORIDIS, 2006)	Visão estática.	Conceitos do domínio, componentes de estilos arquiteturais etc. Depende do acoplamento entre entidades do código.	Sim.	Sim.	Possibilidade através da API da Bunch.	Sim.

Nesse contexto, o cenário ideal para a recuperação de arquitetura é a integração de diferentes abordagens em um mesmo ADS. ANQUETIL e LETHBRIDGE (1999) destacam que uma ferramenta CASE (*Computer-Aided Software Engineering*) poderia oferecer ao usuário diferentes decomposições de um sistema de software para que ele selecionasse a mais compreensível.

Além da recuperação de arquitetura, a ER pode envolver outros objetivos, como, por exemplo, a localização de funcionalidades no código, que pode representar um primeiro passo na recuperação de arquitetura, havendo diversos trabalhos na literatura dedicados exclusivamente a esse tema (CHEN e RAJLICH, 2000; EISENBARTH *et al.*,

2003; ANTONIOL e GUEHENEUC, 2005; EISENBERG e VOLDER, 2005). A diferença entre localização de funcionalidades no código e recuperação de arquitetura, embora ambas possam se basear na análise dinâmica, é que na recuperação de arquitetura a cobertura de cenários monitorados precisa ser maior do que em trabalhos que visam localizar funcionalidades específicas no código.

3.3 – Comparação de Modelos de Software

Na área de Gerência de Configuração, é comum encontrar ferramentas de versionamento que calculam diferenças entre arquivos texto, representando código ou documentação de software, como CVS (CEDERQVIST, 2003), Subversion (COLLINS-SUSSMAN *et al.*, 2004) e RCS (TICHY, 1985). Esse tipo de abordagem apresenta limitações quando há a necessidade de comparar modelos de análise ou projeto, uma vez que não levam em conta a sintaxe, estrutura ou semântica dos elementos envolvidos, realizando uma comparação linha a linha dos documentos. Nesta tese, o foco é no cálculo de diferenças entre modelos arquiteturais, representados através de arquivos XMI (XML *Metadata Interchange*) (OMG, 2002b), para a detecção de variabilidades e opcionalidades em uma arquitetura de referência de domínio. Convém ressaltar que algoritmos que calculam diferenças entre arquivos, de qualquer tipo, são genericamente chamados de algoritmos de **diff**.

Motivações para o cálculo de diferenças entre arquiteturas incluem: reconciliar diferentes versões de uma arquitetura que passa por manutenções ao longo do tempo (ex: (MEHRA *et al.*, 2005)); comparar versões de modelos modificados por diferentes desenvolvedores (ex: (OLIVEIRA *et al.*, 2005a)); reconciliar diferentes arquiteturas representando duas variantes de uma LP, a fim de apoiar a sua evolução (ex: (CHEN *et al.*, 2003)) etc. A maior parte das abordagens existentes para calcular diferenças e juntar (*merge*) visões arquiteturais é baseada em suposições restritivas, tais como que os elementos das visões nas diferentes arquiteturas tenham identificadores únicos (ALANEN e PORRES, 2003; OHST *et al.*, 2003; MEHRA *et al.*, 2005; OLIVEIRA *et al.*, 2005a) ou tipos e nomes exatamente iguais (CHEN *et al.*, 2003). Dentre as abordagens pesquisadas, exceções foram encontradas na abordagem de ABI-ANTOUN *et al.* (2006), que calcula diferenças estruturais entre modelos, relaxando essas restrições.

Em (OLIVEIRA *et al.*, 2005a), é apresentado um mecanismo de controle de versões que atua sobre modelos descritos de acordo com o metamodelo da UML (OMG,

2001), através de arquivos XMI. Nesse trabalho, é ressaltada a distinção entre elementos sintáticos e elementos semânticos de modelos criados em ferramentas de modelagem, como ferramentas CASE (*Computer-Aided Software Engineering*), onde elementos semânticos são aqueles que representam conceitos e contêm os dados reais da entidade sendo modelada, enquanto os elementos sintáticos são as representações dos elementos semânticos em diagrama, apresentando características como cor, posição e tamanho. A abordagem permite a comparação de modelos através dos seus elementos semânticos, onde qualquer subtipo de elemento de modelo (i.e. *ModelElement* no metamodelo da UML) pode ser utilizado como unidade de comparação entre duas versões de um modelo. Exemplos de elementos de modelo incluem: classes, atributos, casos de uso, atores, associações, herança etc. A comparação de modelos XMI é feita levando-se em conta o identificador MOF (*Meta Object Facility*) (OMG, 2002a) (MOF ID) dos elementos nas diferentes arquiteturas. Por trabalhar com XMI, a abordagem atinge certa independência de ambiente de desenvolvimento.

MEHRA *et al*. (2005) apresentam uma abordagem de versionamento, **diff** e *merge* em diagramas, implementada em uma ferramenta CASE proprietária, i.e. Pounamu. As diferenças entre 2 versões de um mesmo diagrama são apresentadas graficamente e os usuários podem selecionar as diferenças que devem ser aceitas ou rejeitadas. São detectadas diferenças semânticas e sintáticas entre os modelos através da comparação dos identificadores dos elementos nos diagramas, que são os mesmos em diferentes versões do modelo. Entretanto, diferente da abordagem de (OLIVEIRA *et al.*, 2005a), não há flexibilidade em relação ao ambiente de desenvolvimento e nem em relação à unidade de comparação de modelos.

Em (ABI-ANTOUN *et al.*, 2006), uma abordagem que detecta diferenças baseada em informações estruturais é apresentada, utilizando algoritmos de comparação de árvores que identificam correspondências (*matches*) e resolvem diferenças estruturais. O algoritmo calcula o custo de operações de inserção, remoção e movimentações entre nós de sub-árvores, a fim de que a partir de uma árvore inicial seja possível se chegar a uma árvore final (i.e. *merge*). A abordagem compara nós em 2 árvores levando em conta seus nomes, embora identifiquem casos em que nós são renomeados, calculando o grau de similaridade entre eles através do tamanho de seqüências de *substrings* comuns identificadas. A abordagem permite que o usuário force ou proíba correspondências entre nós, embora a relação de ancestrais deva ser

preservada. Entretanto, a abordagem não compara elementos que, embora semanticamente equivalentes, possam possuir nomes totalmente diferentes.

Em (CHEN *et al.*, 2003), são apresentados 2 algoritmos para apoiar a evolução de LPs: (1) o primeiro algoritmo realiza **diff** entre 2 diferentes produtos (ou variantes) de uma LP que evoluem separadamente; (2) e o segundo algoritmo realiza a junção das diferenças com a arquitetura original da LP. A comparação é semântica e realizada a nível arquitetural, onde os elementos das diferentes arquiteturas são comparados através do seu nome, tipo e estrutura interna, uma vez que se assume que nas variantes de uma LP os elementos arquiteturais correspondentes não possuem necessariamente os mesmos identificadores. As diferenças detectadas atingem uma granularidade fina, uma vez que levam em conta diferenças entre ligações de componentes e conectores, estrutura interna dos elementos, assinaturas de operações nas interfaces dos componentes etc. As diferenças são mostradas a nível arquitetural através de arquiteturas representadas em xADL (DASHOFY *et al.*, 2002), uma ADL desenvolvida pelo próprio grupo de pesquisa dos autores.

3.4 – Considerações Finais

Neste capítulo, foram apresentadas técnicas e abordagens de ER, além de técnicas de comparação de modelos. Uma abordagem de ER, conforme apresentado, se inicia pela aplicação de técnicas de análise estática e dinâmica sobre o sistema legado, disponível na forma de código e/ou executável. As informações brutas extraídas precisam ser abstraídas para o nível de projeto detalhado ou arquitetural, a fim de facilitar a compreensão do software. Nesse contexto, abordagens de recuperação de arquitetura empregam técnicas de agrupamento de elementos, detecção de padrões, reconhecimento de planos de programação, mineração de dados etc., para abstrair uma representação arquitetural do software.

Algumas abordagens de ER pesquisadas aplicam técnicas de mineração de dados, principalmente sobre modelos estáticos, para abstrair um modelo de alto nível. Técnicas de mineração de dados podem ser de grande utilidade nesse contexto, levando à descoberta de conhecimento implícito sobre a massa "bruta" de dados.

Dentre as técnicas de ER, a análise dinâmica gera um grande volume de informação resultante do monitoramento do sistema em tempo de execução. Dessa forma, as abordagens que extraem modelos do sistema com base na análise dinâmica

apresentam estratégias para a redução do volume de informação, como filtragem, partição dos rastros de execução, ocultação de informação etc.

Em relação às abordagens de recuperação de arquitetura, percebe-se que alguns pontos ainda se apresentam em aberto, como a possibilidade de integração dos resultados a um ADS, além de abordagens genéricas e que extraiam diferentes visões arquiteturais. Cada abordagem extrai um tipo de elemento arquitetural, que pode representar, em geral, um conceito do domínio ou um componente de estilo arquitetural. Nesse contexto, nenhuma abordagem de recuperação de arquitetura é adequada a todos os sistemas e todos os objetivos pretendidos com a ER, onde o cenário ideal é um ADS que incorpore diferentes abordagens.

As técnicas de comparação de modelos podem ser complementares à ER no sentido de detectar opcionalidades e variabilidades entre sistemas de um domínio, evolução entre um modelo extraído e seu estágio atual, compatibilidade entre um modelo original do software e o modelo recuperado etc. Nesta tese, o objetivo é utilizar uma técnica de comparação de modelos para apoiar a criação de arquiteturas de referência de domínio. Nesse sentido, variabilidades e opcionalidades devem ser detectadas entre modelos arquiteturais de sistemas recuperados de um domínio por meio de técnicas de ER. Porém, as abordagens de comparação de modelos pesquisadas não visam à detecção de variabilidades, além de não considerarem sistemas diferentes de um mesmo domínio, os quais podem apresentar diferenças de nomenclatura e estruturais entre os elementos de diferentes modelos.

Capítulo 4 – LegaToDSSA: Abordagem de Apoio à Criação de Arquiteturas de Referência de Domínio

4.1 – Introdução

Conforme apresentado no capítulo 2, seção 2.3, no contexto de Engenharia de Domínio (ED) e Linha de Produtos (LP) são encontradas abordagens que dão um apoio mais efetivo à especificação de arquiteturas de referência através de processos de cima para baixo (BRAGA, 2000; XAVIER, 2001; ATKINSON *et al.*, 2002; KANG *et al.*, 2002; GOMAA, 2004; BLOIS, 2006). Considerando que sistemas legados representam uma das fontes de informação essenciais do domínio, embutindo conhecimento sobre o negócio que muitas vezes não pode ser obtido de nenhuma outra fonte, percebe-se que abordagens que se baseiem na análise desses sistemas, através de processos que partam de baixo para cima, oferecendo um apoio efetivo à criação de arquiteturas de referência, ainda são necessárias.

Como os sistemas legados, em geral, não apresentam documentação atualizada de apoio para facilitar a sua compreensão, os métodos de baixo para cima devem iniciar por alguma abordagem de Engenharia Reversa (ER). Os modelos resultantes da ER devem ser avaliados, uma vez que os mesmos poderão ser reutilizados em diversas aplicações do domínio e, conseqüentemente, devem possuir algum grau de qualidade assegurado. Finalmente, os modelos avaliados devem ser comparados, a fim de que possam ser detectadas as variabilidades e opcionalidades do domínio em nível arquitetural.

Como visto no capítulo 2, na seção 2.3.2, STOERMER e O'BRIEN (2001) e GOMAA (2004) propõem métodos de apoio à especificação de arquiteturas de referência através da análise de sistemas legados no domínio, entretanto, não oferecem apoio efetivo às etapas do processo, como a comparação das arquiteturas. Além disso, outras abordagens que se baseiam na análise de sistemas legados no domínio, não visam a especificação de arquiteturas de referência, estando focadas em um aspecto específico, como a extração de componentes (GANESAN e KNODEL, 2005) ou a refatoração das variabilidades no código (ALVES *et al.*, 2007). Em geral, as abordagens de ED e LP que se baseiam na análise de sistemas legados não propõem um processo sistemático,

com atividades e critérios adequadamente definidos, para apoiar efetivamente a criação de arquiteturas de referência.

Diante desse contexto, esta tese propõe uma abordagem de apoio à criação de arquiteturas de referência de domínio (VASCONCELOS e WERNER, 2004a; VASCONCELOS e WERNER, 2007a), denominada LegaToDSSA, que cobre 2 etapas, realizadas através de 2 sub-abordagens, a saber : 1) ArchMine, que visa a reconstrução de modelos arquiteturais estáticos e dinâmicos de sistemas legados OO; 2) e ArchToDSSA, que cobre a detecção de opcionalidades e variabilidades, e criação de arquiteturas de referência de domínio parcialmente especificadas. A Figura 4.1 ilustra uma visão geral da abordagem proposta para apoio à criação de arquiteturas de referência.

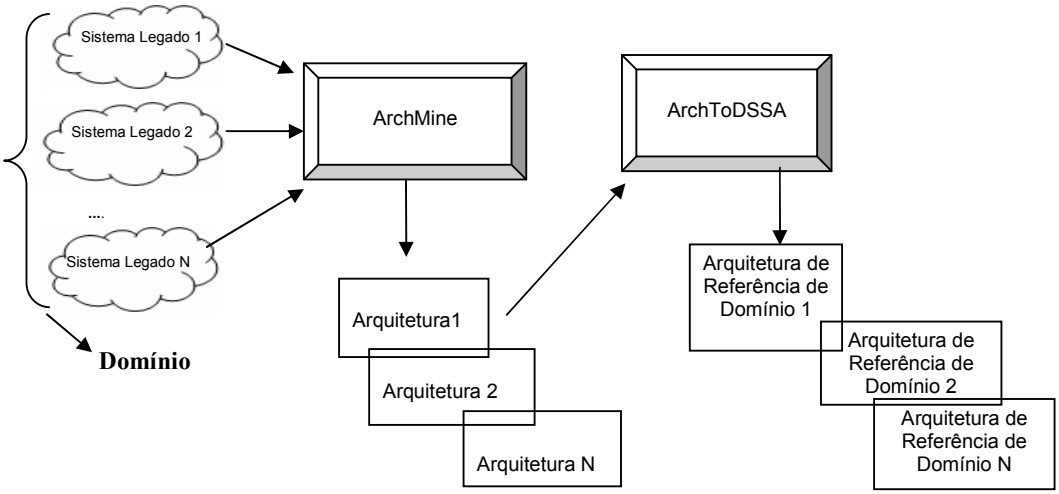

Figura 4.1: Visão geral da abordagem LegaToDSSA: apoio à criação de arquiteturas de referência de domínio.

ArchMine recupera a arquitetura de um sistema legado por vez, gerando um conjunto de arquiteturas de um domínio como entrada para ArchToDSSA. Sobre esse conjunto, ArchToDSSA pode gerar diferentes arquiteturas de referência de domínio parcialmente especificadas, sendo gerada uma por vez.

A integração entre as etapas da abordagem proposta se dá através dos artefatos gerados e consumidos entre elas. ArchMine (VASCONCELOS e WERNER, 2004b; VASCONCELOS *et al.*, 2005; VASCONCELOS e WERNER, 2005a; VASCONCELOS e WERNER, 2005b; CEPEDA *et al.*, 2006) representa a abordagem de Engenharia Reversa (ER) que visa a recuperação de modelos arquiteturais estáticos e dinâmicos para sistemas legados OO. Assumindo que os sistemas legados são "nebulosos", por não apresentarem documentação atualizada de apoio que facilite a sua

compreensão, ArchMine aplica técnicas de análise estática e dinâmica para a extração de informações sobre o software, além de técnicas de mineração de dados e agrupamentos de elementos para elevar o nível de abstração dos modelos extraídos. ArchMine recupera a arquitetura de 1 sistema legado por vez, sendo que, segundo STOERMER e O'BRIEN (2001), para a criação de arquiteturas de referência de domínio, um número de 3 a 4 sistemas deve ser analisado. Além de contribuir com a criação de arquiteturas de referência, ArchMine também apóia a manutenção dos sistemas legados, através da reconstrução de modelos que facilitam a sua compreensão.

Os modelos arquiteturais recuperados são avaliados através da abordagem ArqCheck (BARCELOS e TRAVASSOS, 2006; BARCELOS, 2006), que foi estendida nesta tese para a avaliação do atributo de qualidade Reusabilidade, conforme será visto na seção 4.2.6. ArqCheck, como descrito na seção 2.2.3, representa uma abordagem de avaliação arquitetural baseada em inspeção, que utiliza como técnica de detecção de defeitos na arquitetura um *checklist*. ArqCheck é utilizada na abordagem ArchMine, na etapa de avaliação da arquitetura recuperada.

Finalmente, a fim de consolidar o apoio à criação de arquiteturas de referência, as arquiteturas recuperadas dos sistemas legados no domínio são analisadas através de ArchToDSSA (KÜMMEL e WERNER, 2006; KÜMMEL, 2007), sendo identificadas as suas variabilidades e opcionalidades. ArchToDSSA permite a geração de diferentes DSSAs a partir de um mesmo conjunto de arquiteturas legadas, gerando uma DSSA por vez.

Este capítulo descreve cada uma das etapas da abordagem de apoio à criação de arquiteturas de referência de domínio a partir de sistemas legados, estando organizado da seguinte forma: a seção 4.2 descreve a etapa de recuperação de arquitetura, realizada através de ArchMine, demonstrando as suas contribuições em relação às abordagens de recuperação de arquitetura apresentadas no capítulo 3, na seção 3.2.3; a seção 4.3 descreve a etapa de comparação de arquiteturas, realizada através de ArchToDSSA, que visa suprir limitações para a comparação de arquiteturas de diferentes sistemas em um domínio, encontradas nas abordagens de comparação de modelos existentes; por fim, a seção 4.4 apresenta as considerações finais em relação à abordagem proposta.

4.2 – ArchMine: Recuperação de Arquitetura de Sistemas Legados

Esta seção descreve a abordagem de recuperação de arquitetura proposta, i.e. ArchMine. Nesta tese, um conjunto de requisitos foi definido para uma abordagem de recuperação de arquitetura que vise apoiar a compreensão do sistema para fins de manutenção e a sua reutilização no contexto de ED e LP. Esses requisitos foram definidos a partir da análise de princípios sobre arquitetura de software, como a necessidade de representação de diferentes perspectivas arquiteturais, além de uma abordagem de avaliação da arquitetura que possa indicar as inconsistências entre as diferentes perspectivas e as inconsistências referentes ao atendimento a requisitos. Foram analisados também conceitos sobre abordagens de ED e LP, como a necessidade de representação de características do domínio, com suas variabilidades e opcionalidades.

Além disso, com base na análise das abordagens de recuperação de arquitetura existentes, verificou-se que para o apoio à manutenção, os modelos arquiteturais recuperados devem permitir a localização de uma funcionalidade na arquitetura e no código, além de representar as dependências entre elementos nos modelos estático e dinâmico, ajudando a estimar os impactos de uma manutenção. A partir da análise dessas abordagens, observou-se que a generalidade em relação a domínio e linguagem de programação e o apoio ferramental, integrado a um ambiente de desenvolvimento, também deveriam ser objetivos perseguidos. A fim de detalhar o problema que visa ser solucionado através de ArchMine, esses requisitos são descritos como se segue:

- **Elemento arquitetural que represente um conceito do domínio:** o elemento arquitetural deve representar um conceito do domínio, i.e. um conjunto de funcionalidades ou serviços coesos que se refiram a um conceito ou a uma característica do domínio. Dessa forma, ele pode ser visto como uma unidade no domínio que possui uma variabilidade e uma opcionalidade, podendo ser incluído ou não na instanciação de aplicações. Esse elemento arquitetural pode representar um candidato a ser refatorado para um componente de software do domínio.
- **Nomes com semântica e, preferencialmente, gerados automaticamente**: a fim de que os elementos arquiteturais recuperados representem conceitos do domínio de fato, esses devem possuir um nome com semântica que retrate o

conceito representado. Além disso, o ideal é que esses nomes sejam gerados de forma automática, a fim de não exigir grande esforço do desenvolvedor que recupera a arquitetura na atribuição de nomes, dadas as grandes dimensões dos sistemas legados.

- **Recuperação de múltiplas visões arquiteturais, separando aspectos estáticos e dinâmicos**: a arquitetura deve ser representada a partir de diferentes perspectivas, que possam apoiar mais efetivamente a compreensão do domínio. Destaca-se aqui a importância de uma visão estática dos elementos arquiteturais e seus relacionamentos, que permita a compreensão da estrutura e conceitos que compõem o domínio, e de uma visão dinâmica, que represente o comportamento do sistema em sua arquitetura, destacando questões de comunicação e de concorrência.
- **Rastro para requisitos funcionais**: a visão dinâmica deve estar associada a cenários de uso do sistema, permitindo o rastreamento de requisitos funcionais para elementos arquiteturais e elementos que compõem a implementação do software.
- **Abordagem independente de domínio e de linguagem de programação**: a análise das abordagens de recuperação de arquitetura existentes demonstrou que, em geral, elas são baseadas em critérios para a reconstrução de elementos arquiteturais específicos para uma determinada linguagem de programação, para uma aplicação em particular ou para um domínio de aplicação. Embora não seja viável conceber uma abordagem de recuperação de arquitetura que seja adequada a todos os sistemas, em diferentes domínios, utilizando diferentes paradigmas e linguagens, e seguindo diferentes formas de organização estrutural, uma abordagem de recuperação de arquitetura deve ser o mais genérica quanto possível, a fim de poder ser integrada a um ambiente de desenvolvimento de software e reutilizada em diferentes contextos.
- **Abordagem para avaliação da arquitetura recuperada**: a arquitetura deve ser avaliada através de uma abordagem que imponha algum formalismo à avaliação, evitando que inconsistências na arquitetura se propaguem na sua reutilização em novas aplicações do domínio.
- **Apoio ferramental**: um apoio ferramental deve ser provido ao processo de recuperação de arquitetura, a fim de reduzir o esforço humano inerente a uma

atividade dessa natureza.

- **Integração a um ambiente de reutilização**: a fim de que os modelos recuperados possam ser reutilizados na elaboração de uma arquitetura de referência de domínio, a abordagem de recuperação de arquitetura deve poder ser integrada a um ambiente de desenvolvimento de software com apoio à reutilização.

Convém ressaltar que o atendimento aos 2 últimos requisitos descritos, i.e. apoio ferramental e integração a um ambiente de reutilização, são detalhados no capítulo 5, que apresenta o ambiente e ferramental de apoio à execução do processo proposto em ArchMine.

Em relação ao requisito de generalidade, i.e. independência de domínio e de linguagem de programação, ArchMine é uma abordagem de recuperação de arquitetura independente de linguagem de programação, porém, voltada à análise de sistemas OO, onde a unidade sob análise para a reconstrução dos elementos arquiteturais é a classe. Abordagens que recuperam a arquitetura de sistemas procedurais costumam apresentar maior flexibilidade e trabalhar no nível de granularidade de arquivo ou de funções e variáveis.

De acordo com os requisitos estabelecidos, ArchMine visa à recuperação de um modelo arquitetural que represente conceitos do domínio e suas ligações. Conforme ressaltado nesta tese, um sistema de software pode possuir mais de um modelo arquitetural válido para a sua compreensão. ArchMine utiliza prioritariamente técnicas de análise dinâmica e de mineração de dados para a reconstrução dos elementos arquiteturais. Os elementos arquiteturais são reconstruídos aplicando-se uma técnica de mineração de dados sobre os rastros de execução, i.e. regras de associação (AGRAWAL e SRIKANT, 1994), com o objetivo de detectar classes funcionalmente coesas que participam juntas de um conjunto de cenários de casos de uso e que devem compor elementos arquiteturais.

O algoritmo desenvolvido em ArchMine não é hierárquico, ou seja, ele não trabalha recursivamente, agregando elementos arquiteturais em elementos maiores. Na realidade, o algoritmo de ArchMine propõe uma partição do sistema, representando uma abordagem de remodularização. Dessa forma, não há a necessidade de lidar com o problema de decidir o nível de abstração na hierarquia de onde extrair os elementos

arquiteturais de interesse. A hierarquia estabelecida em ArchMine é no sentido dos elementos arquiteturais para os seus módulos constituintes.

ArchMine representa os modelos arquiteturais em UML e seguindo parcialmente o modelo de visões 4+1 (KRUCHTEN, 1995). A adoção da UML se deve ao fato dessa representar uma linguagem padrão de modelagem e ser menos complexa do que as ADLs, facilitando a compreensão da arquitetura por diferentes *stakeholders* e a manipulação das suas diferentes visões.

Conforme mostra Figura 4.2, a visão lógica é representada através de diagramas de pacotes e de classes; a visão de processo, através de diagramas de seqüência; e a visão de cenários ou de casos de uso é descrita através da associação dos diagramas de seqüência a cenários de casos de uso do sistema.

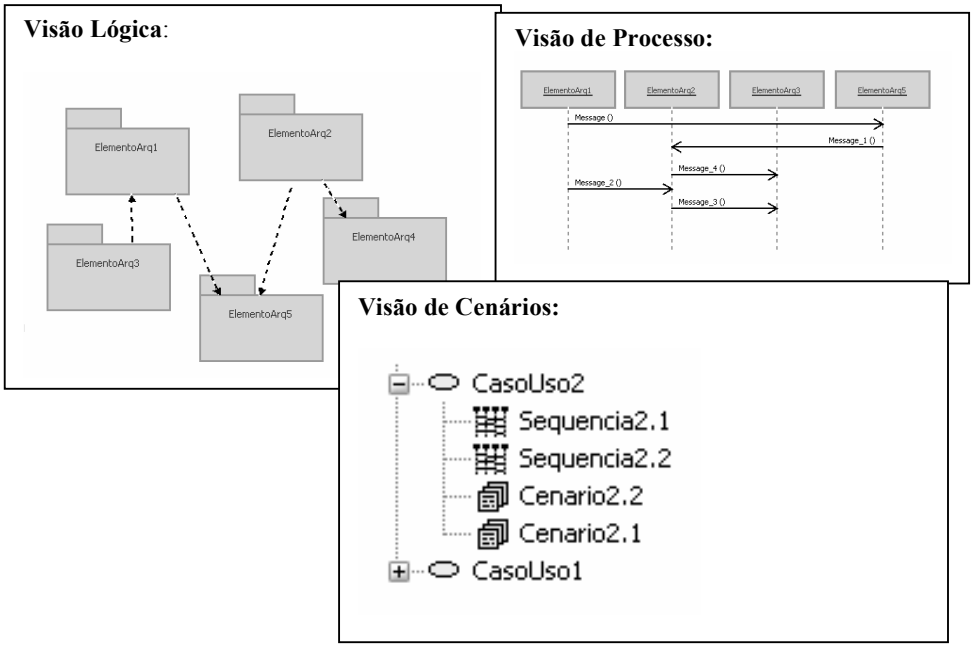

Figura 4.2: Visões arquiteturais em ArchMine.

ArchMine é semi-automatizada e a recuperação da arquitetura deve ser realizada por um desenvolvedor com interesse em compreender o sistema, ou por um Engenheiro de Domínio ou desenvolvedor participando de um processo de ED ou de LP.

O processo de recuperação de arquitetura de ArchMine é apresentado na Figura 4.3. Convém ressaltar que todos os processos apresentados nesta tese seguem a notação de modelagem de processos baseada no metamodelo SPEM[8] (*Software Process*

[8] O metamodelo SPEM, definido pela OMG, faz uso de uma notação estendida de diversos modelos da UML com o intuito de permitir a modelagem de processos.

Engineering Metamodel).

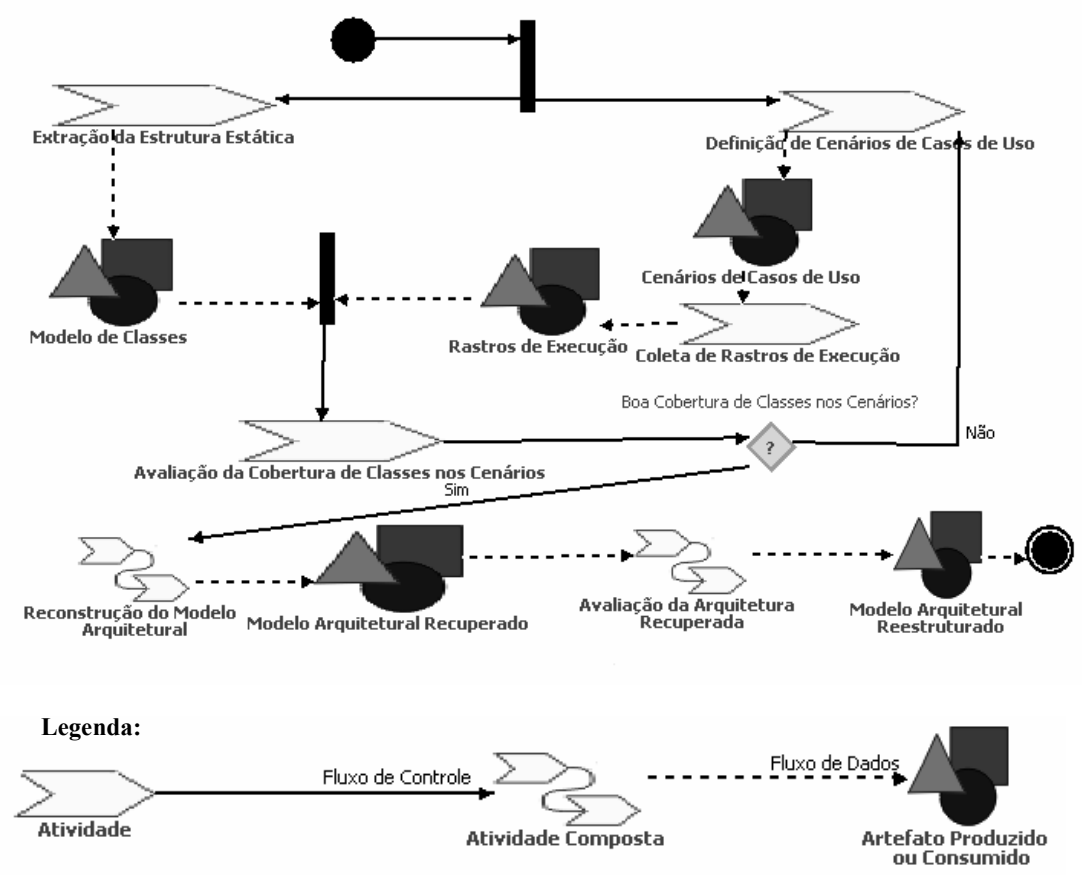

Figura 4.3: Processo de recuperação de arquitetura da abordagem ArchMine.

O processo de recuperação de arquitetura em ArchMine se inicia pelas atividades de **Extração da Estrutura Estática** e **Definição dos Cenários de Casos de Uso**. Essas atividades podem ser executadas em paralelo, uma vez que uma não representa pré-requisito para a outra.

A **Extração da Estrutura Estática** é feita com base na análise do código fonte do sistema e gera como saída um modelo de classes do sistema alvo. A **Definição de Cenários de Casos de Uso** visa à definição dos cenários de uso do sistema que vão guiar a **Coleta dos Rastros de Execução** através da análise dinâmica. Essa última atividade deve utilizar como artefato de entrada, além dos cenários de casos de uso, o código fonte, binário ou executável do sistema, dependendo da linguagem de programação e ferramental disponível para análise dinâmica. Convém ressaltar que na Figura 4.3, que descreve o processo de ArchMine, são apresentados apenas os artefatos produzidos e consumidos entre as atividades de ArchMine. Artefatos consumidos, como o código fonte ou binário do sistema, não são representados por não serem produzidos

pelas atividades de ArchMine.

A **Avaliação da Cobertura de Classes nos Cenários** compara as classes do modelo estático com as classes monitoradas nos rastros de execução, verificando o percentual de classes monitorado e conduzindo a uma tomada de decisão referente à cobertura de classes atingida. Caso a cobertura seja considerada "boa" pelo desenvolvedor que recupera a arquitetura, a atividade de **Reconstrução do Modelo Arquitetural** pode ser iniciada, gerando o modelo arquitetural recuperado por ArchMine. Caso contrário, a atividade de **Definição de Cenários de Casos de Uso** é reexecutada, podendo ser definidos novos cenários ou redefinidos alguns cenários já monitorados. Esses novos cenários ou cenários redefinidos devem ser reexecutados e monitorados, sendo coletados os rastros de execução correspondentes.

A **Reconstrução do Modelo Arquitetural**, como mostra a Figura 4.3, representa uma atividade complexa e, portanto, composta de subatividades, reconstruindo tanto um modelo arquitetural estático do sistema, quanto modelos dinâmicos, esses últimos tanto em nível arquitetural quanto em nível de projeto detalhado.

Finalmente, a arquitetura recuperada deve ser avaliada. A atividade de **Avaliação da Arquitetura Recuperada** utiliza como artefato de entrada o modelo arquitetural recuperado e gera como saída a arquitetura reestruturada com base nas inconsistências identificadas.

As seções a seguir apresentam a descrição detalhada das atividades de ArchMine. Um exemplo envolvendo um sistema de controle escolar hipotético é utilizado para ilustrar algumas atividades do processo. Ao final, é apresentada uma comparação entre ArchMine e as abordagens de recuperação de arquitetura apresentadas na seção 3.2.3.

4.2.1 – Extração da Estrutura Estática

A **Extração da Estrutura Estática** visa à extração de um diagrama de classes da UML a partir do código fonte de um sistema OO. O diagrama de classes representa um grafo bidirecional que descreve as entidades do código, i.e. classes e interfaces do software, implementadas em uma determinada linguagem de programação, com seus relacionamentos de herança, associação, dependência e realização (ou implementação), esse último representando um relacionamento entre classe e interface.

Esse modelo de classes extraído do código representa o modelo da

implementação do software e se encontra em um baixo nível de abstração. As suas classes são posteriormente agrupadas em elementos arquiteturais.

4.2.2 – Definição de Cenários de Casos de Uso

A **Definição de Cenários de Casos de Uso** visa à definição de cenários de uso do sistema para guiar a sua execução durante a análise dinâmica. A fim de compreender os objetivos e passos envolvidos nessa atividade, algumas definições se fazem necessárias. Convém ressaltar que o objetivo da definição dos cenários, para a recuperação de arquitetura, é exercitar o maior número possível de classes do sistema no maior número possível de cenários de casos de uso distintos onde elas são utilizadas.

Um caso de uso (**uc**) representa a descrição de seqüências de ações, incluindo variantes, que uma entidade (ex: um sistema) executa para produzir um resultado de valor observável para um ator (BOOCH *et al.*, 1998). Um **uc** representa um requisito funcional (**rf**) do sistema, indicando que: **uc** $\Leftrightarrow$ **rf**. Cada seqüência de ações em um **uc** determina um cenário de caso de uso, representando uma forma de obter aquele **rf** (BOOCH *et al.*, 1998). Além disso, seqüências de eventos disparados por ações do usuário no sistema (como seleção de opção de menu, preenchimento de campos, clique de botão etc.) levam à execução de operações do sistema que implementam ou realizam um cenário de caso de uso, gerando cenários concretos. Cada **uc** possui vários cenários de casos de uso relacionados, sendo que cada cenário de caso de uso possui vários cenários concretos relacionados, ou seja, vários caminhos de execução no software para se obter aquele cenário. Vale ressaltar que os conceitos de caso de uso e de cenário de caso de uso são provenientes da UML, ao passo que a definição de cenário concreto representa uma contribuição deste trabalho.

Sobre esses conceitos definidos, as diretrizes listadas na Tabela 4.1, a seguir, devem direcionar a definição dos cenários de casos de uso. Essas diretrizes foram sendo refinadas à medida que estudos experimentais foram realizados sobre a abordagem de recuperação de arquitetura ArchMine, conforme descritos no capítulo 6.

Essa lista de diretrizes não pretende ser exaustiva. Ela segue a forma como os requisitos funcionais são disponibilizados aos usuários na maioria dos sistemas OO.

Entretanto, a participação de um *stakeholder* com conhecimento do funcionamento do sistema é importante, nesse momento, para garantir a definição de cenários de casos de uso que não são descobertos pelas diretrizes sugeridas. Além disso, manuais de usuário e documentação de requisitos do sistema podem ser consultados

para apoiar essa atividade, caso se encontrem disponíveis. Deve-se levar em conta, entretanto, que, em geral, esses documentos não correspondem à versão atual do sistema em execução, devendo ser utilizados de forma complementar às diretrizes propostas.

Tabela 4.1: Diretrizes para a definição de cenários de casos de uso em ArchMine.

Diretriz para a Definição de Cenário de Caso de Uso	Descrição
DUC_1	Para cada opção de menu principal e menu *popup* derive um cenário de caso de uso. Se as opções contiverem subgrupos de opções, então escolha a opção no último nível da hierarquia para a derivação do cenário.
DUC_2	Preencha todos os dados de entrada dos painéis disponibilizados quando uma opção de menu é acionada, garantindo assim que classes que tratem dos diferentes valores de entrada serão acionadas.
DUC_3	Para cada conjunto de cenários concretos que correspondam ao mesmo cenário de caso de uso, como, por exemplo, o mesmo cenário executado através de uma opção de menu principal e de menu *popup*, apenas um cenário concreto deve ser executado.
DUC_4	Botões de barras de ferramentas também devem ser consultados como candidatos à derivação de cenários de casos de uso, entretanto, deve ser respeitada a heurística DUC_3.
DUC_5	Abas de painéis com abas (t*abbed pan*es) podem indicar cenários de casos de uso distintos ou painéis distintos para o preenchimento de dados de um objeto. A semântica dos painéis deve ser investigada nesse caso e a heurística DUC_3 deve ser respeitada.
DUC_6	Botões não triviais em painéis, i.e. que não sejam "Ok", "Cancel", "Next", "Back", "Close", "Finish" etc., podem indicar cenários de casos de uso distintos em relação à opção de menu que levou à abertura do painel.
DUC_7	A inicialização do sistema, ex: *login* de usuário, deve derivar um cenário de caso de uso.
DUC_8	Cenários de exceção devem gerar cenários de casos de uso sempre que exercitarem classes do sistema não exercitadas nos demais cenários definidos para o caso de uso.

Vale ressaltar que o objetivo dessa atividade não é atingir uma cobertura de 100% em relação ao conjunto total de cenários de casos de uso do sistema. Porém, quanto maior for essa cobertura, melhor tende a ser a qualidade da arquitetura recuperada. Na realidade, o maior objetivo dessa atividade é permitir que a execução dos cenários de casos de uso definidos exercite o maior número possível de classes do sistema em diferentes situações nas quais as mesmas são utilizadas. Quanto maior for o conhecimento do *stakeholder* que apóia essa atividade em relação à estrutura do sistema, melhor tende a ser a qualidade da arquitetura recuperada.

Os cenários de casos de uso podem ser descritos informalmente, apontando o seu objetivo, as opções na interface do sistema que levam à sua execução e dados de entrada que podem ser informados. Dessa forma, cada cenário de caso de uso definido é descrito

considerando-se uma das opções de cenário concreto através do qual o mesmo pode ser executado no sistema.

A atividade de **Definição de Cenários de Casos de Uso** contribui com a reconstrução de uma documentação atualizada dos requisitos funcionais do sistema, além de servir de base para a coleta dos rastros de execução, a próxima atividade de ArchMine. Convém ressaltar, entretanto, que a reconstrução de um modelo de casos de uso, com a descrição de cenários de casos de uso, é um problema difícil na área de ER e, através das diretrizes sugeridas, ArchMine oferece alguns guias para essa atividade, embora contribuições nessa área não representem o foco do trabalho.

A Tabela 4.2, a seguir, apresenta um conjunto de cenários de casos de uso para um sistema de controle escolar hipotético.

Tabela 4.2: Exemplo de um conjunto de cenários de casos de uso para um sistema de controle escolar hipotético.

Cenários de Casos de Uso	Caso de Uso
1. Informar notas de estudantes de graduação.	Informar Notas
2. Informar notas de estudantes de pós-graduação.	Informar Notas
3. Imprimir grade de notas de estudantes.	Imprimir Grade de Notas
4. Inscrever estudantes de graduação em turmas.	Inscrever Estudantes em Turmas
5. Inscrever estudantes de pós-graduação em turmas.	Inscrever Estudantes em Turmas
6. Imprimir inscrições de estudantes.	Imprimir Inscrições de Estudantes

4.2.3 – Coleta de Rastros de Execução

A **Coleta de Rastros de Execução** envolve a análise dinâmica da aplicação, através da execução monitorada do sistema guiada pelos cenários de casos de uso definidos na atividade anterior. O objetivo é detectar as classes e seqüências de chamadas de métodos que implementam esses cenários. A análise dinâmica é utilizada a fim de detectar exatamente que tipos de objetos (i.e. classes ou interfaces) e métodos são executados para a realização de um cenário de caso de uso, o que não seria possível através da análise estática, em função da vinculação tardia e polimorfismo do paradigma OO (DEITEL e DEITEL, 2002).

Os rastros de execução devem conter as seqüências de execução de métodos que implementam os cenários de casos de uso, ordenados hierarquicamente por ordem de chamada de método. Cada método registrado no rastro deve conter o seu nome, a instância que recebe a chamada de método, o seu tipo (i.e. classe ou interface) e o fluxo de execução (*thread*) no qual o método é executado. Todas essas informações são utilizadas na reconstrução do modelo arquitetural dinâmico, representado através de diagramas de seqüência da UML. As informações a respeito das classes que

implementam os cenários de casos de uso e da *thread* são utilizadas na reconstrução dos elementos arquiteturais do sistema.

A Tabela 4.3, a seguir, retoma o conjunto de cenários de casos de uso para um sistema de controle escolar hipotético, apresentado na Tabela 4.2, com as respectivas classes que os implementam. Essa relação de cenários x classes é estabelecida em ArchMine através desta atividade de **Coleta de Rastros de Execução**.

Tabela 4.3: Exemplo de um conjunto de cenários de casos de uso para um sistema de controle escolar hipotético com as classes que os implementam.

Cenário de Caso de Uso	Classes que Implementam o Cenário
1. Informar notas de estudantes de graduação.	Estudante, EstudanteGraduação, GradeNotas
2. Informar notas de estudantes de pós-graduação.	Estudante, EstudantePosGraduação, GradeNotas
3. Imprimir grade de notas de estudantes.	Estudante, EstudanteGraduação, EstudantePosGraduação, ImpressoraUtils, ImpressoraConfig, GradeNotas
4. Inscrever estudantes de graduação em turmas.	Estudante, EstudanteGraduação, Classes, Turmas, Inscrições
5. Inscrever estudantes de pós-graduação em turmas.	Estudante, EstudantePosGraduação, Classes, Turmas, Inscrições
6. Imprimir inscrições de estudantes.	Estudante, EstudanteGraduação, EstudantePosGraduação, Classes, Turmas, Inscrições, ImpressoraUtils, ImpressoraConfig, RelatUtils

4.2.4 – Avaliação da Cobertura de Classes nos Cenários

O objetivo dessa atividade é verificar se os cenários de casos de uso monitorados apresentam uma boa cobertura em relação às classes do sistema. Essa cobertura é avaliada verificando se os rastros de execução contêm as classes extraídas do código fonte com a análise estática. Dessa forma, as classes do modelo de classes devem ser comparadas com as classes constantes dos rastros de execução. Caso sejam detectadas classes do sistema que não foram monitoradas, a atividade de **Definição de Cenários de Casos de Uso** pode ser reexecutada, conforme mostra a decisão e a iteração no processo de ArchMine apresentado na Figura 4.3. O objetivo não é atingir uma cobertura de 100% das classes monitoradas, mas quanto maior for essa cobertura, melhor tende a ser a qualidade da arquitetura recuperada. A decisão a respeito da reexecução da atividade de **Definição de Cenários de Casos de Uso** é de responsabilidade do desenvolvedor que recupera a arquitetura.

Nessa reexecução, as diretrizes de definição de cenários de casos de uso devem ser revisitadas, a documentação disponível do sistema para apoio à atividade também pode ser novamente consultada, bem como o *stakeholder* que apoiou a execução inicial

da atividade pode ser novamente acionado. Uma lista das classes não monitoradas pode ser gerada e apresentada ao *stakeholder*, a fim de que ele tente descobrir cenários que ainda devem ser definidos (ou redefinidos) e monitorados. Nesse último caso, é exigido um *stakeholder* com conhecimento sobre a estrutura do sistema em seu ambiente de implementação, como um programador, mantenedor, desenvolvedor etc.

4.2.5 – Reconstrução do Modelo Arquitetural

A **Reconstrução do Modelo Arquitetural** é uma atividade composta de subatividades, as quais estão representadas no diagrama da Figura 4.4. Conforme mostra o diagrama, a atividade se inicia pela **Reconstrução de Elementos Arquiteturais,** passando pelas atividades de **Derivação de Nomes e de Relacionamentos entre Elementos Arquiteturais** e **Reconstrução de Modelos Dinâmicos**.

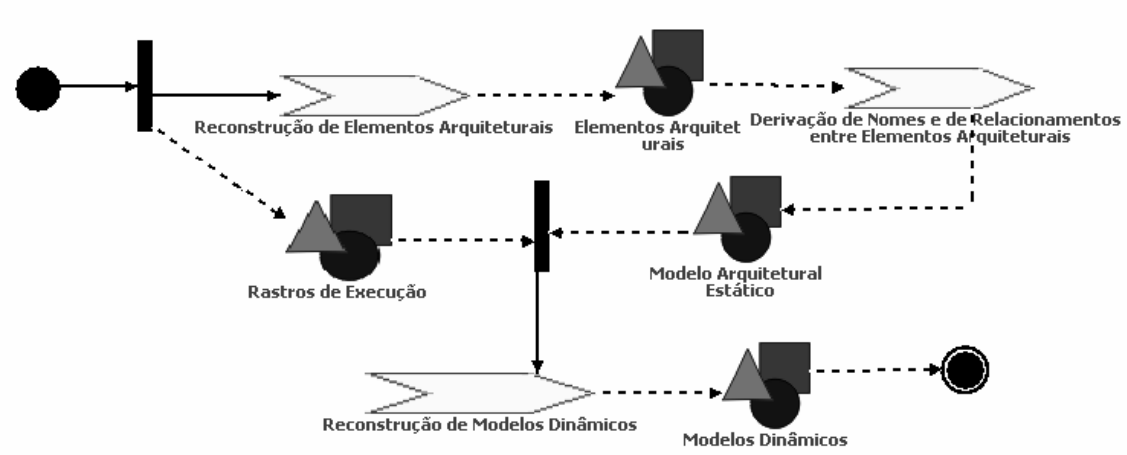

Figura 4.4: Subatividades para a Reconstrução do Modelo Arquitetural.

Reconstruído o modelo arquitetural estático, a atividade de **Reconstrução de Modelos Dinâmicos** é iniciada. Ela utiliza como artefatos de entrada o modelo arquitetural estático reconstruído e os rastros de execução coletados na atividade de **Coleta de Rastros de Execução**, gerando como saída modelos dinâmicos tanto em nível arquitetural quanto em nível de projeto detalhado.

As subatividades da atividade de **Reconstrução do Modelo Arquitetural** são detalhadas a seguir.

4.2.5.1 – Reconstrução de Elementos Arquiteturais

A reconstrução objetiva o agrupamento de classes funcionalmente coesas em elementos arquiteturais que representem conceitos do domínio. A fim de detectar

classes funcionalmente coesas, i.e. que implementam um conjunto comum de funcionalidades do sistema, um algoritmo inspirado no Apriori, descrito na seção 3.2.2, é utilizado para minerar os rastros de execução e descobrir regras de associação entre classes.

O Apriori (AGRAWAL e SRIKANT, 1994) é adaptado nesta tese, a fim de permitir a escalabilidade na interpretação dos resultados retornados no processo de mineração, que pode se tornar uma tarefa árdua em função do grande número de padrões retornado. A variação do Apriori utilizada, ao invés de detectar conjuntos de itens freqüentes e derivar as regras de associação sobre esses itens, obtém um antecedente como valor de entrada para a mineração, a base de transações e os valores de confiança e suporte mínimos. O algoritmo conta o número total de transações na base e separa o subconjunto válido, i.e. aquele que contém o antecedente informado. Nesse subconjunto de transações válidas, o algoritmo verifica em quantas transações cada item de dado, diferente do antecedente, participa, a fim de calcular a confiança da regra de associação de cada item de dado em relação ao antecedente. A confiança é obtida pela seguinte fórmula: *número de ocorrências de um item de dado (≠ antecedente) em diferentes transações onde o antecedente ocorre / número de transações que contêm o antecedente*. O suporte é obtido levando-se em conta o percentual de transações da base em que o antecedente e qualquer outro item de dado, diferente do antecedente, ocorrem em conjunto. O algoritmo retorna regras de associação binárias, com um antecedente e um conseqüente.

Classes que participam de regras de associação com valores de suporte e confiança maiores ou iguais aos valores mínimos informados, são consideradas funcionalmente coesas e candidatas a compor um elemento arquitetural.

A mineração é realizada iterativamente e a cada ciclo só são descobertas regras de associação que possuam como antecedentes as classes informadas. A fim de permitir a utilização de um algoritmo inspirado no Apriori para minerar rastros de execução, alguns conceitos do Apriori são mapeados para o contexto da análise dinâmica, como mostra a Tabela 4.4. Vale ressaltar que nessa Tabela o termo "cenário de caso de uso" refere-se a um cenário de caso de uso definido e monitorado para o sistema.

O processo de mineração e reconstrução dos elementos arquiteturais é conduzido pelo desenvolvedor que recupera a arquitetura, o qual é apoiado pelo conjunto de heurísticas apresentadas a seguir. Essas heurísticas foram derivadas a partir da análise teórica sobre princípios de arquitetura de software, mineração de dados, orientação a

objetos e engenharia reversa, além de terem sido refinadas à medida que a abordagem foi sendo avaliada através de estudos experimentais, conforme descritos no capítulo 6.

Tabela 4.4: Mapeamento de conceitos do algoritmo Apriori para o contexto da análise dinâmica de software.

Conceitos do Apriori	Mapeamentos para o Contexto da Análise Dinâmica
Transação	Um cenário de caso de uso, representado através de um rastro de execução.
Item de Dado	Uma classe que implementa o cenário de caso de uso e consta do seu rastro de execução.
Suporte	Percentual de cenários de casos de uso implementados por uma classe.
Suporte Mínimo	O percentual mínimo de cenários de casos de uso no qual as classes que constam de regras de associação devem aparecer juntas.
Confiança	Percentual de cenários de casos de uso implementados por uma classe X em que uma classe Y também aparece, quando X é o antecedente de uma regra de associação e Y o seu conseqüente.
Confiança Mínima	Percentual mínimo de cenários de casos de uso implementados por uma classe X, em que uma classe Y também deve aparecer, para que a regra de associação entre X e Y seja válida.
Antecedente	Classe que é utilizada como entrada para a descoberta de regras de associação.
Conseqüente	Classe associada ao antecedente com suporte e confiança maiores ou iguais aos valores mínimos informados.

1) **H1**: O **suporte mínimo** deve ser baixo, uma vez que todas ou a maior parte das classes monitoradas devem ser agrupadas em elementos arquiteturais durante a mineração. Se uma classe implementa somente um ou um pequeno número de cenários de casos de uso, ou seja, representa um item de dado que participa de poucas transações, ela deve ser agrupada com as outras classes que também só aparecem nesse(s) cenário(s) de caso(s) de uso.
2) **H2**: A **confiança mínima** deve ser ajustada ao longo do processo de mineração. Esse ajuste pode ser realizado apresentando resultados intermediários a um *stakeholder* (i.e. programador, mantenedor ou desenvolvedor), caso esse se encontre disponível. Observou-se que, considerando valores de Precisão e Revocação (BAEZA-YATES e RIBEIRO-NETO, 1999), onde Precisão (*Precision*) pode ser definida como a fração das classes recuperadas que são relevantes e Revocação (*Recall*), como a fração das classes relevantes que são recuperadas em um elemento arquitetural, quanto maior a confiança mínima, mais precisos tendem a ser os elementos arquiteturais recuperados e quanto menor a confiança mínima, mais completos (i.e. melhor a Revocação) tendem a ser os elementos arquiteturais. Nesse sentido, estudos experimentais (ver capítulo 6)

revelaram que uma confiança mínima de 60% tende a produzir resultados balanceados.

3) **H3**: Classes devem ser mineradas e agrupadas dos maiores para os menores valores de suporte. Classes com altos valores de suporte são as mais gerais, provendo serviços requeridos por várias outras classes. Elas compõem elementos arquiteturais mais gerais que tendem a implementar serviços de infra-estrutura, suporte ou alguma funcionalidade central da aplicação[9]. Elas tendem a aparecer em vários resultados da mineração. Classes com menor suporte provêem serviços mais específicos no sistema. Dessa forma, os **antecedentes** para a mineração são escolhidos dos maiores para os menores valores de suporte, sendo que numa mesma faixa de valor de suporte, os antecedentes podem ser escolhidos aleatoriamente.
4) **H4**: Grupos de classes já formados devem ser **filtrados** dos ciclos de mineração subseqüentes, caso contrário, não seria possível separar grupos de classes mais gerais dos grupos mais específicos.
5) **H5**: Toda vez que ocorrerem **interseções** entre as regras de associação obtidas para diferentes antecedentes, ou seja, toda vez que houver classes em comum nos conjuntos de regras de associação de diferentes antecedentes, essas devem ser priorizadas na composição dos elementos arquiteturais, uma vez que tendem a representar classes fortemente relacionadas. Os próprios antecedentes das regras são incluídos nas interseções. Interseções devem ser calculadas de tal forma que sejam priorizados grupos de classes que apareçam no maior número possível de interseções entre conjuntos de regras de associação de diferentes antecedentes.

   Vale ressaltar que caso o desenvolvedor que recupera a arquitetura julgue que os grupos formados são pequenos, ele pode combinar um grupo de classes que ocorre na interseção de um conjunto de antecedentes com outro grupo de classes que ocorre na interseção de um conjunto maior de antecedentes, do qual o primeiro grupo represente um subconjunto.

   A fim de facilitar a compreensão desta heurística **H5**, a Figura 4.5 apresenta um esquema gráfico da sua aplicação. Nessa Figura, foram minerados como

[9] Convém ressaltar que não é objetivo de ArchMine estabelecer uma classificação dos elementos arquiteturais em função do tipo de serviço oferecido.

antecedentes as classes C_1, C_4 e C_{12}. No resultado da mineração de C_1, retornaram as seguintes classes como conseqüentes nas suas regras de associação: C_2, C_3, C_6, C_4, C_5, C_{11}, C_{12}, C_{16}, C_{17} e C_{18}. Para o antecedente C_4, as seguintes classes retornaram nos conseqüentes das suas regras de associação: C_7, C_8, C_9, C_{10}, C_1, C_5, C_{11}, C_{12}, C_{16}, C_{17} e C_{18}. Para o antecedente C_{12}, retornaram as classes C_{13}, C_{14}, C_{15}, C_{11}, C_{16}, C_{17} e C_{18}. Dessa forma, segundo a heurística **H5**, os seguintes grupos de classes são sugeridos nas interseções: C_{11}, C_{12}, C_{16}, C_{17} e C_{18}; e C_1, C_4 e C_5. Vale lembrar que os próprios antecedentes entram nos cálculos das interseções, como, por exemplo, C_1 e C_4 no cálculo da segunda interseção. Segundo a heurística **H5**, o desenvolvedor que recupera a arquitetura tem a opção de formar esses dois grupos de elementos arquiteturais ou uni-los em um grupo maior.

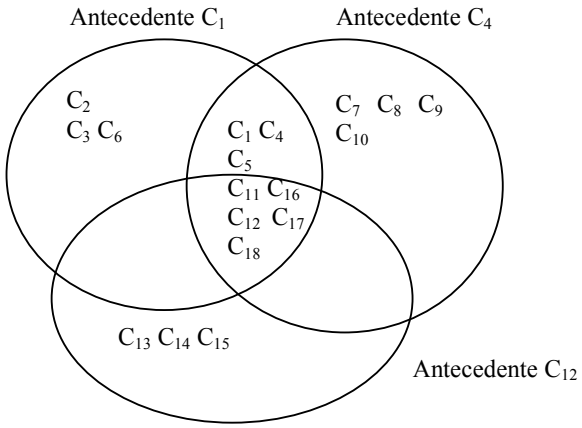

Figura 4.5: Esquema gráfico ilustrativo da heurística H5 do processo de mineração.

6) **H6**: Cada **classe** que porventura **não** seja **agrupada** durante o processo de mineração, deve ser agrupada no elemento arquitetural cujas classes apresentem um suporte mais próximo do seu e um bom valor de confiança para ela. Dessa forma, deve ser calculada a diferença entre a média de suporte das classes do elemento arquitetural e o suporte da classe não agrupada. Quanto mais essa diferença se aproximar de 0%, melhor. A seguinte fórmula, formalizada e descrita a seguir, deve ser utilizada para o cálculo da diferença de suporte (**ds**).

   Assuma que $C=\{c_1,c_2,....,c_n\}$ represente o conjunto de todas as classes do sistema e $EA=\{ea_1, ea_2,..., ea_n\}$ represente o conjunto de elementos arquiteturais gerados com o processo de mineração. Assuma que c_x represente uma classe do sistema tal que $c_x \in C$, que não tenha sido alocada

durante o processo de mineração. Assuma que ea_x represente um elemento arquitetural qualquer do conjunto de elementos arquiteturais gerados tal que $ea_x \in EA$ e que $c_k \in C$ represente uma classe qualquer de ea_x ($c_k \in ea_x$). Assim, a **ds** de c_x em relação a cada ea_x é denotada como $DS_{cx\text{-}eax}$ e deve ser calculada da seguinte forma:

$$DS_{cx\ -\ eax} = \left| \frac{\sum_{k=1}^{|eax|} suporte(c_k)}{|\ ea_x\ |} - suporte(c_x) \right|$$

A fórmula indica que deve ser calculada a média de suporte das classes de um elemento arquitetural e que desta deve ser subtraído o valor de suporte da classe não agrupada, extraindo-se o valor absoluto do resultado obtido.

Deve ser calculada também a confiança de cada classe c_k de cada elemento arquitetural ea_x para a classe não agrupada c_x. Esta confiança é denotada $Conf_{ck\text{-}cx}$. Os valores de confiança obtidos são somados e extrai-se uma média por elemento arquitetural ea_x. Quanto mais esse valor da média de confiança (**mc**) se aproximar de 100%, melhor. Assim, a média de confiança de c_x em relação a cada ea_x é denotada como $MC_{cx\text{-}eax}$ e deve ser calculada através da seguinte fórmula:

$$MC_{cx\ -\ eax} = \frac{\sum_{k=1}^{|eax|} Confc_{k\ -\ c_x}}{|\ ea_x\ |}$$

Deve ser selecionado o elemento arquitetural ea_x para o qual a classe não agrupada c_x apresente a melhor solução de compromisso entre valores de suporte e confiança. A solução de compromisso (**sc**) de c_x em relação a cada ea_x é denotada como $SC_{cx\text{-}eax}$ e pode ser calculada levando-se em conta quanto falta para a **mc** chegar a 100% e a **ds** a 0%, através da seguinte fórmula:

$$SC_{cx\text{-}eax} = (100\% - MC_{cx\text{-}eax}) + DS_{cx\text{-}eax}$$

Quanto menor o valor de **sc**, melhor é a solução de compromisso. Entretanto, caso a semântica da classe não agrupada e dos elementos arquiteturais seja conhecida pelo desenvolvedor que recupera a arquitetura, esse fator também pode ser levado em conta no agrupamento.

7) **H7**: Classes cujos métodos não são executados na ***thread*** principal podem apresentar 3 opções de agrupamento: (1) elas podem ser filtradas do processo de mineração e agrupadas em função das *threads* em que seus métodos são executados, assumindo-se, nesse caso, que classes que participam juntas de *threads* secundárias implementam funcionalidades ou serviços específicos no sistema; (2) elas podem simplesmente ser consideradas itens de dados do cenário de caso de uso onde essas *threads* são executadas; (3) *threads* secundárias podem ser mapeadas para transações distintas em relação ao cenário de caso de uso onde são executadas, sendo incluídas na mineração.
8) **H8**: Finalizada a mineração, **classes não monitoradas**, ou seja, **superclasses e interfaces**, que não recebem mensagens diretamente, devem ser agrupadas no elemento arquitetural que apresente um maior acoplamento para elas, i.e. que apresente a maior parte das suas subclasses, para as superclasses, ou a maior parte das classes que a realizam, para as interfaces.

As 6 primeiras heurísticas para o processo de mineração são exemplificadas através da Tabela 4.5, que apresenta os ciclos de mineração para os cenários de casos de uso de um sistema de controle escolar hipotético, apresentados na Tabela 4.2. A relação de cenários de casos de uso x classes para o sistema de controle escolar hipotético foi apresentada na Tabela 4.3. A título de ilustração, 20% das classes desse exemplo hipotético não são agrupadas durante o processo de mineração, embora nos estudos experimentais realizados com ArchMine um percentual menor de classes não agrupadas tenha sido observado.

Conforme a heurística **H2**, a confiança mínima considerada foi de 60%. Segundo a heurística **H3**, a mineração se inicia pela classe Estudante, que é a que tem o maior valor de suporte, i.e. 100%. EstudanteGraduação e EstudantePosGrauação têm uma confiança de 66,7% para Estudante, derivando o primeiro elemento arquitetural.

As próximas candidatas a antecedentes da mineração são GradeNotas, Classes, Turmas e Inscrições que têm um suporte de 50%. GradeNotas, Classes e Turmas são escolhidas aleatoriamente como antecedentes. Em relação a GradeNotas, nenhuma classe é retornada como conseqüente da mineração, uma vez que somente Estudante, EstudanteGraduação e EstudantePosGraduação têm confiança maior que 60% para ela e, de acordo com a heurística **H4**, essas classes são filtradas da mineração.

De acordo com a heurística **H5**, as classes Turmas, Inscrições e Classes vão compor um mesmo elemento arquitetural, uma vez que compõem interseções entre os resultados da mineração. Seguindo a heurística **H3,** os próximos candidatos a antecedentes são ImpressoraUtils e ImpressoraConfig, com suporte de 33,3%, compondo um elemento arquitetural. Em função da heurística **H1**, mesmo com suporte baixo (i.e. 16,7%), RelatUtils é o próximo antecedente da mineração. Entretanto, em função da heurística **H4**, de filtros de classes já agrupadas, o resultado da sua mineração retorna vazio.

Tabela 4.5: Resultados da aplicação das heurísticas de mineração para o sistema de controle escolar hipotético.

Ciclo de Mineração	Antecedente	Supor-te do Atece-dente	Regras de Associação/ Confiança	Elemento Arquitetural
1	Estudante	100%	Estudante ⇒ EstudanteGraduação – 66,7% Estudante ⇒ EstudantePosGraduação – 66,7%	EA1 = {Estudante, EstudanteGraduação, EstudantePosGraduação}
2	GradeNotas, Classes e Turmas	50%	GradeNotas ⇒ {} Classes ⇒ Turmas-100% Classes ⇒ Inscrições – 100% Turmas ⇒ Classes-100% Turmas ⇒ Inscrições – 100%	EA2 = {Classes, Turmas, Inscrições}
3	ImpressoraUtils	33,3%	ImpressoraUtils ⇒ ImpressoraConfig – 100%	EA3 = {ImpressoraUtils, ImpressoraConfig}
4	RelatUtils	16,7%	RelatUtils => {}	

Finalizada a mineração, as classes RelatUtils e GradeNotas não foram alocadas na arquitetura. Assim, em função da heurística **H6,** RelatUtils é alocada no EA3, junto com ImpressoraUtils e ImpressoraConfig, pois esse elemento é o que apresenta a melhor solução de compromisso para RelatUtils, ou seja, o que apresenta uma menor diferença de suporte (**ds**) e uma melhor média de confiança (**mc**) para RelatUtils. A média de confiança de todos os elementos arquiteturais para RelatUtils, conforme mostra a Tabela, é igual, i.e. 100%. Porém, a diferença de suporte é menor para o EA3.

GradeNotas, por sua vez, apresenta um suporte mais próximo do elemento arquitetural EA2, onde a **ds** é de 0%, como mostra a Tabela 4.6. Porém, as classes desse elemento arquitetural apresentam uma média de confiança (**mc**) de 0% em relação à GradeNotas. Observando-se a **ds** e a **mc** para os demais elementos arquiteturais, percebe-se que GradeNotas apresenta uma melhor solução de compromisso (**sc**) para o

elemento arquitetural EA1.

Tabela 4.6: Alocação de classes não agrupadas na arquitetura.

Classe Não Agrupada	Suporte	Diferença de Suporte e Média de Confiança para cada Elemento Arquitetural	Elemento Arquitetural
RelatUtils	16,7%	$DS_{RelatUtils\text{-}EA1} \cong 61\%$ e $MC_{RelatUtils\text{-}EA1}=100\%$, $SC_{RelatUtils\text{-}EA1} \cong 61\%$. $DS_{RelatUtils\text{-}EA2}=33,3\%$ e $MC_{RelatUtils\text{-}EA2}=100\%$, $SC_{RelatUtils\text{-}EA2}=33,3\%$. $DS_{RelatUtils\text{-}EA3}=16,6\%$ e $MC_{RelatUtils\text{-}EA3}=100\%$, $SC_{RelatUtils\text{-}EA3}=16,6\%$.	EA3 = {ImpressoraUtils, ImpressoraConfig, RelatUtils}
GradeNotas	50%	$DS_{GradeNotas\text{-}EA1} \cong 27,8\%$ e $MC_{GradeNotas\text{-}EA1} \cong 77,8\%$, $SC_{GradeNotas\text{-}EA1} \cong 50\%$. $DS_{GradeNotas\text{-}EA2}=0\%$ e $MC_{GradeNotas\text{-}EA2}=0\%$, $SC_{GradeNotas\text{-}EA2}=100\%$. $DS_{GradeNotas\text{-}EA3}=16,7\%$ e $MC_{GradeNotas\text{-}EA3} \cong 33,3\%$, $SC_{GradeNotas\text{-}EA3} \cong 83,4\%$.	EA1 = {Estudante, EstudanteGraduação, EstudantePosGraduação, GradeNotas}

Como mencionado, essas heurísticas foram derivadas a partir da análise de princípios de arquitetura de software, mineração de dados, orientação a objetos e engenharia reversa. Dessa forma, a heurística **H3** diz respeito a princípios de arquitetura de software, tratando da escolha de antecedentes para a mineração de dados que levam à composição de elementos mais gerais e mais específicos na arquitetura. Heurísticas que dizem respeito a princípios de mineração de dados incluem: a **H1**, que trata do valor de suporte mínimo; a **H2**, que trata do valor de confiança mínima; e a **H5,** que trata das interseções que ocorrem entre os conseqüentes das regras de associação de diferentes antecedentes. Heurísticas que dizem respeito a princípios de OO incluem: a **H7,** que trata questões referentes a sistemas concorrentes, i.e. *multi-thread*; e a **H8**, que trata do agrupamento de classes não mineradas por não constarem dos rastros de execução, como superclasses e interfaces. Finalmente, a heurística que diz respeito a princípios de ER é a **H6**, que trata de classes que podem não ser indicadas para agrupamento durante a mineração.

As heurísticas foram refinadas, e algumas definidas, com base nos resultados dos estudos experimentais apresentados no capítulo 6. A elaboração de um conjunto inicial de heurísticas contou também com a avaliação da arquitetura de sistemas disponíveis no grupo de pesquisa do qual esta tese faz parte, i.e. o Odyssey-MDA (MAIA et al., 2005) e o Ares (VERONESE e NETTO, 2001).

4.2.5.2 – Derivação de Nomes e de Relacionamentos entre Elementos Arquiteturais

A fim de gerar nomes de elementos arquiteturais com semântica e que possam ser derivados automaticamente, as seguintes diretrizes são seguidas em ArchMine:

1) Nomes de classes dos elementos arquiteturais devem ser quebrados em *substrings*, através da identificação de caracteres especiais, como letra maiúscula e sinal de sublinhado (*underscore*).
2) A cada elemento arquitetural é associado um vetor de *substrings*, sendo estabelecido um peso a cada *substring* em função da sua freqüência de ocorrência nos nomes das classes que compõem o elemento arquitetural.
3) O nome do elemento arquitetural é derivado, concatenando-se as *substrings* com maior peso.
4) É possível que nomes redundantes sejam gerados quando dois ou mais elementos arquiteturais apresentem classes com *substrings* em comum. Por isso, um seqüencial é adicionado aos nomes dos elementos arquiteturais, permitindo torná-los sempre únicos.

A fim de ilustrar o processo de derivação dos nomes de elementos arquiteturais, os elementos arquiteturais reconstruídos no exemplo escolar hipotético são retomados. A Tabela 4.7 apresenta os nomes derivados aplicando-se as diretrizes sugeridas.

A motivação para essas diretrizes se origina do fato de ter sido observado ao longo dos estudos experimentais que, embora similaridade de nomes entre classes não seja um critério para o seu agrupamento, elementos arquiteturais tendem a possuir classes com nomes similares.

Tabela 4.7: Representação do processo de derivação de nomes de elementos arquiteturais.

Elemento Arquitetural	Substrings com Pesos	Nome do Elemento Arquitetural
EA1 = {Estudante, EstudanteGraduação, EstudantePosGraduação, GradeNotas}	Estudante 3, Graduação 2, Pos 1, Grade 1, Notas 1	Estudante + seqüencial 1 = Estudante1
EA2 = {Classes, Turmas, Inscrições}	Classes 1, Turmas 1, Inscrições 1	ClassesTurmasInscrições + seqüencial 2 = ClassesTurmasInscrições2
EA3 = {ImpressoraUtils, ImpressoraConfig, RelatUtils}	Impressora 2, Utils 2, Config 1, Relat 1	ImpressoraUtils + seqüencial 3 = ImpressoraUtils3

É possível que nomes de elementos arquiteturais sem semântica também sejam gerados, principalmente quando não há *substrings* em comum que prevaleçam nos nomes das classes e muitas *substrings* são concatenadas para a derivação do nome do elemento arquitetural. Nesse caso, nomes de elementos arquiteturais têm que ser alterados manualmente durante a avaliação da arquitetura recuperada.

Quanto à derivação dos relacionamentos entre elementos arquiteturais, que são representados através de pacotes da UML na abordagem proposta, os relacionamentos entre classes de diferentes elementos arquiteturais são analisados para derivar as dependências entre os pacotes. Os relacionamentos entre classes foram extraídos do código através da primeira atividade do processo, i.e. **Extração da Estrutura Estática**.

Dessa forma, as seguintes diretrizes são seguidas para a derivação dos relacionamentos entre elementos arquiteturais:

1) A cada associação entre 2 classes de elementos arquiteturais distintos, uma dependência do pacote que contém a classe origem da associação é gerada para o pacote que contém a classe destino da associação. No caso de associações bidirecionais, duas dependências, uma em cada direção, são derivadas entre os pacotes que representam os elementos arquiteturais.
2) A cada herança entre classes de diferentes elementos arquiteturais, uma dependência é gerada do pacote que contém a subclasse para o pacote que contém a superclasse.
3) A cada dependência entre classes ou entre classe e interface de elementos arquiteturais distintos, uma dependência é gerada do pacote que contém a origem da dependência para aquele que contém o destino.
4) A cada realização ou implementação entre uma classe que se encontra em um elemento arquitetural e uma interface que se encontra em outro elemento arquitetural, uma dependência é gerada do pacote que contém a origem da realização para aquele que contém o destino.
5) Dependências entre pacotes ou elementos arquiteturais só são geradas uma vez em cada sentido.

As diretrizes apresentadas seguem a própria sintaxe estabelecida pelo modelo de classes e de pacotes da UML. Embora a UML estabeleça alguns estereótipos pré-definidos para a determinação do tipo de dependência entre 2 pacotes, esses não são adotados nesta tese.

Em ArchMine, o modelo de pacotes gerado através da atividade de **Reconstrução dos Elementos Arquiteturais** representa a visão lógica da arquitetura, de acordo com o modelo de visões 4+1 (KRUCHTEN, 1995) - Figura 4.2.

4.2.5.3 – Reconstrução de Modelos Dinâmicos

O objetivo da **Reconstrução de Modelos Dinâmicos** é reconstruir os modelos de interação entre objetos e entre elementos arquiteturais, a fim de demonstrar o comportamento do sistema para a realização dos requisitos funcionais. O modelo adotado para representar as trocas de mensagens são os diagramas de seqüência da UML, que são associados aos cenários de caso de uso. Os diagramas de seqüência são reconstruídos com base na leitura dos rastros de execução coletados para os cenários de casos de uso. Através dos rastros de execução, é possível estabelecer rastros dos elementos arquiteturais e classes para os requisitos funcionais e vice-versa.

Como a análise dinâmica apresenta o problema de explosão do volume de informação nos rastros de execução, algumas técnicas são aplicadas por ArchMine para reduzir esse volume de informação, viabilizando a reconstrução dos diagramas de seqüência. As técnicas aplicadas são: **filtragem**; **partição dos rastros de execução**; e **ocultação de informação**.

A filtragem envolve a seleção da *thread* a partir da qual as chamadas de métodos devem ser lidas para a geração do diagrama de seqüência. Caso não seja selecionada uma *thread*, o diagrama apresentará o comportamento concorrente entre as *threads*. Além da *thread*, a chamada de método a partir da qual iniciar a reconstrução do diagrama de seqüência também é permitida.

A partição dos rastros de execução é feita por nível de profundidade de chamada de método na hierarquia. Ou seja, os diagramas de seqüência podem ser reconstruídos até o nível de profundidade 2, 3 etc. A fim de ilustrar as técnicas de filtragem e partição de rastros utilizadas, considere a seqüência de chamadas de método hipotética ilustrada na Figura 4.6. Nessa Figura, cada nó representa um método e cada seta representa uma chamada de método.

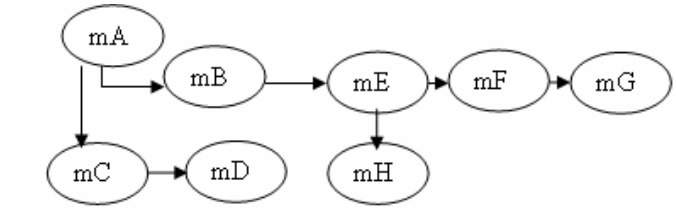

Figura 4.6: Seqüência de chamadas de método hipotética.

O diagrama da Figura 4.7 representa um diagrama de seqüência reconstruído para chamadas de métodos até o nível de profundidade 3. O diagrama de seqüência da Figura 4.8 detalha o método ME(). Dessa forma, o rastro de execução simplificado da

Figura 4.6 é particionado em 2 diagramas, sendo que no segundo diagrama é aplicada a técnica de filtragem por método inicial.

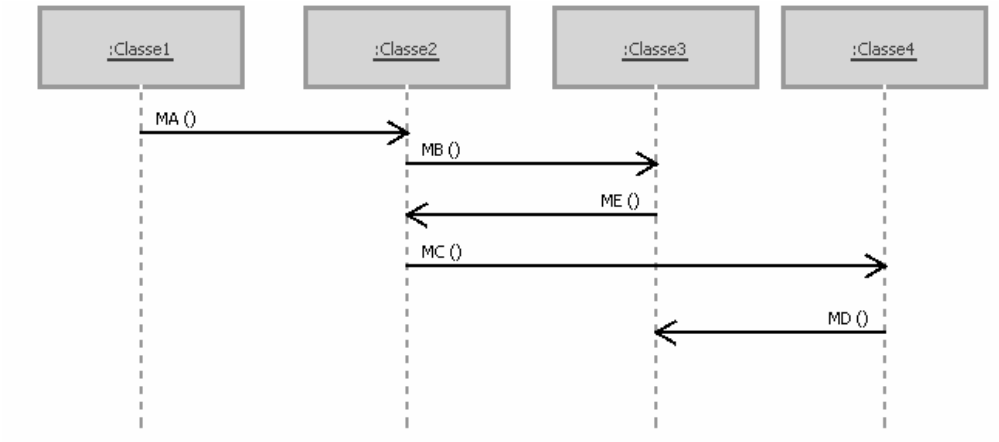

Figura 4.7: Diagrama de seqüência por nível de profundidade.

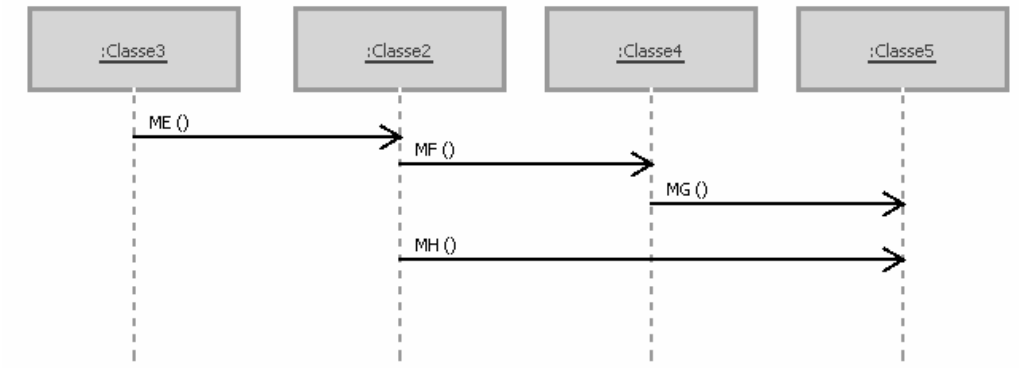

Figura 4.8: Diagrama de seqüência com filtro de método inicial.

Além dessas 2 técnicas, é possível ainda reconstruir diagramas de seqüência por nível de abstração, agrupando classes de um mesmo elemento arquitetural em uma única representação desse elemento. Mensagens trocadas entre objetos dessas classes são ocultadas, gerando um diagrama de seqüência em nível de abstração arquitetural.

Suponha que: as classes Classe2 e Classe3 do diagrama da Figura 4.7 componham um mesmo elemento arquitetural EA2; a classe Classe1 componha o elemento EA1; e a classe Classe4 componha o elemento EA3. O diagrama da Figura 4.9 retrata a troca de mensagens apresentada no diagrama da Figura 4.7, em nível de abstração arquitetural, ocultando as mensagens trocadas entre as classes Classe2 e Classe3 que compõem o mesmo elemento arquitetural. Dessa forma, é possível obter um diagrama em alto nível de abstração e compreender como o sistema se comporta em sua arquitetura para a realização dos requisitos funcionais, uma vez que cada cenário de

caso de uso pode estar associado a um ou mais diagramas de seqüência em nível de abstração arquitetural.

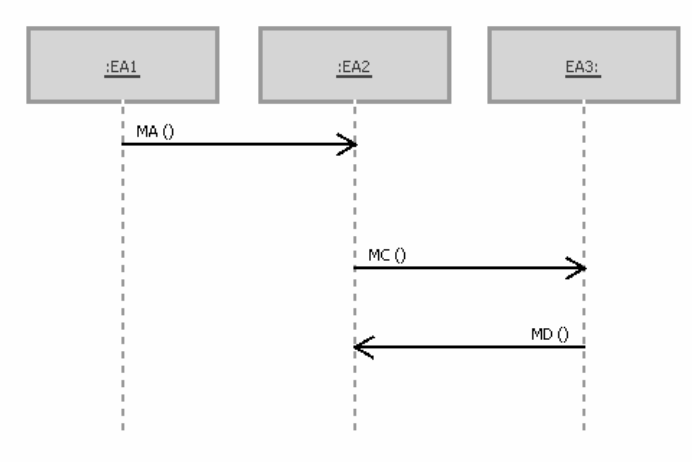

Figura 4.9: Diagrama de seqüência em nível de abstração arquitetural.

Os diagramas de seqüência que mostram trocas de mensagens entre instâncias de classes são reconstruídos em nível de abstração de classe por ArchMine, conforme apresentado nos diagramas da Figura 4.7 e da Figura 4.8. Essa estratégia também é utilizada para simplificar os rastros de execução. Se por um lado se tem uma perda de informação quando, por exemplo, duas instâncias de uma mesma classe trocam mensagens entre si, por outro, tem-se ganho de clareza nos diagramas que têm seu tamanho e complexidade bastante reduzidos.

Os diagramas de seqüência representam a visão de processos da arquitetura segundo o modelo 4+1. Eles demonstram comportamento concorrente entre múltiplas *threads* de execução do software. A associação dos diagramas a cenários de casos de uso representa a visão de cenários do modelo 4+1 (Figura 4.2).

4.2.6 – Avaliação da Arquitetura Recuperada

Visto que o objetivo principal com a recuperação de arquiteturas é que essas possam ser reutilizadas na especificação de uma arquitetura de referência de domínio, a avaliação da qualidade das arquiteturas recuperadas se torna importante, a fim de evitar que defeitos sejam propagados a diferentes aplicações do domínio. As abordagens de Engenharia Reversa, em geral, avaliam os modelos recuperados de forma *ad-hoc*, consultando especialistas do domínio para apoiar a avaliação, porém, sem uma abordagem que possa direcioná-la.

Inicialmente, a avaliação de arquiteturas recuperadas com ArchMine também era realizada de forma *ad-hoc*, entretanto, como se percebeu que essa forma de avaliação exigia grande esforço dos especialistas e ainda implicava em muita subjetividade, foi incorporada à ArchMine a abordagem de avaliação de arquiteturas ArqCheck (BARCELOS, 2006). ArqCheck foi estendida nesta tese para a avaliação de discrepâncias nos elementos arquiteturais que possam comprometer a sua Reusabilidade (VASCONCELOS e WERNER, 2007b).

ArqCheck, conforme descrito na seção 2.2.3, representa uma abordagem de inspeção de documentos arquiteturais, que utiliza como técnica de detecção de defeitos um *checklist*. A escolha de ArqCheck como abordagem de avaliação de arquiteturas nesta tese se deu em função das seguintes características da abordagem: é independente da representação arquitetural utilizada, podendo ser adaptada para a representação arquitetural recuperada com ArchMine; é extensível, permitindo a criação de novas questões no *checklist*; exige um esforço menor do que outras abordagens de avaliação arquitetural baseadas em cenário (ex: SAAM (KAZMAN *et al.*, 1994)); foi desenvolvida no mesmo ambiente acadêmico desta tese, onde o material e o pesquisador se encontravam disponíveis, facilitando a sua extensão e adaptação; e já foi avaliada através de dois estudos de caso, que forneceram indícios da sua viabilidade em detectar defeitos em documentos arquiteturais.

Além disso, as arquiteturas recuperadas no contexto desta tese atendem a boa parte dos requisitos estabelecidos por ArqCheck como essenciais para que uma arquitetura possa ser avaliada, como: a identificação dos elementos arquiteturais que compõem a solução e a sua organização, a representação do software através de diferentes perspectivas e o rastreamento de requisitos funcionais para elementos arquiteturais. ArqCheck requer ainda que o papel de cada elemento seja descrito na arquitetura. Em ArchMine, esse papel pode ser inferido através do nome atribuído ao elemento arquitetural e do seu rastreamento para requisitos funcionais. Quanto ao rastreamento para requisitos não-funcionais (ou atributos de qualidade), esse requisito de ArqCheck não é cumprido por ArchMine, sendo possível apenas avaliar o atendimento ao requisito de qualidade Reusabilidade.

A Figura 4.10 apresenta o processo de inspeção de ArqCheck, adaptado de (BARCELOS, 2006). No **Planejamento** é identificado o moderador para a inspeção, que seleciona os inspetores. O desenvolvedor que recupera a arquitetura deve ser o moderador da inspeção e a avaliação deve ser conduzida por desenvolvedores que

tenham conhecimento do domínio, atuando no papel de inspetores. Nessa atividade, também ocorre a configuração do *checklist* à documentação arquitetural que será avaliada, no caso, à documentação arquitetural recuperada com ArchMine. Essa configuração envolve classificar os itens do *checklist* em aplicáveis ou não para a documentação em questão, selecionar itens referentes aos atributos de qualidade de interesse e descrever os itens de qualidade como cenários de qualidade.

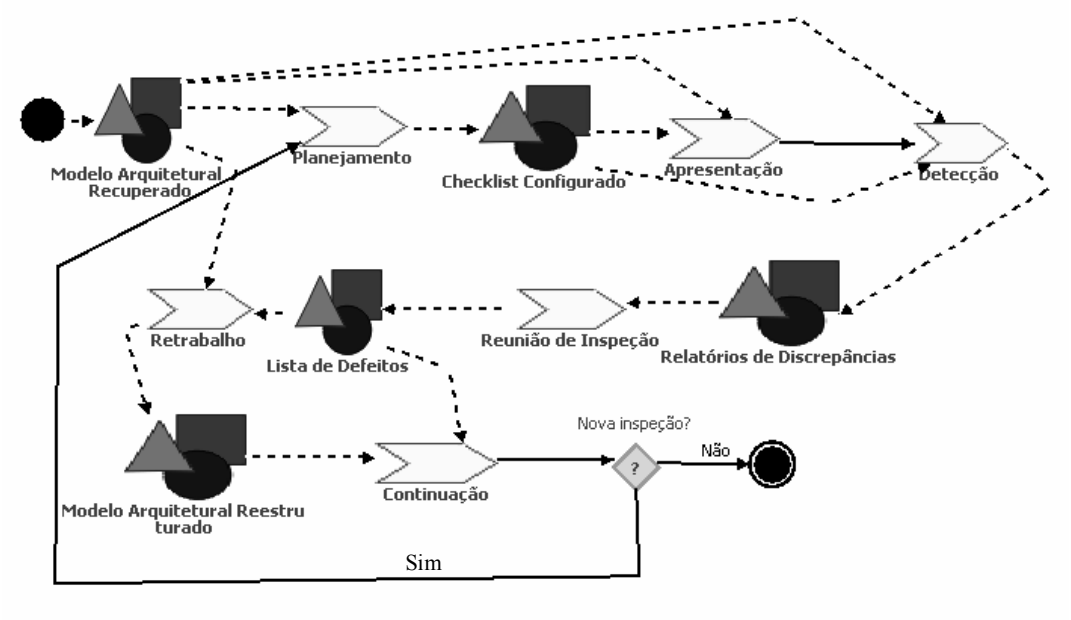

Figura 4.10: Processo de inspeção de ArqCheck. Adaptado de BARCELOS (2006).

ArqCheck considera os atributos de qualidade que, segundo BASS *et al.* (2003), são relevantes a nível arquitetural, i.e. Desempenho, Disponibilidade, Modificabilidade, Usabilidade, Segurança e Testabilidade. Visto que nesta tese o objetivo principal é recuperar elementos arquiteturais que possam ser reutilizados na criação de uma arquitetura de referência de domínio, extensões foram realizadas no *checklist* de ArqCheck para a avaliação desses elementos quanto à sua Reusabilidade.

As questões originais do *checklist* de ArqCheck para a avaliação do atendimento a atributos de qualidade estão embasadas no conhecimento existente nas táticas arquiteturais, extraídas principalmente de (BASS *et al.*, 2003). Uma vez que não foram encontradas táticas arquiteturais para o atendimento a Reusabilidade, as extensões do *checklist* para a avaliação de Reusabilidade se embasaram principalmente em princípios encontrados na literatura do paradigma de DBC (SAMETINGER, 1997; VITHARANA *et al.*, 2004; BLOIS, 2006), o qual enfatiza o desenvolvimento de elementos de software

reutilizáveis. Esses princípios foram tratados de forma genérica na elaboração das questões, podendo ser aplicados a uma arquitetura OO.

As questões criadas no *checklist* de ArqCheck para a avaliação de Reusabilidade e as razões (*rationale*) que embasam essas extensões são apresentadas no Anexo C/C.3. Convém ressaltar, entretanto, que princípios para se atingir Reusabilidade são semelhantes àqueles seguidos para se atingir Modificabilidade, requisito a partir do qual algumas questões do *checklist* original puderam ser adaptadas nas extensões realizadas.

O *checklist* apresentado no Anexo C/C.3 representa a instanciação do *checklist* para a avaliação da arquitetura de uma aplicação específica, o que impacta o requisito e o cenário de qualidade de Reusabilidade. O requisito descreve a instanciação do atributo de qualidade para a aplicação em questão, pois cada aplicação apresenta os seus próprios requisitos referentes a cada atributo de qualidade. Um cenário de qualidade, por sua vez, consiste em um conjunto de informações que permitem caracterizar um atributo de qualidade, facilitando o seu entendimento (BASS *et al.*, 2003). Os cenários de qualidade visam descrever os atributos de qualidade (ou requisitos não-funcionais) assim como os cenários de casos de uso descrevem os requisitos funcionais. O *checklist* apresenta ainda as "Guias de Identificação de Contexto" (Anexo C/C.3), que utilizam a descrição do atributo de qualidade e o cenário de qualidade para apoiar o inspetor na identificação dos elementos arquiteturais relacionados ao atendimento do atributo de qualidade. Essas guias são genéricas, não necessitando ser instanciadas quando o *checklist* é utilizado. O requisito e o cenário de qualidade são instanciados durante a atividade de **Planejamento** da inspeção.

Durante a **Apresentação** (Figura 4.10), o moderador deve realizar um treinamento com os inspetores para a utilização do *checklist*. Na atividade de **Detecção**, os inspetores selecionados inspecionam individualmente o documento arquitetural recuperado com ArchMine, utilizando o *checklist* configurado, e registram as discrepâncias identificadas no relatório de discrepâncias (Anexo C/C.4). Essas discrepâncias ainda devem ser classificadas em defeitos ou falso-positivos durante a **Reunião de Inspeção**. Participam da **Reunião de Inspeção**, os inspetores, o moderador e o autor do documento. Nesse caso, o autor do documento é o próprio moderador, i.e. o desenvolvedor que recupera a arquitetura.

Após a **Reunião de Inspeção** ocorre o **Retrabalho**, atividade na qual o desenvolvedor que recuperou a arquitetura corrige os seus defeitos, reestruturando-a, quando necessário. Nessa atividade, o desenvolvedor pode necessitar da ajuda dos

inspetores, contactando-os sempre que necessário, pois o desenvolvedor que recupera a arquitetura não necessita ter conhecimento do domínio. Na atividade de **Continuação**, o desenvolvedor que recuperou a arquitetura, atuando no papel de moderador e autor do documento, decide sobre a realização de uma nova inspeção, tomando como base os defeitos que foram corrigidos e a arquitetura reestruturada.

A fim de facilitar a compreensão das contribuições obtidas nesta tese em relação à ArqCheck, o Anexo G apresenta uma versão simplificada do *checklist* original de ArqCheck adaptado à documentação arquitetural de um dos sistemas utilizado nos estudos de viabilidade da abordagem (BARCELOS, 2006). Essa versão se encontra simplificada, pois nem todos os requisitos de qualidade e seus Cenários estão detalhados, bem como algumas questões do *checklist* estão omitidas. O Anexo D/D.4 apresenta um exemplo de instanciação do *checklist* estendido de ArqCheck para a avaliação de arquiteturas recuperadas com ArchMine. Comparando a versão do *checklist* do Anexo G com a versão apresentada no anexo D/D.4, verifica-se que o *checklist* foi estendido, nesta tese, através das questões referentes ao atendimento do atributo de qualidade Reusabilidade, o que representa uma contribuição em relação à ArqCheck.

Verifica-se também, comparando as versões dos *checklists* nos anexos G e D/D.4, que as questões não aplicáveis à documentação arquitetural recuperada com ArchMine não são consideradas. Finalmente, os termos referentes à representação arquitetural utilizados no *checklist* do Anexo G são substituídos pelos termos referentes à representação arquitetural recuperada com ArchMine, onde *cluster* equivale a elemento arquitetural e visão modular equivale a visão estrutural. Além disso, as visões de alocação e de contexto geral não são recuperadas com ArchMine. Dessa forma, uma outra contribuição nesta tese se refere à instanciação e configuração do *checklist* em um contexto diferente dos quais ele havia sido utilizado em (BARCELOS, 2006).

As questões criadas, nesta tese, para avaliar o atendimento do atributo de qualidade Reusabilidade representam um possível caminho para a detecção de inconsistências sob esse ponto de vista em uma arquitetura. Outros trabalhos consideram outros pontos de vista (GANESAN e KNODEL, 2005; KNODEL e MUTHIG, 2005).

4.2.7 – Considerações Finais sobre ArchMine

De acordo com a definição de arquitetura adotada nesta tese, i.e. a de BASS *et*

al. (2003), os seguintes pontos são atendidos por ArchMine: recuperação de elementos que compõem o software, recuperação dos relacionamentos entre eles e recuperação de diferentes perspectivas arquiteturais. Quanto à abstração (i.e. propriedades externamente visíveis), ArchMine recupera o nome que representa a semântica do elemento e os requisitos funcionais que ele satisfaz. Seria interessante a derivação também das interfaces do elemento, i.e. serviços providos e requeridos, o que representa um tópico de trabalho futuro.

Além de apoiar a reutilização, ArchMine pode ser utilizada para apoiar atividades de manutenção. Diagramas de seqüência para cenários de casos de uso que devem sofrer manutenção podem ser recuperados, a fim de facilitar a compreensão da implementação dessas funcionalidades no sistema.

A Tabela 4.8 apresenta um comparativo entre as abordagens de recuperação de arquitetura apresentadas no capítulo 3 e ArchMine, em função dos requisitos estabelecidos nesta tese. Dois dos requisitos estabelecidos, i.e. apoio ferramental e integração a um ambiente de reutilização, são avaliados e acrescentados na tabela no capítulo 5, que apresenta o ambiente e ferramental de apoio à execução do processo proposto em ArchMine. Convém ressaltar que diferentes abordagens de recuperação de arquitetura apresentam diferentes objetivos e, conseqüentemente, requisitos para a recuperação. Entretanto, nesta tese, procurou-se estabelecer requisitos focados nos objetivos de apoio à reutilização e manutenção, sendo as abordagens existentes avaliadas sob esses pontos de vista.

Dois requisitos são apontados como sendo atendidos parcialmente por ArchMine, i.e. geração de nomes de elementos arquiteturais com semântica de forma automática e generalidade da abordagem. Embora ArchMine busque generalidade em relação a domínio e linguagem de programação, algum conhecimento do sistema é necessário para a definição dos cenários de casos de uso e, posteriormente, para a avaliação da arquitetura recuperada. Além disso, ArchMine é voltada à análise de sistemas OO. Entretanto, diferente de várias abordagens de recuperação de arquitetura pesquisadas (ex: (HARRIS *et al.*, 1997b; KAZMAN e CARRIÈRE, 1997; RIVA e RODRIGUEZ, 2002; SCHMERL *et al.*, 2006)), ArchMine apresenta critérios genéricos para a reconstrução dos elementos arquiteturais, os quais podem ser reutilizados para diferentes domínios e aplicações.

Tabela 4.8: Quadro comparativo entre as abordagens de recuperação de arquitetura e ArchMine.

Abordagens: Recuperação de Arquitetura	**Elemento Arquitetural = Conceito do Domínio**	**Nomes de Elementos Arquiteturais com Semântica e Gerados Automaticamente**	**Rastro para Requi-sitos Funci-onais**	**Genera lidade**	**Abordagem de Avaliação**	**Separa-ção de Aspectos Estáticos e Dinâmi-cos**
ManSART (HARRIS *et al.*, 1997a)	**Parcial.** Pode representar quando o componente, ex: executável, reflete o conceito.	**Não.**	**Não.**	**Não.**	**Não**	**Não.**
X-Ray (MENDONÇA e KRAMER, 2001)	**Parcial.** Pode representar quando o processo reflete o conceito.	**Não.**	**Não.**	**Não**	**Não.**	**Não.**
Pattern-supported (GALL e PINZGER, 2002)	**Parcial.** Pode representar quando o componente reflete o conceito.	**Não.**	**Não.**	**Não.**	**Não**	**Não.**
DiscoTect (SCHMERL *et al.*, 2006)	**Parcial.** Pode representar quando o componente reflete o conceito.	**Não.**	**Não.**	**Parcial**	**Sim.** Ferramenta compara com arquitetura original e descobre desvios	**Não**.
Dali (KAZMAN e CARRIÈRE, 1997)	**Sim.**	**Parcial.** Elementos representam conceitos do domínio possuindo nomes correspondentes, mas atribuídos manualmente.	**Não.**	**Parcial**	**Sim.** Ferramenta compara com arquitetura original e descobre desvios	**Não**
High-Level Views for OO Applications (RICHNER e DUCASSE, 1999)	**Sim.**	**Parcial.** Nomes atribuídos manualmente.	**Não.**	**Parcial**	**Não**	**Sim.**
Sincronização entre Visões Estática e Dinâmica (RIVA e RODRIGUEZ, 2002)	**Sim.**	**Parcial.** Nomes atribuídos manualmente.	**Sim.**	**Parcial**	**Não**	**Sim.**
Alborz (SARTIPI e KONTOGIANNIS, 2003)	**Sim.**	**Parcial.** Nomes atribuídos manualmente.	**Não.**	**Parcial**	**Não.**	**Não.**
Abordagem direcionado por Casos de Uso (BOJIC e VELASEVIC, 2000)	**Sim.**	**Parcial.** Nomes derivados dos casos de uso.	**Sim.**	**Parcial**	**Não**	**Não.**
Focus (DING e MEDVIDOVIC, 2001)	**Parcial.**	**Parcial.** Nomes atribuídos manualmente.	**Sim.**	**Sim**	**Não**	**Sim.**
File Names (ANQUETIL e LETHBRIDGE, 1999)	**Parcial.** Depende dos conceitos contidos nos nomes de arquivos.	**Parcial.** Heurística para detectar subsistemas considerando a primeira *substring* de nomes de arquivos que não seja uma única letra.	**Não.**	**Sim**	**Sim.** Comparação com decomposição fornecida por especialistas.	**Não.**
Bunch (MITCHELL e MANCORIDIS, 2006)	**Parcial.** Somente se classes acopladas refletem um conceito.	**Não.**	**Não.**	**Sim**	**Não**	**Não.**
ArchMine	**Sim.**	**Parcial.**	**Sim.**	**Parcial**	**Sim**	**Sim.**

Em relação à geração de nomes de elementos arquiteturais, ArchMine apresenta um conjunto de diretrizes para a geração de nomes com semântica e de forma automática. Entretanto, em alguns casos, esses nomes precisam ser corrigidos por um

stakeholder com conhecimento do domínio. Um exemplo é quando os nomes gerados se tornam extensos por não haver *substrings* comuns nos nomes das classes que caracterizem o elemento. Dessa forma, o nome do elemento arquitetural acaba sendo formado pela concatenação de um grande número de *substrings*.

O requisito referente à abordagem de avaliação, visa verificar se a abordagem de recuperação de arquitetura propõe uma abordagem de avaliação da arquitetura recuperada ou se essa avaliação é realizada de forma *ad-hoc* por especialistas das aplicações. Observa-se que a maioria das abordagens se baseia em avaliações *ad-hoc*, conduzidas sem um direcionamento, o que pode não garantir que os objetivos com a recuperação da arquitetura estejam sendo atingidos.

Dessa forma, pela Tabela 4.8, verifica-se que o desenvolvimento de ArchMine foi motivado por não haver uma abordagem de recuperação de arquitetura, dentre as pesquisadas, que atendesse plenamente ou minimamente a todos os requisitos estabelecidos nesta tese.

A abordagem ArchMine foi sendo refinada ao longo dos estudos experimentais realizados, apresentados no capítulo 6. Dessa forma, as diretrizes e heurísticas apresentadas como guias para as atividades do processo são baseadas em conceitos presentes no referencial teórico de arquitetura de software, reutilização de software, mineração de dados e engenharia reversa, além de observações durante os estudos experimentais e *feedback* recebido dos especialistas dos sistemas para os quais a arquitetura foi recuperada.

Algumas limitações se apresentam à abordagem proposta, como o impacto na qualidade da arquitetura recuperada resultante dos cenários de casos de uso definidos, antecedentes selecionados e valores de confiança e suporte mínimos adotados para a mineração. Porém, outras abordagens de recuperação de arquitetura, como a apresentada em (MITCHELL e MANCORIDIS, 2006), também apresentam aleatoriedade nos resultados, havendo diferenças na arquitetura recuperada a cada execução do algoritmo. Esse não é um problema incomum em algoritmos de agrupamento (*clustering*), os quais costumam ser a base para a recuperação de arquitetura. Tanto que em (TZERPOS, 2001), é proposta uma métrica para avaliar a estabilidade de algoritmos de agrupamento. O autor argumenta que o algoritmo deve atender ao princípio de "modificação mínima", ou seja, não deve haver mudanças radicais na arquitetura a cada execução do algoritmo. Nesse sentido, seria interessante realizar testes com ArchMine variando esses valores de entrada da abordagem para

verificar o grau de variação nos resultados obtidos, possivelmente aplicando alguma métrica como a sugerida em (TZERPOS, 2001). Entretanto, apesar desse impacto, os estudos experimentais realizados até o momento oferecem indícios de que as heurísticas e diretrizes de ArchMine produzem bons resultados.

Convém ressaltar que à medida que novos estudos sejam realizados com ArchMine, as diretrizes e heurísticas definidas devem continuar sendo refinadas em um processo iterativo e contínuo de melhoria da abordagem. Essa idéia é reforçada pelos resultados obtidos para ArchMine em seu estágio atual, que estimulam a continuidade de investimento em pesquisa no seu aprimoramento.

Finalmente, a recuperação da arquitetura em ArchMine é direcionada por requisitos funcionais. O único requisito não-funcional considerado, entretanto, é a Reusabilidade, onde o agrupamento de classes que implementam os mesmos requisitos funcionais tende a gerar elementos arquiteturais que representam grupos de serviços coesos, capazes de representar conceitos do domínio.

4.3 – ArchToDSSA: Comparação de Arquiteturas e Criação de Arquiteturas de Referência

A abordagem ArchToDSSA foi desenvolvida em um trabalho de mestrado em fase final de conclusão (KÜMMEL, 2007), que faz parte do contexto da solução proposta nesta tese. O detalhamento do problema e de caminhos para a solução apresentados por ArchToDSSA, foram realizados em sintonia com o trabalho desenvolvido nesta tese.

ArchToDSSA segue a notação Odyssey-FEX (OLIVEIRA, 2006b) para a representação de opcionalidades e variabilidades. O processo de ArchToDSSA é realizado através das atividades apresentadas na Figura 4.11. O processo se inicia pela **Detecção de Opcionalidades**, atividade que utiliza como artefato de entrada as arquiteturas de sistemas legados do domínio recuperadas com ArchMine. Essa atividade gera como saída elementos equivalentes encontrados nas diferentes arquiteturas e, com base nessas equivalências, uma marcação de elementos opcionais e mandatórios nas diferentes arquiteturas é realizada.

A próxima atividade é a **Detecção de Variabilidades,** que utiliza como artefatos de entrada as arquiteturas recuperadas no domínio e as equivalências identificadas na atividade anterior. Com base nessas informações, pontos de variação (VPs) e variantes

são identificados nas diferentes arquiteturas. Elementos que não são nem VPs e nem variantes, são considerados invariantes segundo a notação Odyssey-FEX. Opcionalidades e variabilidades são propriedades ortogonais na Odyssey-FEX e, dessa forma, um mesmo elemento na arquitetura pode ser ao mesmo tempo um VP, variante ou invariante e mandatório ou opcional. Assim, a última atividade de ArchToDSSA, i.e. a **Criação de DSSA**, utiliza as informações dos elementos mandatórios e opcionais, combinadas às informações de variabilidades, para reproduzir diferentes arquiteturas de referência a partir das arquiteturas recuperadas com ArchMine, gerando uma arquitetura por vez.

O papel responsável pela execução dessas atividades é o Engenheiro de Domínio ou o Engenheiro em uma abordagem de LP, responsável pela elaboração da arquitetura de referência de domínio (ou arquitetura da linha de produtos).

ArchToDSSA lê arquiteturas recuperadas em UML, as quais podem ser externalizadas através do formato XMI. Dessa forma, as arquiteturas de referência geradas também são representadas em formato XMI, o que proporciona uma certa independência de ambiente de desenvolvimento, como será visto no capítulo 5. Segue nas próximas sub-seções um detalhamento das atividades que compõem a abordagem ArchToDSSA.

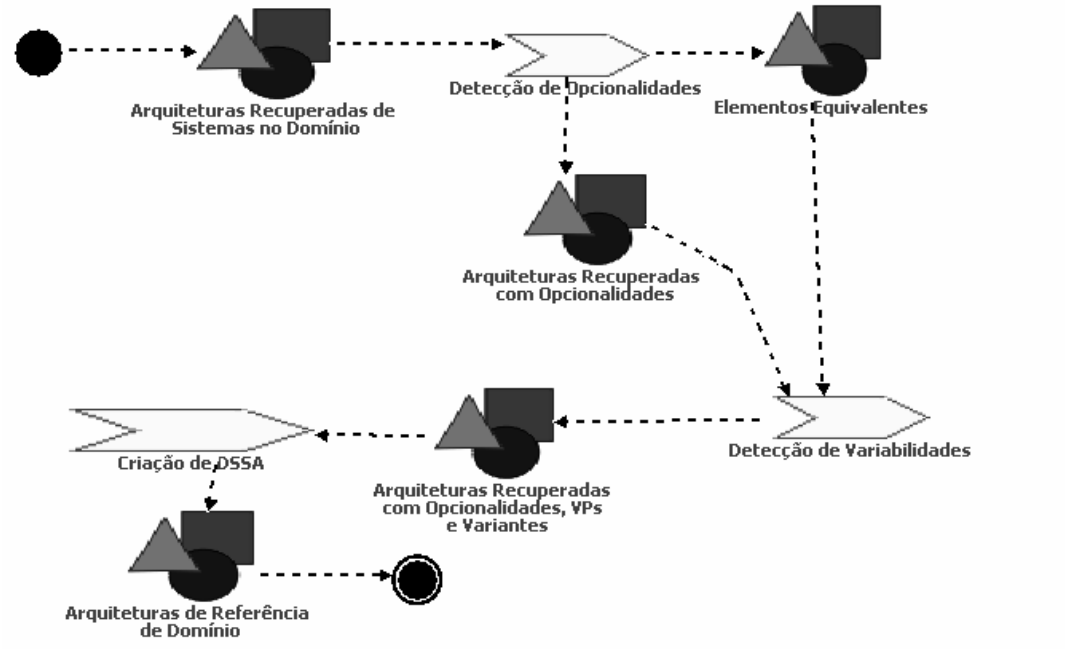

Figura 4.11: Processo da abordagem ArchToDSSA.

4.3.1 – Detecção de Opcionalidades

Nessa primeira atividade, o objetivo é identificar dentre as arquiteturas recuperadas, as similaridades e diferenças entre seus respectivos elementos arquiteturais. Dessa forma, é possível identificar elementos mandatórios e opcionais no domínio, ou seja, as opcionalidades do domínio em questão. Essas similaridades e diferenças são identificadas tanto em nível de elemento arquitetural quanto em nível de suas classes (estrutura interna).

Para a identificação de elementos mandatórios e opcionais entre as arquiteturas, o primeiro passo é a identificação de equivalências (*matches*). Equivalências representam elos de ligação entre elementos equivalentes entre diferentes arquiteturas. Elementos equivalentes podem apresentar o mesmo nome ou nomes diferentes nas arquiteturas comparadas em função delas se originarem de diferentes sistemas do domínio, conforme mostra a Figura 4.12. Nesse exemplo, com arquiteturas recuperadas para um domínio escolar, "Aluno" e "Estudante", por exemplo, representam o mesmo conceito, embora apresentando nomes diferentes. Da mesma forma que "Turmas" e "Turmas" representam uma equivalência.

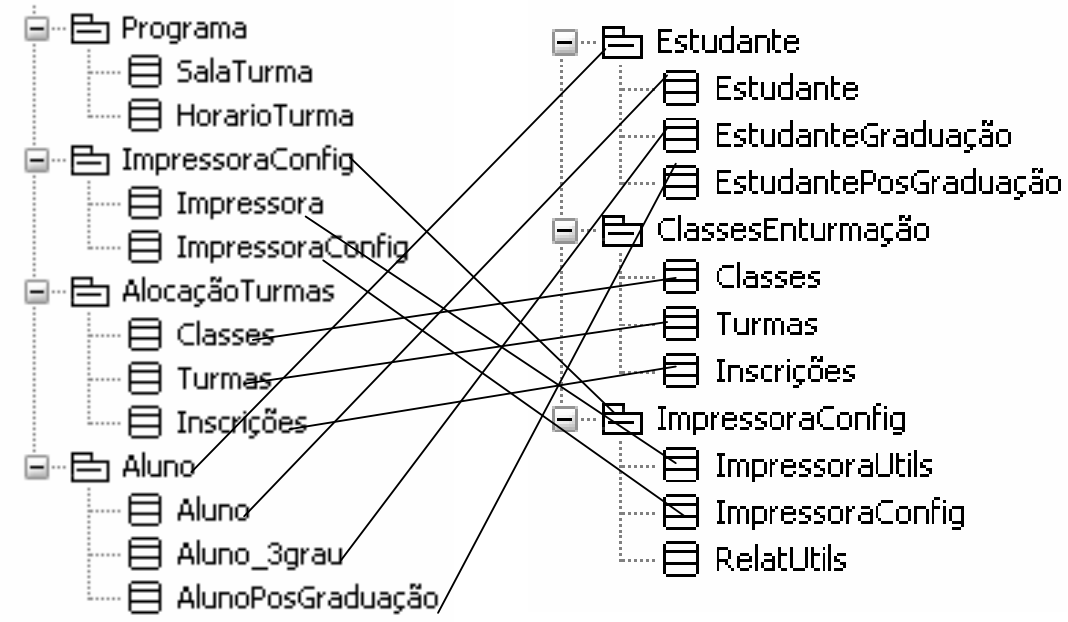

Figura 4.12: Equivalências entre diferentes arquiteturas de um domínio.

Essa equivalência semântica em ArchToDSSA é obtida através de um dicionário de sinônimos que é alimentado pelo Engenheiro de Domínio. A cada ligação estabelecida pelo Engenheiro de Domínio entre elementos de diferentes arquiteturas que

possuam nomes diferentes, uma entrada no dicionário é criada, definindo esses nomes como sinônimos. O Engenheiro de Domínio pode ainda definir diretamente sinônimos no dicionário sem necessariamente estar efetuando ligações entre arquiteturas. Dessa forma, a comparação das equivalências com novas arquiteturas do mesmo domínio é simplificada através do dicionário, o qual é constantemente alimentado ao longo do processo.

Além da comparação de nomes iguais ou sinônimos, a abordagem admite ainda 3 possibilidades de configuração para a determinação de equivalências de forma automática:

- **Lista de Palavras Ignoradas**: é possível que a abordagem ignore palavras sem semântica "forte", como Gerente, Utils, Controlador etc., na detecção de equivalências. Essas palavras não contribuem semanticamente com a comparação. Assim, uma lista de palavras ou abreviações que devem ser ignoradas é mantida pela abordagem.
- **Comparação por *Substrings***: é possível que os nomes sejam divididos em partes durante a comparação. Os critérios para a divisão em partes envolvem a detecção de uma letra maiúscula ou um sinal de sublinhado. Dessa forma, no nome “ImpressoraUtils” poderão existir duas partes, i.e. “Impressora” e “Utils”, e no nome “ClassesTurmasInscrições” três partes, i.e. “Classes”, “Turmas” e “Inscrições”. Após dividir os nomes em partes, é verificado para cada parte (em cada nome), se a referida parte se encontra na “lista de palavras a serem ignoradas”, caso se encontre, essa parte é retirada da lista de partes da palavra. Em seguida, cada parte do primeiro nome é comparada com cada parte do segundo nome. Se uma parte de um nome possui uma parte igual em outro nome, essas duas partes são marcadas e não podem mais ser envolvidas na comparação das outras partes dos nomes comparados. Assim, por exemplo, as *strings* BaBeBa e BaBaBe são consideradas iguais, uma vez que as *substrings* "Ba", "Be" e "Ba" são encontradas na segunda *string*. Entretanto, as *strings* BaBeBa e BaBeBe são diferentes uma vez que "Ba" e "Be" são encontradas na segunda string, mas "Ba" não é encontrada na segunda *string* em uma segunda ocorrência.
- **Comparação de Elementos do Mesmo Tipo**: indica se deve ser considerado o tipo do elemento na comparação, i.e. classes só devem ser comparadas com classes e pacotes só devem ser comparados com pacotes.

Em relação ao exemplo apresentado na Figura 4.12, observa-se que a comparação é realizada por nome, sinônimo, comparação de *substrings*, palavras ignoradas e tipo do elemento arquitetural. Por exemplo, "Aluno" pacote (i.e. elemento arquitetural) corresponde a "Estudante" pacote, ao passo que "Aluno" classe corresponde a "Estudante" também classe. "Aluno_3grau", classe (i.e. parte da estrutura interna de um elemento arquitetural), corresponde a "EstudanteGraduação", também classe. "Graduação" e "3grau" são sinônimos nesse caso, bem como "Aluno" e "Estudante". A comparação é realizada por *substrings*.

Considerando a lista de palavras ignoradas e a comparação por *substrings*, observa-se que a *substring* "Utils" é ignorada nessa comparação. Dessa forma, quebrando "ImpressoraUtils" na 2ª arquitetura em 2 *substrings*, i.e. "Impressora" e "Utils", e comparando esse nome com a 1ª arquitetura, obtém-se uma equivalência com "Impressora", uma vez que "Utils" é ignorada na comparação. Todas essas opções podem ser determinadas como aplicáveis ou não pelo Engenheiro de Domínio em ArchToDSSA.

Nessas comparações entre as arquiteturas, é sempre registrado o número de arquiteturas onde a equivalência é detectada. Quando elementos equivalentes são encontrados em todas as arquiteturas comparadas, eles são candidatos a elementos mandatórios no domínio, ao passo que elementos equivalentes encontrados em algumas das arquiteturas comparadas são candidatos a elementos opcionais do domínio.

4.3.2 – Detecção de Variabilidades

Na segunda fase de ArchToDSSA, as arquiteturas recuperadas e a lista de equivalências obtidas na primeira fase são analisadas para que se possa definir VPs e suas respectivas variantes no domínio, ou seja, as variabilidades do domínio em questão.

A abordagem faz uso de arquiteturas extraídas de sistemas legados OO, descritas através de diagramas de pacotes e de classes da UML. Com base nessas informações, é possível descobrir os relacionamentos entre as classes. Os relacionamentos de herança e implementação, especificamente, são de grande interesse para a descoberta de possíveis VPs e suas respectivas variantes. De acordo com as regras de mapeamento na Odyssey-FEX, partindo do modelo de características, um VP pode ser mapeado para uma superclasse ou interface no modelo de classes, e suas variantes podem ser mapeadas

para subclasses ou classes que implementam uma interface no modelo de classes[10].

Além disso, é possível que o Engenheiro de Domínio determine em quantas arquiteturas uma dada interface ou superclasse deve possuir equivalência para que ela possa ser considerada um VP. Dessa forma, ArchToDSSA é capaz de indicar candidatos a VPs e variantes, sendo que as variantes candidatas são todas as subclasses encontradas nas diferentes arquiteturas que estejam subordinadas ao VP ou todas as classes que implementam a interface que representa o VP.

Entretanto, essas regras de mapeamento da Odyssey-FEX, embora tendo sido derivadas com base em estudos experimentais, podem não ser válidas para todos os casos. Assim, é importante que o Engenheiro de Domínio possa determinar aleatoriamente VPs e variantes nas diferentes arquiteturas comparadas, bem como recusar sugestões de VPs e variantes detectados pela abordagem.

4.3.3 – Criação de DSSA

Finalmente, na última fase de ArchToDSSA, é selecionada uma das arquiteturas recuperadas como base para a criação da arquitetura de referência de domínio. A seleção de uma arquitetura é necessária para evitar a existência de relacionamentos inconsistentes na arquitetura de referência resultante, o que poderia ocorrer caso elementos de diferentes arquiteturas fossem combinados aleatoriamente para gerar a arquitetura de referência. Dessa forma, assegura-se que todos os elementos e relacionamentos dessa arquitetura "base" estarão incondicionalmente presentes na arquitetura de referência.

Convém ressaltar, entretanto, que diferentes arquiteturas de referência podem ser criadas com base em diferentes escolhas realizadas pelo Engenheiro de Domínio. A fim de facilitar essa escolha, ArchToDSSA apresenta em cada arquitetura, os elementos opcionais, mandatórios, VPs e variantes. Entretanto, essa escolha não é apoiada por ArchToDSSA, podendo ser levado em conta o número de defeitos identificados em cada arquitetura com ArqCheck, a experiência do Engenheiro do Domínio ou o cálculo de métricas sobre as arquiteturas recuperadas, como coesão e acoplamento, sendo essa

[10] As regras de mapeamento na Odyssey-FEX foram obtidas a partir da análise de modelos em três domínios diferentes. Especialistas nos respectivos domínios estabeleceram manualmente correspondências entre os modelos de características e de classes e, dessa forma, as regras foram sendo derivadas.

diretriz, entretanto, a menos aconselhada uma vez que arquiteturas com alto acoplamento podem ainda refletir bem os conceitos ou elementos arquiteturais do domínio e uma avaliação subjetiva pode ser mais adequada nesse caso.

As informações de opcionalidades e variabilidades são combinadas na criação da DSSA, uma vez que representam propriedades ortogonais na Odyssey-FEX, sendo então estabelecidos VPs mandatórios e opcionais, bem como variantes mandatórias e opcionais, além de invariantes mandatórias e opcionais. Elementos opcionais de outras arquiteturas, que não a arquitetura "base" selecionada, podem também compor a arquitetura de referência. Nesse caso, eles são marcados como "não definidos" na arquitetura de referência, marcação essa permitida pela notação Odyssey-FEX, o que indica que eles ainda precisarão ser "ligados" e analisados minuciosamente na arquitetura de referência. Esses elementos opcionais de outras arquiteturas são propositadamente incorporados de forma "solta" à arquitetura de referência, uma vez que não há como garantir que seus relacionamentos na sua arquitetura original se mantenham na arquitetura de referência, pois muitos dos elementos com os quais eles se relacionam direta ou indiretamente, podem não possuir equivalência na arquitetura "base". Ainda que possuam equivalência, não há como assegurar que as estruturas internas dos elementos sejam exatamente iguais nas diferentes arquiteturas para garantir que as ligações continuem fazendo sentido.

A exceção se encontra para o caso das variantes opcionais incorporadas de outras arquiteturas, que são ligadas na arquitetura de referência, através de herança ou implementação de interface, mas, ainda assim, marcadas como "não definidas", pois um exame minucioso da sua compatibilidade estrutural e semântica em relação ao VP ainda deve ser realizado pelo Engenheiro de Domínio. Convém ressaltar, ainda, que caso sejam selecionados elementos equivalentes opcionais de diferentes arquiteturas para a criação da arquitetura de referência, ArchToDSSA admite apenas uma ocorrência do elemento na arquitetura final. Se ele existir na arquitetura "base" escolhida, será então priorizada a sua ocorrência nessa arquitetura.

Os elementos selecionados de outras arquiteturas, que não a arquitetura "base", estarão hierarquicamente subordinados ao seu elemento superior mais direto com equivalência na arquitetura "base" (i.e. uma superclasse, um subpacote ou um pacote). Caso essa equivalência não seja encontrada em nenhum nível da hierarquia para o elemento incorporado de outra arquitetura, ele é alocado diretamente à raiz da arquitetura, ou seja, a um pacote "DSSAmodel" criado no XMI.

As arquiteturas de referência resultantes são representadas através de XMI. Uma vez que opcionalidades e variabilidades, bem como o conceito de "não definido", não fazem parte do metamodelo da UML, essas propriedades são representadas através de um mecanismo de extensão da UML, i.e. valores chaveados (*tagged values*). Valores chaveados permitem definir novos tipos de propriedades, adicionando tuplas de <nome, valor> a qualquer elemento de modelo. Dessa forma, cada elemento na arquitetura de referência resultante conterá 3 tuplas de <nome, valor>, a saber:

- Nome = "opcionalidade"; valor = "mandatório" ou "opcional";
- Nome = "variabilidade"; valor = "VP" ou "variante" ou "invariante";
- Nome = "não definido"; valor = "verdadeiro" ou "falso".

4.3.4 – Considerações Finais sobre ArchToDSSA

ArchToDSSA gera arquiteturas de referência de domínio parcialmente especificadas, permitindo a geração de uma arquitetura por vez, a partir da seleção de uma arquitetura base. Dessa forma, além de representar uma abordagem de **diff** entre modelos, ArchToDSSA identifica opcionalidades e variabilidades estruturais no domínio, seguindo a taxonomia proposta na Odyssey-FEX (OLIVEIRA, 2006b).

Essas opcionalidades e variabilidades são identificadas tanto em nível de elemento arquitetural quanto em nível de suas classes, i.e. estrutura interna. Entretanto, nesse momento, a opcionalidade das classes não impacta a opcionalidade do elemento arquitetural, o que pode ser visto como uma necessidade para trabalho futuro. O Engenheiro de Domínio obtém apenas a relação de equivalências e opcionalidades, podendo deduzir se existem conflitos entre essas opcionalidades, como, por exemplo, um percentual alto de classes opcionais em um elemento arquitetural mandatório.

Mesmo se baseando em arquivos XMI, ArchToDSSA não compara os identificadores dos elementos nos diferentes arquivos (i.e. o MOF ID), mas sua equivalência através de nomes e tipos. Assume-se que arquivos XMI representando arquiteturas de diferentes sistemas ou aplicações não possuem a correspondência entre identificadores que existe entre arquivos XMI representando diferentes versões de uma mesma arquitetura. Além disso, uma vez que a comparação é realizada em nível de arquivo XMI, ela se detém aos elementos semânticos do modelo. Questões sintáticas ou diagramáticas não são consideradas na comparação.

Como possibilidades de melhorias sobre ArchToDSSA, verificam-se algumas oportunidades interessantes, como:

1. **A indicação da consistência das opcionalidades entre diferentes níveis hierárquicos**: seria interessante desenvolver mecanismos de checagem em ArchToDSSA onde um percentual significativo de opcionalidades e variabilidades de um determinado tipo na estrutura interna de um elemento arquitetural impactasse a sua própria opcionalidade e variabilidade. O mesmo mecanismo poderia checar as equivalências, possivelmente indicando como equivalentes elementos arquiteturais com um grande número de equivalências em suas estruturas internas.
2. **Detecção de diff em um nível de granularidade mais "fino"**: a comparação dos modelos arquiteturais e estabelecimento de equivalências poderiam chegar ao nível de métodos e atributos, pois opcionalidades e variabilidades em um domínio podem se manifestar nesse nível "fino" de granularidade. Notações para a representação de variabilidade, como a apresentada em (MORISIO *et al.*, 2000) prevêem esse nível de granularidade.

Vale ressaltar que as heurísticas adotadas na abordagem ArchToDSSA se originaram da observação da estrutura de sistemas existentes em domínios diversificados e de fundamentos teóricos, apresentados na literatura (GOMAA, 2004; BLOIS, 2006; OLIVEIRA, 2006b).

4.4 – Considerações Finais

Este capítulo apresentou a abordagem proposta nesta tese para apoio à criação de arquiteturas de referência de domínio. O produto final da abordagem são arquiteturas de referência parcialmente especificadas para um domínio de aplicação.

A contribuição desta tese é a proposta da abordagem LegaToDSSA, integrando a recuperação de arquiteturas, i.e. ArchMine, e a comparação de arquiteturas para a detecção de semelhanças e diferenças, i.e. ArchToDSSA, no apoio à criação de arquiteturas de referência de domínio. Conforme descrito, LegaToDSSA estabelece atividades sistemáticas, realizadas com o apoio de técnicas, diretrizes e heurísticas. As abordagens de ED e LP existentes não costumam oferecer esse apoio. A abordagem proposta está dividida em 2 fases, porém, conforme observado ao longo do capítulo, as maiores contribuições e o maior esforço de trabalho desta tese se concentram na fase de recuperação de arquiteturas de sistemas legados OO, realizada através de ArchMine. Através da análise da Tabela 4.8, percebe-se que ArchMine atende quase totalmente aos

requisitos estabelecidos nesta tese, oferecendo contribuições na área de ER em relação às características encontradas em outras abordagens de recuperação de arquitetura.

Uma etapa importante em abordagens de ED e LP representa o Planejamento do Domínio, que, no caso da análise de sistemas legados, deve envolver a seleção dos sistemas candidatos no domínio. Essa etapa não é coberta nesta tese, embora alguns requisitos mínimos se apresentem para que os sistemas possam ser analisados por ArchMine, como: ser desenvolvidos segundo o paradigma OO; apresentar o código fonte e uma versão passível de ser executada disponíveis; e haver disponibilidade de pelo menos um *stakeholder* que possua conhecimento do sistema para apoiar a definição dos cenários de casos de uso e a avaliação das arquiteturas recuperadas. Outras diretrizes para o Planejamento de Domínio podem ser obtidas em métodos de ED, como o método proposto por BRAGA (2000).

Capítulo 5 – Ferramental de Apoio à Abordagem Proposta

5.1 – Introdução

O capítulo 4 apresentou a abordagem de apoio à criação de arquiteturas de referência de domínio, LegaToDSSA, proposta nesta tese. Conforme descrito, a abordagem é composta de 2 etapas, executadas através das abordagens **ArchMine,** recuperação de arquitetura, e **ArchToDSSA,** comparação de arquiteturas e geração de arquiteturas de referência de domínio parcialmente especificadas. Este capítulo apresenta as ferramentas construídas para apoiar a execução das atividades de **ArchMine** e **ArchToDSSA**.

Em relação às abordagens de recuperação de arquitetura, 2 requisitos estabelecidos no capítulo anterior são avaliados neste capítulo, a saber: apoio ferramental, a fim de reduzir o esforço de execução das atividades do processo, e integração a um ambiente de reutilização, para que os modelos recuperados possam ser reutilizados em novos desenvolvimentos, principalmente, em processos de Engenharia de Domínio (ED) e Linha de Produtos (LP).

A fim de atender ao segundo requisito citado, as ferramentas desenvolvidos nesta tese apresentam a possibilidade de integração com o ambiente de apoio à reutilização baseado em modelos de domínio, Odyssey (ODYSSEY, 2007). O Odyssey foi selecionado por oferecer recursos que apóiam a abordagem proposta, além de estar sendo desenvolvido no mesmo contexto de pesquisa desta tese, i.e. pelo grupo de reutilização de software da COPPE/UFRJ.

A título de ilustração do apoio oferecido pelas ferramentas desenvolvidas, um exemplo de recuperação de arquitetura em que ArchMine foi aplicada é apresentado. Esse exemplo envolve a recuperação da arquitetura de ArchTrace, uma ferramenta para apoiar a evolução de ligações de rastreabilidade existentes entre uma arquitetura e a sua implementação (MURTA *et al.*, 2006).

O apoio às atividades de ArchToDSSA é ilustrado através de um exemplo hipotético em um domínio de telefonia móvel, uma vez que estudos de caso reais envolvendo ArchToDSSA ainda não foram executados, sendo parte do trabalho de mestrado de KÜMMEL (2007).

Partindo desta Introdução, o restante do capítulo está organizado da seguinte forma: a seção 5.2 apresenta as principais características do ambiente Odyssey para o apoio à ED, Engenharia de Aplicação (EA) e integração das ferramentas desenvolvidas nesta tese; a seção 5.3 apresenta as ferramentas de apoio às atividades de recuperação de arquitetura de ArchMine; a seção 5.4 apresenta a ferramenta de apoio à execução das atividades de comparação de arquiteturas e geração de arquiteturas de referência de domínio parcialmente especificadas de ArchToDSSA; e a seção 5.5 apresenta as considerações finais do capítulo.

5.2 – O Ambiente Odyssey

O ambiente Odyssey (ODYSSEY, 2007) é um ambiente de desenvolvimento de software baseado em reutilização, que disponibiliza funcionalidades e ferramentas para apoio a abordagens de reutilização, como ED, LP e DBC. O Odyssey começou a ser desenvolvido em 1998 (WERNER *et al.*, 1999) e vem sendo evoluído até os dias atuais, através de uma série de trabalhos de doutorado (BRAGA, 2000; BLOIS, 2006; MANGAN, 2006; MURTA, 2006), mestrado (MILER, 2000; XAVIER, 2001; MURTA, 2002; TEIXEIRA, 2003; DANTAS, 2005; LOPES, 2005; MAIA, 2006; OLIVEIRA, 2006a; OLIVEIRA, 2006b) e graduação (MURTA, 1999; DANTAS, 2001; VERONESE e NETTO, 2001; MELO JR, 2005; SILVA, 2005), que desenvolvem ou estendem funcionalidades e ferramentas para o ambiente.

Inicialmente, o Odyssey consistia de uma infra-estrutura contando com diagramadores e funcionalidades/ferramentas de apoio à reutilização. Em 2002, com a incorporação da máquina de processos Charon (MURTA *et al.*, 2002), o Odyssey passou a apoiar a execução de processos de ED e EA, se tornando um ambiente de desenvolvimento de software de fato.

Na sua criação, foram adotadas as linguagens Java (SUN, 2007a), para a sua implementação, e UML como notação para os modelos desenvolvidos em processos de ED e EA. A adoção da recém-criada linguagem Java na ocasião se deu, principalmente, em função da sua promessa de independência de plataforma. Além disso, apesar de ser uma linguagem nova, ela já apresentava uma ampla biblioteca de componentes disponível, dentre eles, componentes de interface gráfica e de comunicação via rede. A adoção da UML, por outro lado, foi motivada pelo fato dessa representar a unificação de diversas notações para a modelagem de software OO, realizada por solicitação da OMG (*Object Management Group*) (OMG, 2007a).

No contexto do trabalho de mestrado de MILER (2000), foi introduzido no Odyssey o modelo de características, com o objetivo de capturar as abstrações de alto nível do domínio, representando as suas capacidades conceituais, funcionais e tecnológicas. OLIVEIRA (2006b) estendeu esse modelo, criando a notação Odyssey-FEX para a modelagem de características. Além do modelo estendido de características, OLIVEIRA (2006b) propôs regras de mapeamento desse modelo para os modelos UML do domínio, permitindo manter a consistência da representação de opcionalidades e variabilidades nos diferentes níveis de abstração da modelagem de um domínio.

Com o desenvolvimento, ao longo dos anos, de novas funcionalidades e ferramentas para o ambiente Odyssey, este foi se tornando um ambiente complexo, gerando problemas de escalabilidade e desempenho para seus usuários. Além disso, a sua manutenção estava se tornando dificultada em função da degradação da sua arquitetura inicial. Esses fatores motivaram uma iniciativa de reestruturação do ambiente em 2003, levando a equipe do projeto Odyssey a analisar o ambiente e classificar as suas funcionalidades em essenciais ou secundárias para a ED e a EA. As funcionalidades essenciais passaram a compor o núcleo (*kernel*) do ambiente e as funcionalidades secundárias foram extraídas do ambiente na forma de ferramentas, ou *plugins*. Essa reestruturação derivou uma nova distribuição do ambiente, denominada OdysseyLight[11], na qual ferramentas podem ser baixadas pelos usuários sob demanda através de um mecanismo de carga dinâmica.

As sub-seções a seguir descrevem brevemente o apoio à ED e à EA no ambiente Odyssey e o mecanismo de carga dinâmica, apresentando alguns *plugins* do ambiente utilizados nesta tese.

5.2.1 – Apoio à Engenharia de Domínio e Engenharia de Aplicação

Inicialmente, a ED e EA eram apoiadas no Odyssey pelos processos Odyssey-DE (BRAGA, 2000) e Odyssey-AE (MILER, 2000). A fim de estender o apoio oferecido à especificação de arquiteturas de referência baseadas em componentes de software no contexto da ED, o processo CBD-Arch-DE e o seu apoio ferramental foram integrados ao Odyssey durante o trabalho de pesquisa de BLOIS (2006). O CBD-Arch-DE prevê um processo de cima para baixo para a especificação de arquiteturas de

[11] Os nomes Odyssey e OdysseyLight são utilizados indiscriminadamente a partir deste ponto da monografia para se referir ao ambiente Odyssey na sua distribuição *Light*.

referência de domínio no Odyssey, ao passo que a abordagem LegaToDSSA, proposta nesta tese, prevê um apoio de baixo de cima, partindo de sistemas legados no domínio, conforme mostra a Figura 5.1.

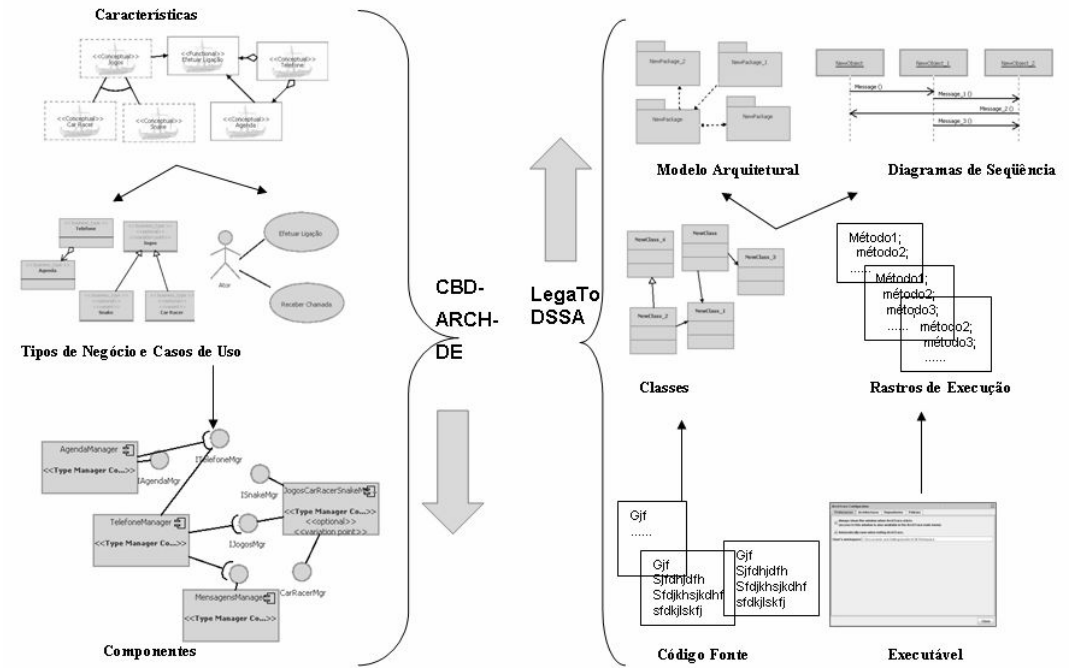

Figura 5.1: Processo CBD-Arch-DE e o processo da abordagem LegaToDSSA.

O Odyssey suporta a representação de opcionalidades e variabilidades nos diferentes níveis de abstração de modelagem do domínio, conforme mostra a Figura 5.2. Isso é possível em função dos mapeamentos entre modelos no ambiente Odyssey, que podem ser realizados de forma manual ou automática. Em (OLIVEIRA, 2006b), são apresentadas heurísticas de mapeamento automático do modelo de características para o modelo de classes. Essas heurísticas são estendidas em (BLOIS, 2006), para o mapeamento do modelo de características para casos de uso e tipos de negócio, chegando até a geração de componentes do domínio. A vantagem desses mapeamentos é a manutenção da consistência de representação de variabilidades e opcionalidades entre os diferentes modelos do domínio.

Como exemplos de mapeamentos, na Figura 5.2 são apresentados 3 modelos para um domínio escolar, construídos no Odyssey, i.e. modelo de características, de classes e de casos de uso. O modelo de características segue a notação Odyssey-FEX, onde características opcionais, como a "Ensino à Distância", têm a sua borda pontilhada. De acordo com as heurísticas para o mapeamento entre modelos de OLIVEIRA (2006b), a característica conceitual e opcional "Ensino à Distância" dá

origem a uma classe no modelo de classes também opcional. A opcionalidade e variabilidade nos modelos UML do domínio no Odyssey são representadas através de estereótipos.

Além de características conceituais, a Odyssey-FEX permite a representação de outros tipos de características de domínio, como características funcionais e de entidade. Como exemplo de característica funcional no modelo da Figura 5.2, tem-se a característica "Matricular Alunos", que dá origem ao caso de uso de mesmo nome. Uma característica funcional também pode dar origem a um método, segundo as heurísticas de mapeamento implementadas no Odyssey. Pontos de variação com suas variantes podem derivar hierarquias no modelo de classes, como mostra o mapeamento das características "Curso", "Graduação" e "PosGraduação" para a hierarquia de classes com nomes de classes equivalentes. Um resumo da taxonomia de características na notação Odyssey-FEX é apresentado no anexo E.

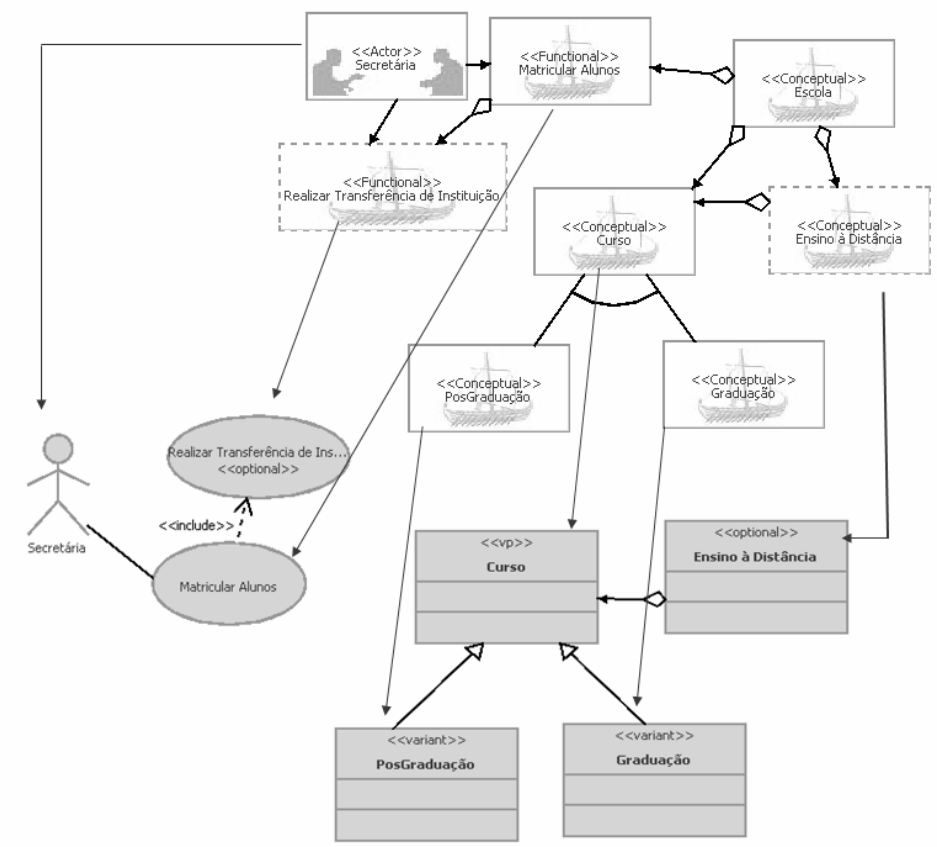

Figura 5.2: Modelos de domínio no Odyssey e seus mapeamentos.

Em função de mapeamentos automáticos ou manuais entre modelos no Odyssey, é estabelecida uma ligação de rastreabilidade entre artefatos em diferentes níveis de abstração. Dessa forma, durante o processo de EA, é possível que a partir da seleção de

um conjunto de características do domínio, o Engenheiro de Aplicação determine os artefatos, nos diferentes níveis de abstração, que vão compor os modelos da sua aplicação.

Além dos mapeamentos de cima para baixo, foram implementados, no contexto desta tese, mapeamentos de baixo para cima do modelo de classes para o modelo de características. Esses mapeamentos foram implementados com base na adaptação das heurísticas de mapeamento apresentadas em (OLIVEIRA, 2006b), invertendo o seu sentido. O objetivo desses mapeamentos no sentido inverso é permitir que os modelos arquiteturais de domínio, gerados por LegaToDSSA, possam ser mapeados para características do domínio no Odyssey, permitindo o recorte e a instanciação de novas aplicações no domínio. Maiores detalhes sobre a notação Odyssey-FEX e as regras de mapeamento entre modelos no ambiente Odyssey podem ser obtidos em (OLIVEIRA, 2006b) e (BLOIS, 2006). As heurísticas de mapeamento no sentido inverso, i.e. de classes para características, que representam uma contribuição desta tese, estão detalhadas no Anexo F.

5.2.2 – Carga Dinâmica

Conforme mencionado, o ambiente Odyssey passou por uma reestruturação em 2003, dando origem à sua distribuição "*light*", denominada OdysseyLight. Como mostra a Figura 5.3, o ambiente Odyssey passou a contar com um núcleo de funcionalidades, além de ferramentas disponibilizadas na forma de *plugins*. Nessa reestruturação, funcionalidades como a modelagem de domínio, a gerência de processos (MURTA, 2002) e a geração de componentes de software (BLOIS, 2006) foram mantidas no núcleo, ao passo que funcionalidades classificadas como secundárias foram extraídas na forma de *plugins*, como, por exemplo, o suporte a padrões (XAVIER, 2001), o suporte à extração de modelos de classes (VERONESE e NETTO, 2001), a notificação de críticas sobre modelos (DANTAS *et al.*, 2001), a edição concorrente de modelos, a exportação e importação de modelos em XMI, dentre outras.

Os *plugins* devem ser baixados pelos usuários do Odyssey sob demanda. Uma lista de *plugins* é mantida no servidor do grupo de reutilização da COPPE e acessada por cada instância do Odyssey, permitindo que os seus usuários selecionem os *plugins* que desejam baixar. Os *plugins* conhecem o núcleo do Odyssey e podem acessá-lo, enquanto o Odyssey não precisa conhecer os seus *plugins*. O padrão de projeto

Observer (GAMMA *et al.*, 1995) foi utilizado para garantir que o Odyssey não possua acoplamento explícito com os seus *plugins*.

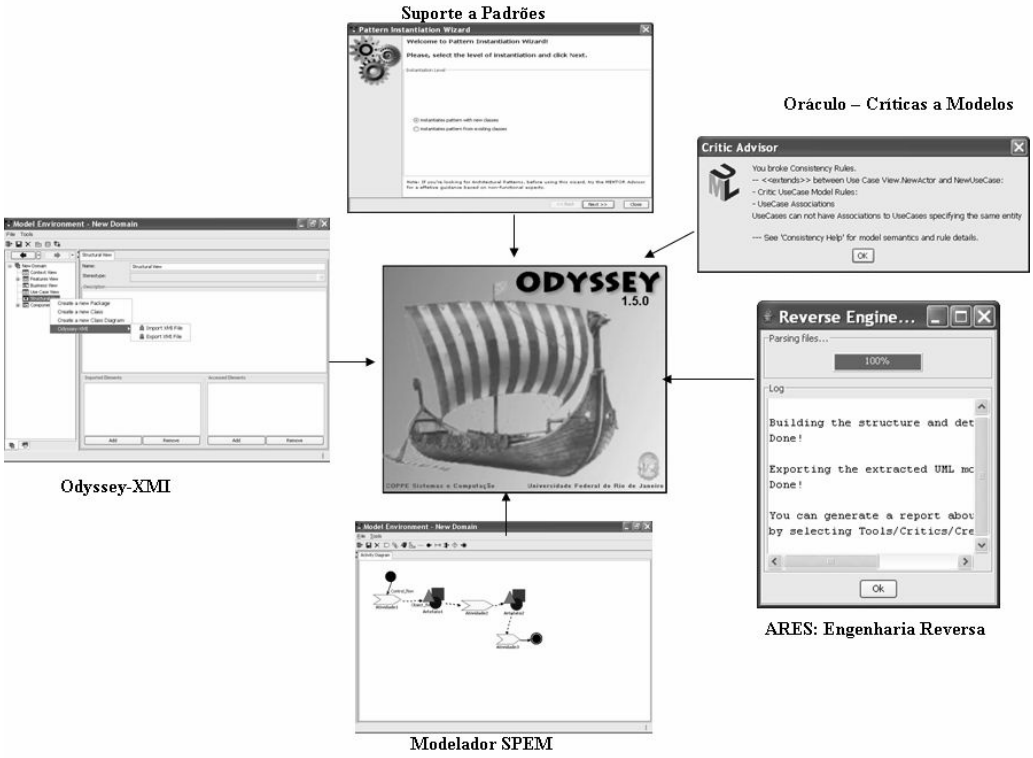

Figura 5.3: Ambiente OdysseyLight e exemplos de alguns de seus *plugins*.

Cada *plugin* do Odyssey deve implementar a interface *Tool* (Figura 5.4), a fim de que suas opções de menu sejam disponibilizadas no Odyssey. Essa interface especifica serviços como *getEnvironmentMenu*(), para disponibilizar uma opção de menu na tela principal do Odyssey, e *getPopupMenu*(), para disponibilizar uma opção de menu *popup* no ambiente. Uma vez que um *plugin* seja baixado por um usuário, ele entra na lista de ferramentas que o Odyssey acessa através dos serviços da interface *Tool*. Maiores detalhes sobre o mecanismo de carga dinâmica do Odyssey podem ser obtidos em (MURTA *et al.*, 2004).

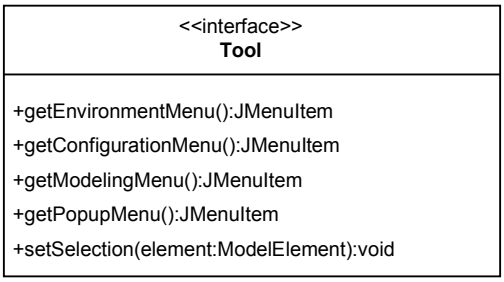

Figura 5.4: Interface *Tool* implementada pelos *plugins* do OdysseyLight.

5.3 – Ferramentas que Apóiam a Execução das Atividades de ArchMine

Nesta seção, são descritas as ferramentas que apóiam a execução das atividades do processo de recuperação de arquitetura de ArchMine. As ferramentas são descritas seguindo a ordem das atividades do processo, o que permite uma leitura transversal das atividades e seu apoio ferramental.

5.3.1 – Ares: Apoio à Extração da Estrutura Estática

A atividade de **Extração da Estrutura Estática** é realizada com o apoio da ferramenta de Engenharia Reversa estática disponível como *plugin* do ambiente Odyssey, a Ares (VERONESE e NETTO, 2001). A Ares lê código fonte Java e gera modelos de classes e pacotes no ambiente Odyssey, com seus relacionamentos. A sua arquitetura segue o estilo arquitetural de dutos e filtros, conforme mostra a Figura 5.5. Cada etapa do processo de análise do código e extração do modelo deve ser executada completamente antes que a ferramenta possa prosseguir para a próxima etapa.

Na arquitetura da Ares, um primeiro filtro, o **Filtro *Parser***, processa os arquivos do código fonte à procura de informações relevantes para a construção do modelo e as coloca em memória, ainda sem a organização necessária. Um segundo filtro, o **Filtro de Inferência**, organiza os elementos do modelo em uma árvore de pacotes e detecta a existência de relacionamentos entre as classes, interfaces e entre os pacotes. Um último filtro, o **Filtro de Transferência**, percorre a estrutura de pacotes do modelo, instanciando pacotes, classes, interfaces e relacionamentos nos meios externos. Os meios externos podem ser o ambiente Odyssey ou uma base de fatos em Prolog.

O **Filtro *Parser*** e o **Filtro de Transferência** estão organizados através de hierarquias de classes, de forma que existe uma superclasse que define serviços comuns e subclasses para linguagens de programação ou formatos de saída específicos. Essa arquitetura provê uma flexibilidade à ferramenta, sendo possível incorporar novas linguagens de programação ou formatos de saída sem impactar partes não relacionadas da ferramenta. Por exemplo, caso seja necessário interpretar código escrito em uma outra linguagem de programação, como C++, basta que o **Filtro *Parser*** seja estendido.

O **Filtro *Parser*** faz uso de um *parser* para a análise léxica e sintática dos arquivos do código fonte Java. Esse *parser* é gerado automaticamente pela ferramenta

JavaCC (SUN, 2007b), distribuída gratuitamente. O *parser* é gerado com base na gramática da linguagem Java, versão 1.5, que é distribuída com a ferramenta.

A Ares recupera um modelo bastante rico de informações em relação ao metamodelo da UML. Dentre os elementos recuperados pela Ares, encontram-se: classificadores (i.e. classe, interface, tipo de dados primitivo – *DataType*) e sua estrutura interna (ex: métodos e atributos); pacotes; associações com cardinalidade, navegabilidade, papel, mutabilidade e visibilidade; dependências, recuperadas a partir da importação de elementos, passagem de parâmetros, retorno de objetos, instanciação de objetos, declaração de variáveis locais; e conversões de tipos (*type casting*).

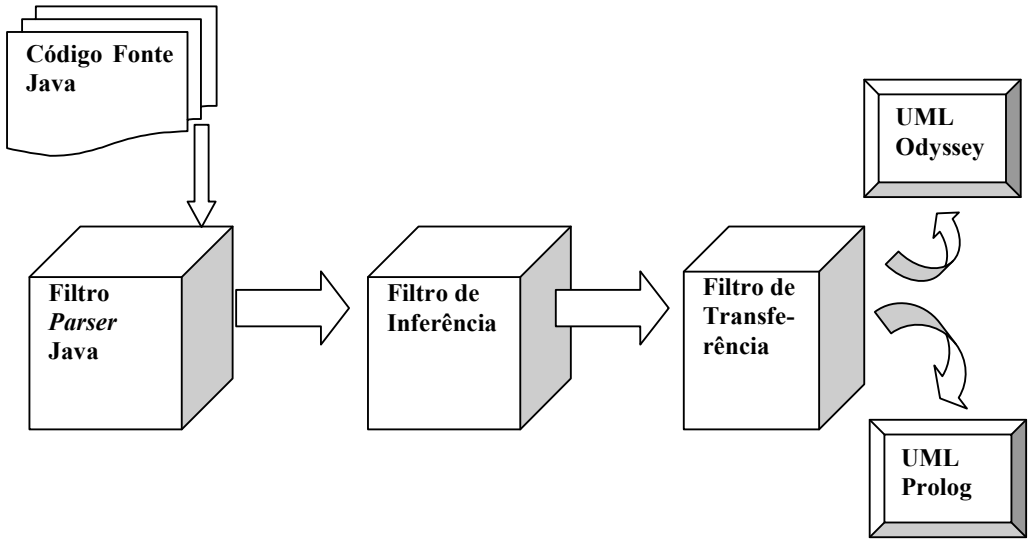

Figura 5.5: Visão geral da arquitetura da ferramenta Ares: Engenharia Reversa de modelo estático.

A Figura 5.6 apresenta a ferramenta Ares no ambiente Odyssey recuperando um modelo estático da UML a partir do código da ArchTrace. A Ares permite uma busca recursiva pelos diretórios do código fonte, conforme mostra a Figura 5.6, através da opção "*Search Subdirectories Recursively* ".

O modelo é extraído para a visão estrutural do OdysseyLight (i.e. *Structural View*), conforme mostra a Figura 5.7. Convém ressaltar que a Ares extrai os elementos semânticos do modelo e não se preocupa com os elementos sintáticos ou diagramáticos, ficando a cargo do usuário remontar os diagramas de classes e pacotes com base nos elementos semânticos, conforme desejado.

5.3.2 – Odyssey: Definição de Cenários de Casos de Uso

A definição de cenários de casos de uso é realizada no ambiente Odyssey pelo

desenvolvedor que recupera a arquitetura, seguindo as diretrizes apresentadas na seção 4.2.2.

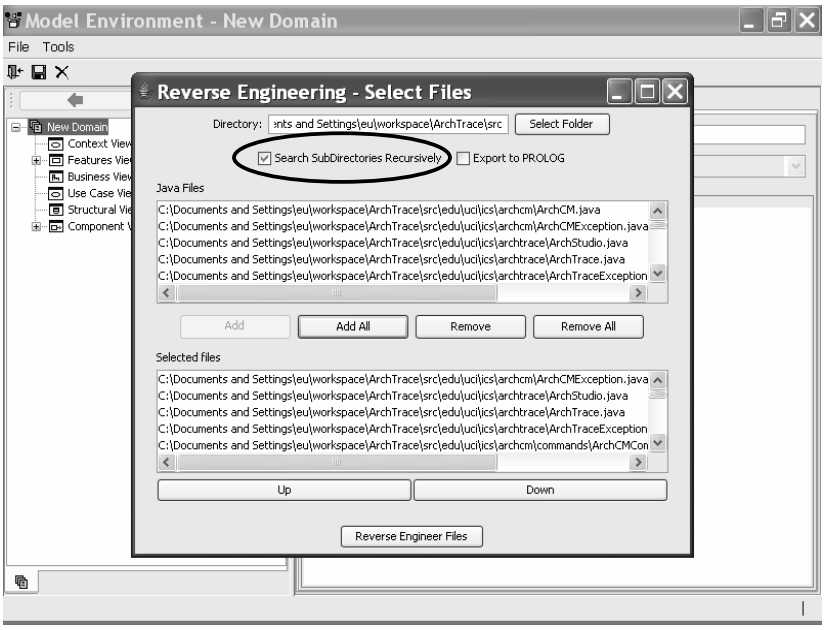

Figura 5.6: Ferramenta Ares no ambiente Odyssey.

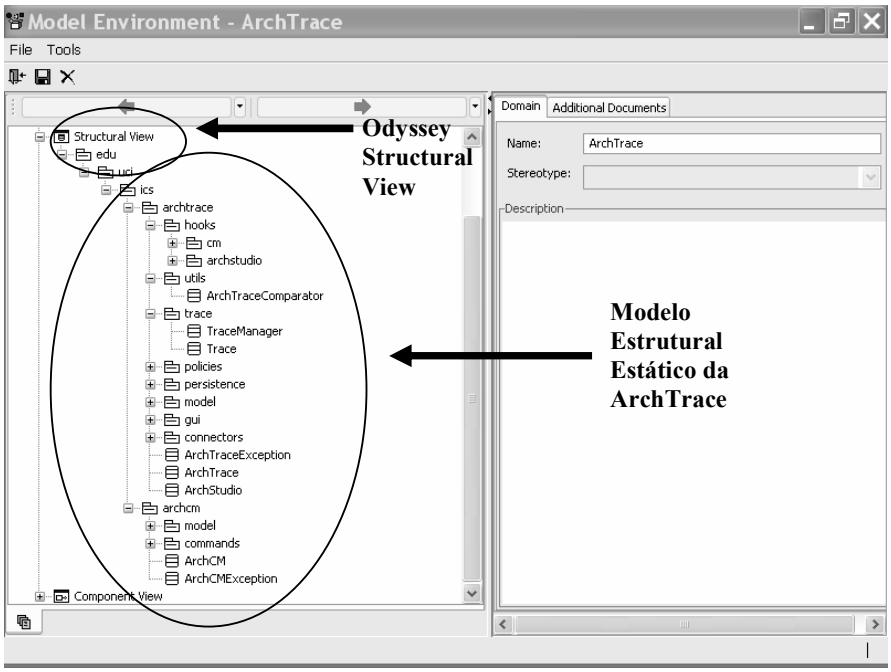

Figura 5.7: Modelo estrutural estático da ArchTrace recuperado com a Ares no Odyssey.

A Tabela 5.1 apresenta os cenários de casos de uso definidos para a ArchTrace. ArchTrace lê uma arquitetura a partir de um repositório de arquiteturas e uma configuração de implementação correspondente a partir de um repositório de código fonte. O usuário deve, inicialmente, estabelecer ligações de rastreabilidade

manualmente. A partir daí, a cada modificação na arquitetura ou no código fonte, ArchTrace é notificada, executando uma série de políticas de gerência de rastreabilidades para manter essas ligações atualizadas e válidas. ArchTrace é notificada, pois apresenta "escutadores de eventos" para cada um dos repositórios. A ferramenta permite ainda que o usuário ligue ou desligue as políticas. A Figura 5.8 apresenta as opções na interface da aplicação que originaram alguns dos cenários de casos de uso na Tabela 5.1, com a indicação da diretriz correspondente.

Tabela 5.1: Cenários de casos de uso definidos para a ArchTrace.

Cenários de Casos de Uso de ArchTrace	
1. Configurar preferências.	8. Remover ligações de rastreabilidade para versões antigas de arquivos.
2. Configurar arquitetura.	9. Proibir criação de ligações de rastreabilidade para elemento arquitetural imutável.
3. Configurar repositório.	10. Executar política de mineração de dados.
4. Configurar políticas.	11. Atualizar ligações de rastreabilidade quando novas versões de arquivos são atualizadas no repositório.
5. Salvar arquivo.	12. Abrir o *Help*.
6. Estabelecer ligações de rastreabilidade para diferentes versões de arquivos.	13. Proibir a criação de uma ligação de rastreabilidade para um arquivo quando já existe uma ligação pro diretório.
7. Remover ligações de rastreabilidade para arquivos quando uma ligação pro diretório é criada.	14. Sugerir ligações de rastreabilidade para as versões mais recentes de arquivos.

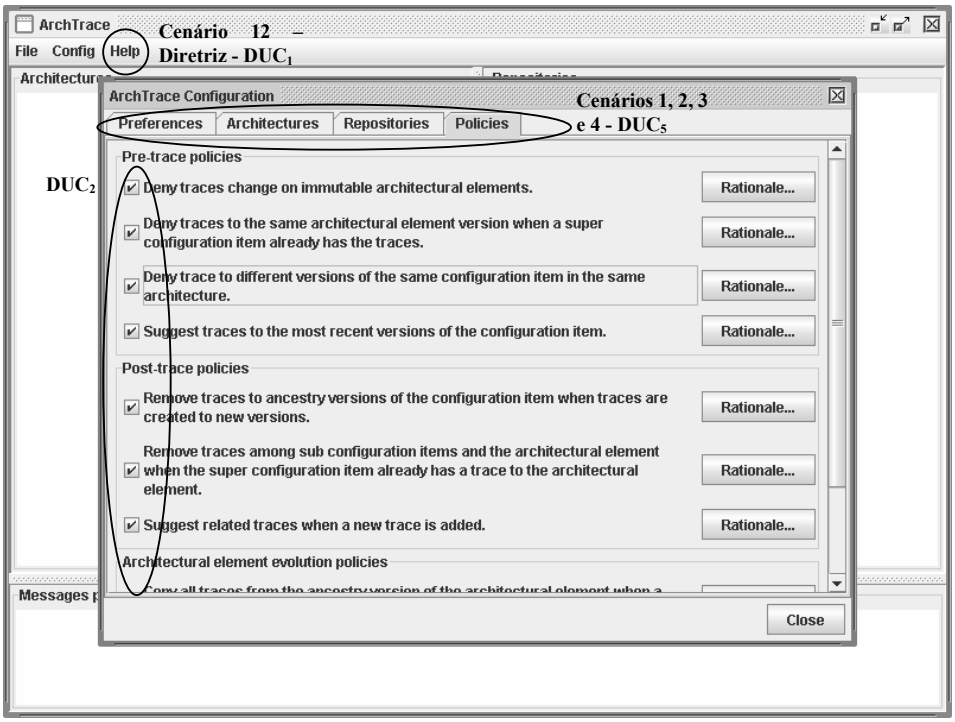

Figura 5.8: Interface gráfica de ArchTrace.

5.3.3 – Tracer: Coleta de Rastros de Execução

A ferramenta Tracer (VASCONCELOS *et al.*, 2005; CEPEDA *et al.*, 2006) é responsável pelo monitoramento da aplicação e coleta dos rastros de execução referentes aos cenários de casos de uso definidos na atividade anterior. O desenvolvimento de uma nova ferramenta de monitoramento de programas foi motivado, principalmente, pelo fato das ferramentas pesquisadas não permitirem a delimitação dos rastros de execução pertencentes a cada cenário de caso de uso. Além disso, os rastros de execução precisavam ser armazenados em um formato que facilitasse o seu pós-processamento por diferentes ferramentas.

A Tracer utiliza programação orientada a aspectos, através da tecnologia AspectJ (ASPECTJ, 2007), uma extensão da linguagem Java para a programação com aspectos. A programação orientada a aspectos pode ser definida como uma tecnologia que permite separar os interesses tranversais (i.e. *crosscutting concerns*) dos interesses-base de um sistema, fazendo com que o software se torne mais inteligível e, conseqüentemente, facilitando a sua manutenção e reutilização (KICZALES *et al.*, 1997). Como exemplos de interesses transversais têm-se o tratamento de segurança, persistência, depuração, rastreamento etc.

Nesta tese, a tecnologia de aspecto é utilizada para o rastreamento da execução do programa. Esses interesses são encapsulados em uma unidade modular chamada aspecto, que além das construções típicas de uma classe, envolve construções próprias do desenvolvimento orientado a aspectos, como: *join points* (pontos de junção), que definem pontos no fluxo de controle do programa que podem receber o código especificado no aspecto; *pointcuts* (conjunto de pontos de junção), que definem conjuntos de pontos de junção que vão receber o código do aspecto; *advice* (adendo), código executado quando um *join point* é alcançado.

AspectJ permite que o código definido nos aspectos seja inserido no código binário através de um processo de pós-compilação ou combinação (*weaving*). Dessa forma, é possível realizar a instrumentação sem modificar o código fonte da aplicação. A Figura 5.9 apresenta o diagrama de classes simplificado da ferramenta Tracer e a Figura 5.10 apresenta a tela principal da ferramenta.

A classe *TracerWindow,* na Figura 5.9, é responsável pela entrada de dados do usuário. A classe *TracerEngine* atua como controladora da *TracerWindow*, tratando os eventos de entrada dos usuários. O aspecto de rastreamento da execução de métodos é

definido através do aspecto abstrato *TracerApect*, estendido pelo aspecto *TraceMyClasses*. Na classe abstrata *TracerAspect* são definidos *pointcuts* concretos e abstratos, i.e. execuções de métodos (*pointcut myMethod*), construtores das classes (*pointcut myConstructor*) e classes que devem ser monitoradas (*pointcut myClass*), sendo esse último um *pointcut* abstrato. Os *pointcuts myMethod* e *myConstructor* utilizam o *pointcut myClass*, a fim de estabelecer as classes que devem ser monitoradas. *TraceMyClasses* estende o *pointcut myClass*.

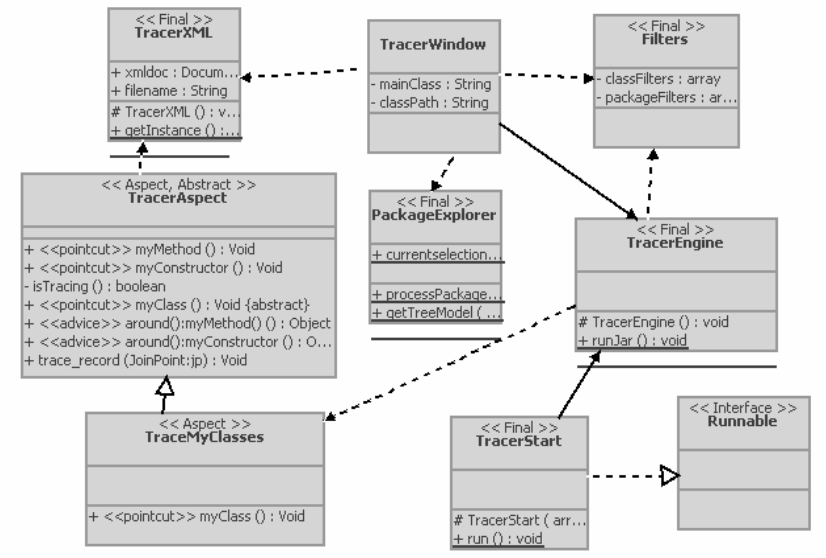

Figura 5.9: Diagrama de classes da ferramenta Tracer.

A Tracer utiliza o *advice around*, que permite que o código de rastreamento seja executado antes do início e após a execução de cada método, permitindo que a Tracer registre o início e término de cada execução de método. *TracerAspect* utiliza a interface *JoinPoint* do AspectJ para recuperar informações de contexto acerca de uma execução de método, como a instância que recebe a chamada, seu tipo (i.e. a classe) e o método que está sendo executado. Além disso, o nome da *thread* corrente também é recuperado. A saída é gravada em XML, através da classe *TracerXML*.

Conforme apresenta a Figura 5.10, o usuário deve informar o código binário (arquivo Jar – "Jar File"), o *classPath* (ClassPath), filtros (Static Filters) e arquivo de *log* (Log File) para a Tracer. O arquivo Jar é o que vai receber o código de rastreamento. Dessa forma, somente as classes pertencentes à aplicação de fato são monitoradas, não sendo incluídas, por exemplo, classes de bibliotecas, nesse processo. A Tracer descobre a classe principal (*MainClass*) da aplicação através de informações no *manifest* do Jar, que contém meta-dados sobre o arquivo Jar. Embora em casos onde

a aplicação possua mais de uma classe principal, essa pode ser informada pelo usuário.

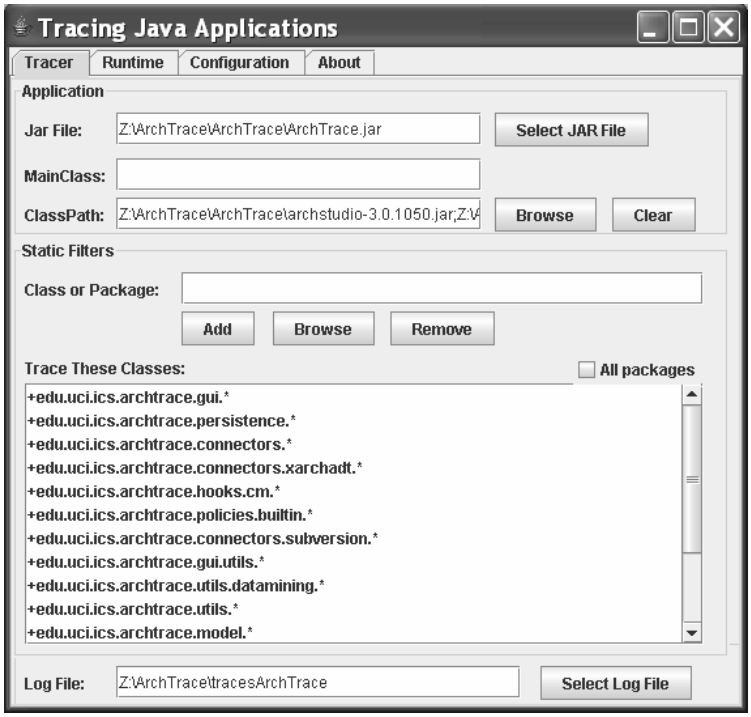

Figura 5.10: Tela principal da ferramenta Tracer.

Através do *classPath*, o usuário informa as bibliotecas e outros recursos utilizados pela aplicação para que a Tracer possa alterar o seu *classloader* (mecanismo que o Java utiliza para carregar classes e outros recursos da aplicação) de modo que este contenha todas as referências para esses recursos externos. Os filtros permitem que o usuário selecione classes e pacotes para o monitoramento, sendo que para a mineração e reconstrução dos elementos arquiteturais por ArchMine, todas as classes da aplicação devem ser monitoradas.

Finalmente, a informação de arquivo de *log* (i.e. *Log File*) representa o nome do arquivo onde a Tracer grava os rastros de execução. É possível gravar um rastro de execução em cada arquivo, onde o nome do arquivo de *log* é agregado de um seqüencial, ou vários rastros no mesmo arquivo. De posse dessas informações, a Tracer pode iniciar a execução da aplicação alvo, gerando como saída um ou mais arquivos XML contendo os métodos invocados em tempo de execução, com a sua informação de contexto, como *thread* (fluxo de execução), classe e instância, ordenados e identados pela sua ordem de chamada. A Figura 5.11 apresenta um exemplo de um trecho de arquivo XML de rastros de execução gerado pela Tracer para a aplicação ArchTrace.

A Tracer permite habilitar e desabilitar a coleta de rastros, delimitando o rastro de execução ou seqüência de execuções de métodos pertencente a um cenário de caso de

uso. Isso é possível através de propriedades do sistema controladas pela classe *System* do Java que permite, por exemplo, verificar o estado atual do programa em relação à coleta de rastros. O cenário de caso de uso no arquivo XML é representado através da *tag* "*Label*", conforme mostra a Figura 5.11, que representa um trecho do arquivo do rastro de execução para o cenário "*Configures architecure*" da ArchTrace.

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
 <Label name="Configures architecture">
  <Method class="edu.uci.ics.archtrace.gui.ArchTraceTreeModel"
     instance="@9195d0" method="getChildCount" thread="AWT-EventQueue-0"
     time="24 de Março de 2006 14h34min57s BRT">
     <Method class="edu.uci.ics.archtrace.model.ArchTraceElement"
       instance="@2c25f" method="getChildCount" thread="AWT-EventQueue-0"
       time="24 de Março de 2006 14h34min57s BRT" />
 </Method>
 <Method class="edu.uci.ics.archtrace.gui.ArchTraceTreeModel"
    instance="@9195d0" method="getChild" thread="AWT-EventQueue-0"
    time="24 de Março de 2006 14h34min57s BRT">
    <Method class="edu.uci.ics.archtrace.model.ArchTraceElement"
       instance="@2c25f " method="getChild" thread="AWT-EventQueue-0"
       time="24 de Março de 2006 14h34min57s BRT" />
 </Method>
.....
</Label>
```

Figura 5.11: Exemplo de um trecho de arquivo de rastros de execução gerado pela Tracer.

Convém ressaltar que o desempenho de execução da aplicação não é impactado pela Tracer, havendo algum consumo de tempo no processo de *weaving*, mas que também não é significativo. A Tracer utiliza recursos de reflexão da linguagem Java para executar a aplicação alvo através do método "*main*" da sua classe principal. Vale ressaltar que a Tracer não é um *plugin* do OdysseyLight, funcionando de forma independente (*standalone*).

5.3.4 – TraceMining e Phoenix: Avaliação da Cobertura de Classes nos Cenários e Reconstrução do Modelo Arquitetural

Nesta seção, são apresentadas as ferramentas TraceMining e Phoenix para apoio à reconstrução do modelo arquitetural do sistema. A TraceMining apóia a avaliação da cobertura de classes nos cenários e a reconstrução do modelo arquitetural estático, através das atividades de mineração e reconstrução dos elementos arquiteturais, derivação de nomes e de relacionamentos entre elementos arquiteturais. A Phoenix apóia a reconstrução de modelos dinâmicos, cobrindo a atividade de reconstrução de modelos dinâmicos, conforme apresentado na Figura 4.4.

5.3.4.1 – TraceMining: Avaliação da Cobertura de Classes, Reconstrução de Elementos Arquiteturais e Derivação de seus Nomes e Relacionamentos

A ferramenta TraceMining (ODYSSEY, 2007) lê os rastros de execução gerados pela Tracer e realiza a mineração. A Figura 5.12 apresenta a sua arquitetura.

A primeira fase de um processo de mineração é o pré-processamento, ou a transformação da base de dados para um formato passível de ser minerado. Essa fase é realizada pelas classes do pacote *preparation*. Porém, antes da realização dessa atividade, TraceMining verifica a cobertura de classes nos cenários, comparando as classes nos rastros de execução com as classes do modelo de classes extraído com a ferramenta Ares, descrita na seção 5.3.1. O modelo extraído com a Ares é exportado para o ambiente Odyssey, que, por sua vez, pode gerar um arquivo XML com a relação das classes do modelo, através de um de seus plugins, o Odyssey-Metrics (MELO JR, 2005). A TraceMining pode ler esse XML e comparar as classes com as classes dos rastros de execução, gerando uma relação textual de classes não monitoradas. Essa atividade é realizada pelas classes do pacote *preparation*.

Após o apoio à atividade de avaliação da cobertura de classes nos cenários, as classes do pacote *preparation* realizam a preparação da base de dados para a mineração, lendo os rastros de execução em XML gerados pela Tracer, e a cada *tag* "Label" nos rastros uma transação na base de transações é criada. A cada ocorrência de classe nos rastros de execução, um item de dado de uma transação é criado. TraceMining utiliza uma biblioteca de mineração de regras de associação, desenvolvida pelo próprio grupo de reutilização da COPPE, que se encontra atualmente em sua versão 1.2.0 (i.e. a DataMining-1.2.0.jar) (DANTAS, 2005).

O elemento arquitetural *integration* é responsável pela integração da TraceMining com o Odyssey, implementando a interface *Tool* (Figura 5.4), requerida pelo Odyssey para os seus *plugins*. A TraceMining também pode funcionar de forma independente em relação ao Odyssey, gerando as sugestões de grupos de classes em arquivo texto.

TraceMining apóia a aplicação das heurísticas definidas para o processo de mineração em ArchMine, descritas na seção 4.2.5.1, conforme pode ser visto nas telas da ferramenta e exemplo apresentado a seguir. O exemplo retrata parte da mineração realizada no processo de recuperação de arquitetura da ArchTrace.

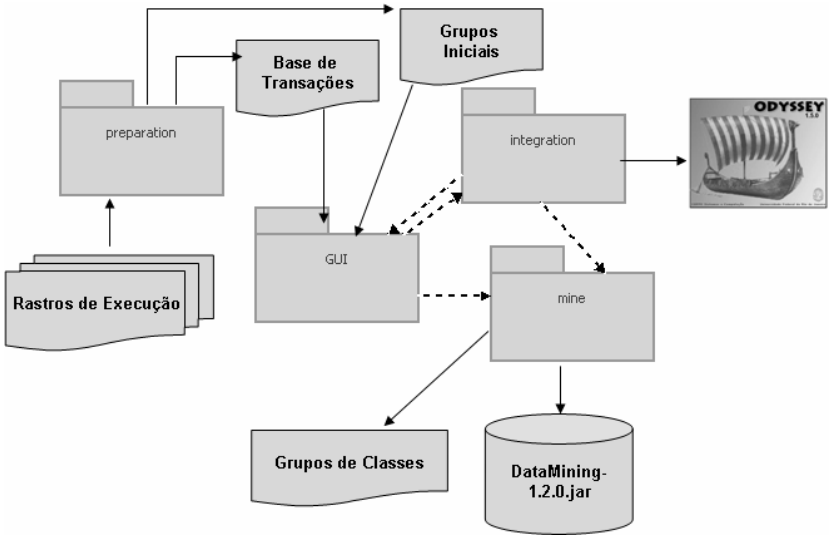

Figura 5.12: Visão geral da arquitetura da ferramenta de mineração TraceMining.

A Figura 5.13 apresenta a tela de mineração da TraceMining, onde as seguintes heurísticas são contempladas: **H1**, que sugere que o valor de suporte mínimo seja baixo, tendo sido estabelecido, nesse caso, um valor igual a 0%; **H2**, que sugere diretrizes para o estabelecimento da confiança mínima, sendo que o valor de 60% foi utilizado na recuperação da arquitetura de ArchTrace; a heurística **H3,** que sugere que classes sejam mineradas dos maiores para os menores valores de suporte, onde, nesse caso, antecedentes estão sendo escolhidos aleatoriamente pela ferramenta numa faixa de valor de suporte de 70 a 100%, embora, caso a TraceMining detecte classes ainda não mineradas em uma faixa de suporte maior que a informada, o usuário seja avisado; **H4**, que sugere que grupos de classes já formados nos primeiros ciclos de mineração sejam filtrados dos ciclos de mineração seguintes. Uma vez que a Figura 5.13 apresenta os dados de entrada para o primeiro ciclo de mineração de ArchTrace, o filtro de grupos já formados está desmarcado.

A Figura 5.14 apresenta os resultados do primeiro ciclo de mineração de ArchTrace, mostrando as classes associadas com o antecedente ArchTraceElement, com o seu suporte e confiança em relação à ArchTraceElement. A Figura 5.15 demonstra a aplicação da heurística **H5**, uma vez que o primeiro elemento arquitetural de ArchTrace está sendo formado a partir da interseção entre resultados das minerações para os antecedentes selecionados.

A Figura 5.16 demonstra a aplicação da heurística **H6**, que determina o agrupamento de classes não agrupadas, durante o processo de mineração, no elemento arquitetural para o qual elas apresentem uma melhor solução de compromisso entre

suporte e confiança. A TraceMining calcula a **diferença de suporte (ds)**, apresentada como "Support Diff" na ferramenta, e o **valor médio de confiança (mc)**, apresentada como "Average Confidence", da classe não agrupada para cada grupo ou elemento arquitetural já formado. Essas métricas são sugeridas pela heurística **H6**, descrita na seção 4.2.5.1. Porém, a TraceMining ainda não calcula automaticamente a solução de compromisso, que pode ser inferida pelo usuário.

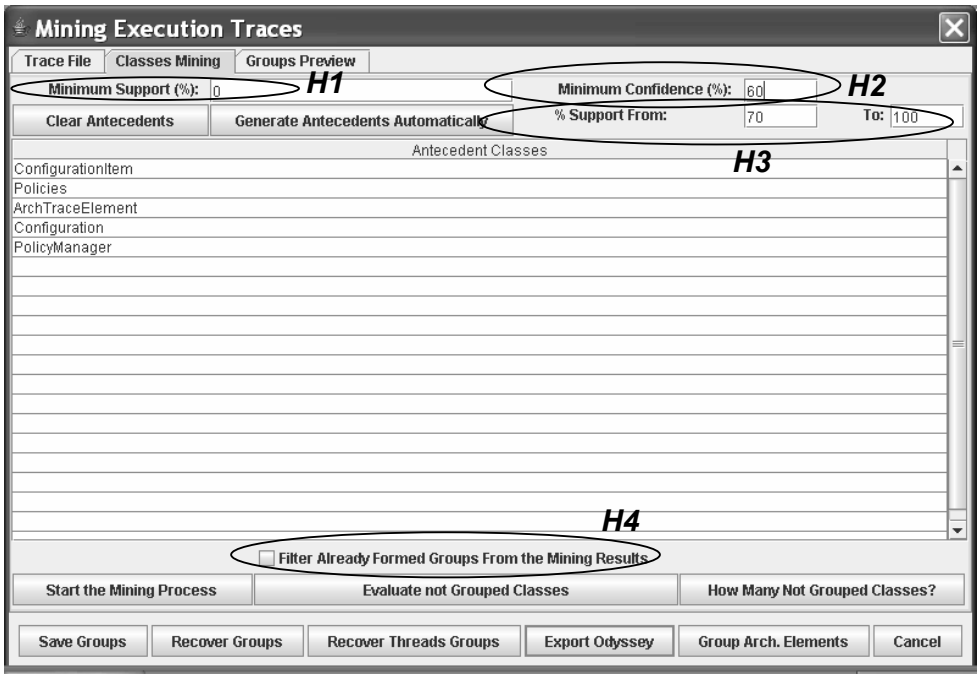

Figura 5.13: TraceMining: tela de mineração.

A heurística **H7** é implementada pelo elemento arquitetural *preparation* (Figura 5.12), que transforma os rastros de execução numa base de transações, permitindo que o usuário escolha se as *threads* secundárias devem compor novas transações ou devem fazer parte da transação referente ao cenário de caso de uso em que são executadas. Além disso, caso o usuário escolha a opção de agrupar classes cujos métodos não são executados na *thread* principal, alguns grupos de classes já podem ser formados nesse momento, como mostra a geração de "Grupos Iniciais" na Figura 5.12. Vale ressaltar que a *thread* principal para aplicações de interface gráfica em Java é a **AWT-EventQueue-0,** conforme mostra a Figura 5.11.

Finalmente, a heurística **H8** é implementada na exportação dos resultados para o Odyssey, uma vez que não representa uma heurística do processo de mineração em si, mas do processo de agrupamento. A fim de que, no processo de exportação dos resultados para o Odyssey, possam ser indicados elementos arquiteturais onde superclasses e interfaces devem ser agrupadas, a ferramenta de métricas do Odyssey, o

plugin Odyssey-Metrics (MELO JR, 2005) é utilizado. Ela permite que o usuário configure as métricas desejadas sobre o modelo de pacotes e classes OO e extraia seus resultados. Essa ferramenta também foi elaborada no contexto da pesquisa desenvolvida nesta tese, através de um trabalho de projeto final do grupo de reutilização.

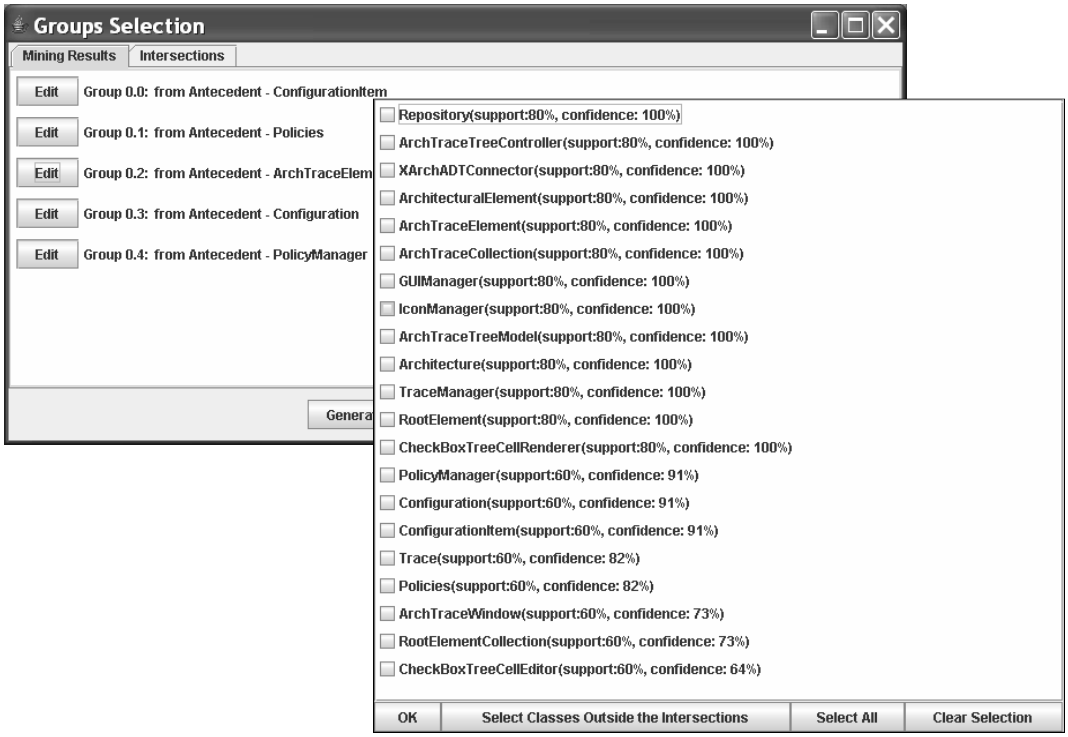

Figura 5.14: Tela de apresentação dos resultados da mineração de TraceMining.

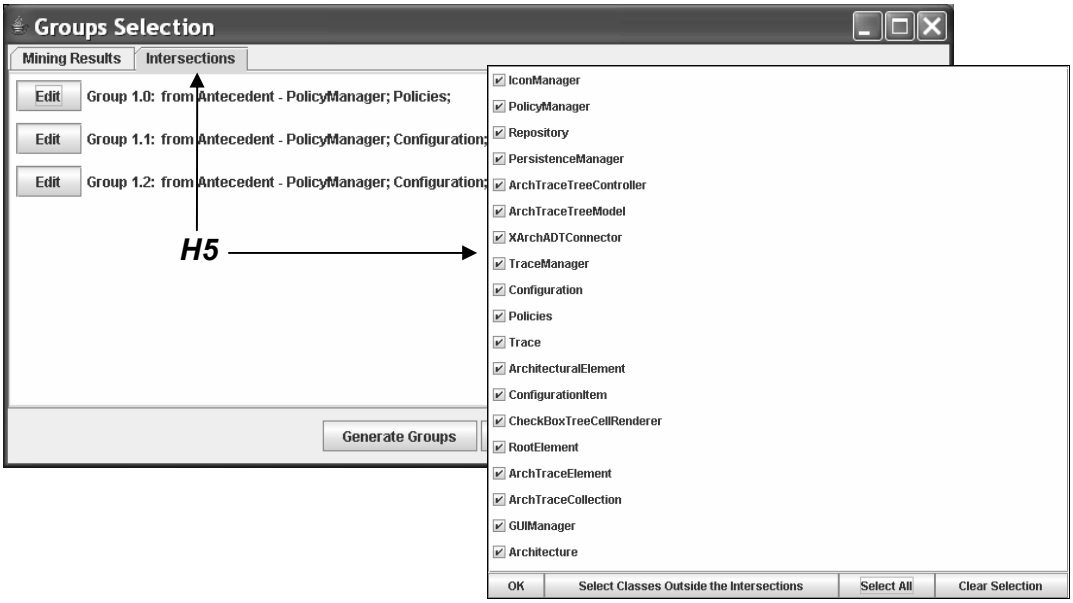

Figura 5.15: Tela de apresentação de interseções entre os resultados da mineração de TraceMining.

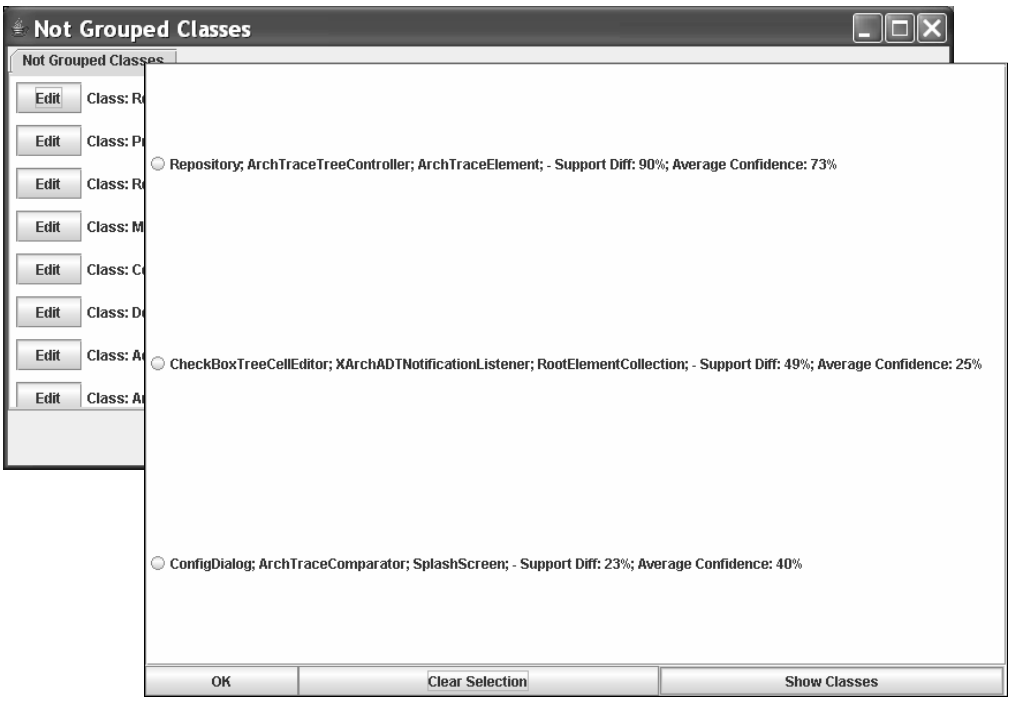

Figura 5.16: Tela de agrupamento de classes não agrupadas na TraceMining.

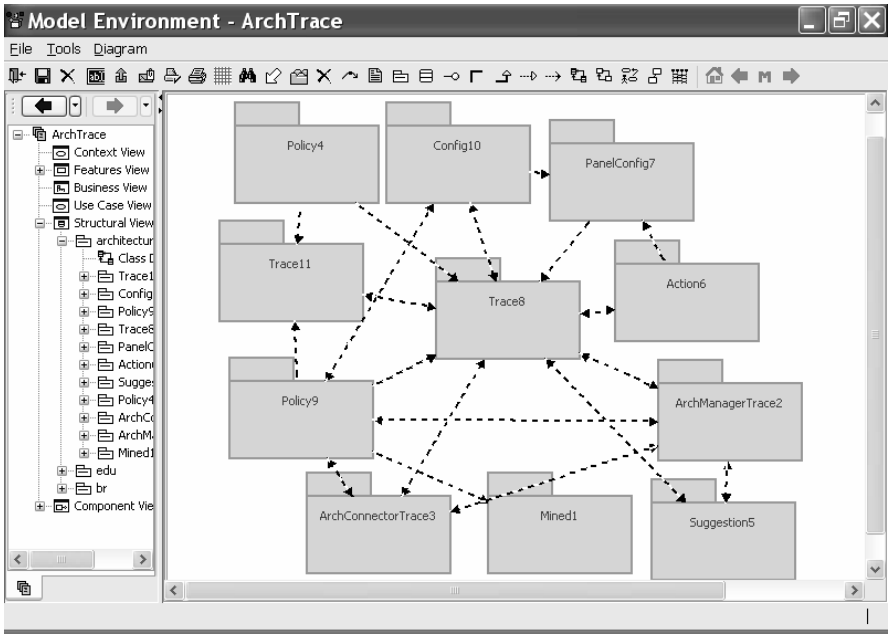

Figura 5.17: Arquitetura recuperada para a ArchTrace.

A derivação de nomes dos elementos arquiteturais é realizada após a formação dos grupos de classes, através das diretrizes descritas na seção 4.2.5.2. Quanto aos relacionamentos entre elementos arquiteturais, a derivação é responsabilidade do

elemento arquitetural *integration* (Figura 5.12) da TraceMining. Relacionamentos entre elementos arquiteturais são derivados apenas durante a exportação da arquitetura recuperada para o Odyssey. O metamodelo do Odyssey é lido diretamente pelo elemento arquitetural *integration* da TraceMining, a fim de descobrir os relacionamentos entre as classes de diferentes elementos arquiteturais e derivar os relacionamentos entre os respectivos elementos arquiteturais. A Figura 5.17 apresenta a arquitetura recuperada para a ArchTrace e reconstruída no ambiente Odyssey.

É importante ressaltar que embora na recuperação da arquitetura da ArchTrace tenha sido selecionada a opção de nomes de classes não qualificados, a TraceMining permite utilizar nomes de classes qualificados na mineração e na exportação dos resultados para o Odyssey. Essa opção se torna importante quando, por exemplo, um sistema possui duas classes com o mesmo nome em diferentes pacotes.

5.3.4.2 – Phoenix: Reconstrução de Modelos Dinâmicos

A ferramenta Phoenix (VASCONCELOS *et al.*, 2005; CEPEDA *et al.*, 2006) lê os rastros de execução gerados pela Tracer, realizando uma busca em profundidade e em largura na árvore de execuções de métodos, e reconstrói os diagramas de seqüência correspondentes no ambiente Odyssey. A Figura 5.18 apresenta a visão geral da arquitetura da ferramenta Phoenix.

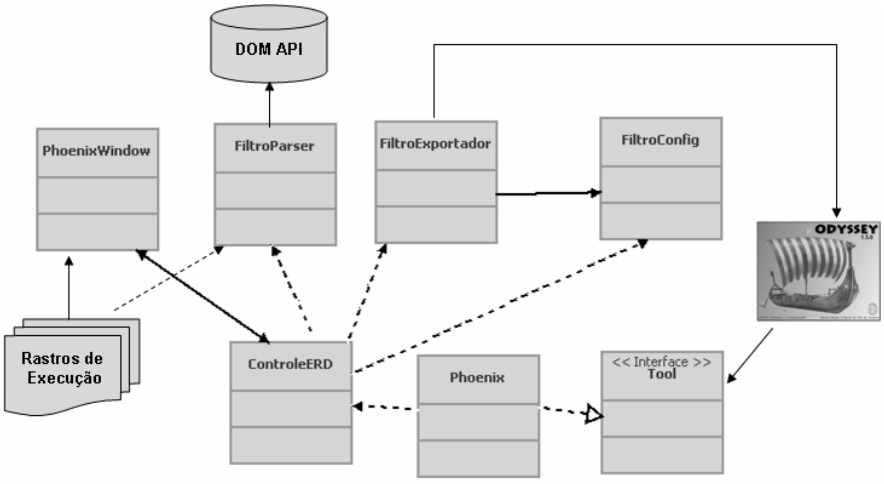

Figura 5.18: Visão geral da arquitetura da ferramenta Phoenix: engenharia reversa de modelos dinâmicos.

A arquitetura da Phoenix segue o padrão arquitetural MVC, misturado com o estilo arquitetural de dutos e filtros. A classe *ControleERD* atua no papel de controladora, tratando os eventos da interface *PhoenixWindow*, que atua no papel de visão. O modelo é criado diretamente no ambiente Odyssey, através do filtro

FiltroExportador. O modelo é representado por elementos de modelagem que compõem o diagrama de seqüência da UML, como objetos, suas linhas de vida (*lifelines*) e mensagens trocadas entre eles.

O *FiltroParser* é responsável pela leitura e interpretação dos rastros de execução gerados pela Tracer, utilizando a biblioteca DOM (W3C, 2007) para a interpretação de arquivos XML. DOM faz parte da API do Java para processamento de arquivos XML. A classe *Phoenix* implementa a interface Tool (Figura 5.4) do Odyssey, a fim de permitir que a ferramenta funcione como um *plugin* para o ambiente. A classe *FiltroConfig* é responsável por armazenar configurações acerca dos filtros selecionados pelo usuário.

Os diagramas de seqüência reconstruídos pela Phoenix são associados a casos de uso no ambiente Odyssey, como mostra a Figura 5.19, que apresenta um diagrama de seqüência em nível de classes extraído pela Phoenix para a ArchTrace.

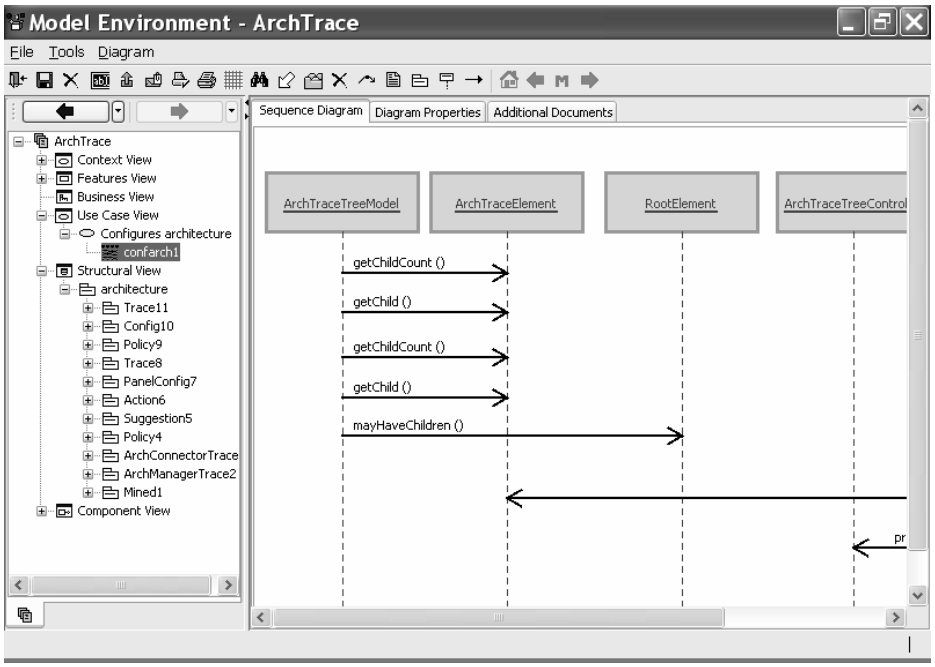

Figura 5.19: Diagrama de seqüência em nível de classe extraído pela Phoenix no ambiente Odyssey.

Como explicado na seção 4.2.2.4, a Phoenix apresenta várias opções de filtro, conforme mostra a sua tela de configuração de filtros da Figura 5.20. O usuário deve selecionar um dos cenários de caso de uso do arquivo de rastros, uma *thread* da qual ler os métodos, ou todas as *threads* (opção *allThreads*), e o método que inicia o diagrama, sendo essa última informação opcional. Se o método não for informado, o diagrama será reconstruído a partir do primeiro método da *thread* informada, ou do primeiro método da *thread* que inicia a execução do cenário.

Os filtros podem envolver ainda a ocorrência do método (*Method Occurrence*), o nível de abstração do diagrama, i.e. em nível de classe ou arquitetural, e o nível de profundidade (*Depth Level*) de mensagens. A ocorrência do método é importante porque o mesmo método na mesma *thread* pode aparecer mais de uma vez no rastro de execução e, nesse caso, é importante especificar a partir de que método exatamente o diagrama deve ser reconstruído. O nível de abstração permite reduzir o tamanho dos diagramas, pois diagramas a nível arquitetural encapsulam mensagens trocadas por objetos de classes do mesmo elemento arquitetural.

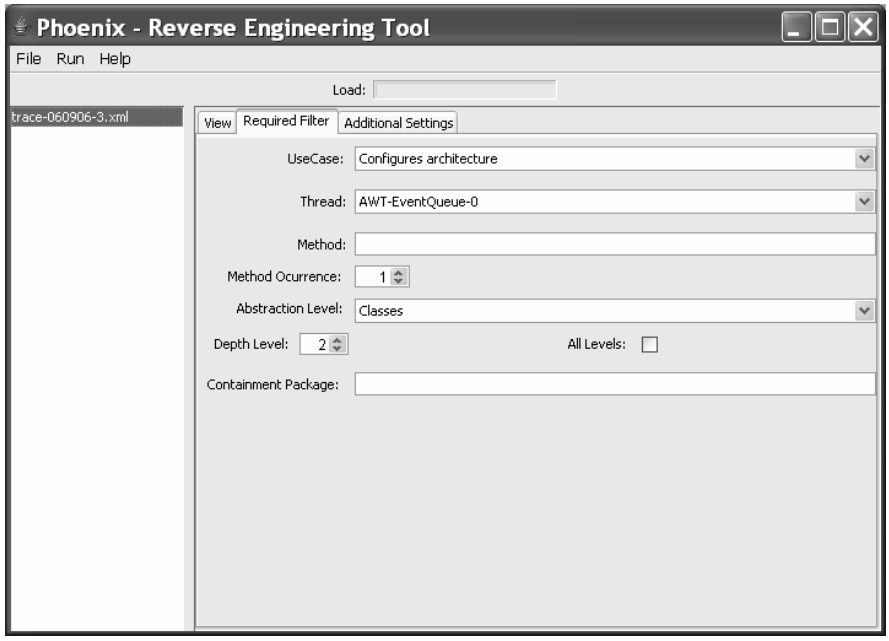

Figura 5.20: Tela de configuração de filtros da ferramenta Phoenix.

Além dessas opções de filtro, algumas opções criadas a partir da análise de rastros de execução para sistemas Java foram criadas. A Figura 5.21 apresenta duas opções de filtro, "*Access from Inner Class Filter*" e "*Superclass Filter*", que reduzem consideravelmente o tamanho de diagramas de seqüência reconstruídos para sistemas em Java. A primeira opção se refere a chamadas de métodos do tipo "*access*$", que ocorre quando uma classe interna, i.e. *inner class* em Java (classe definida dentro de outra classe), acessa uma *outer class* (classe na qual uma *inner class* é definida). A segunda opção é útil porque os construtores de superclasses em Java são chamados quando uma instância é criada, e essas chamadas, em geral, não acrescentam valor ao diagrama de seqüência, uma vez que interessa saber qual é o tipo exato da instância que está sendo criada. Os seus supertipos podem ser descobertos pelo diagrama estático.

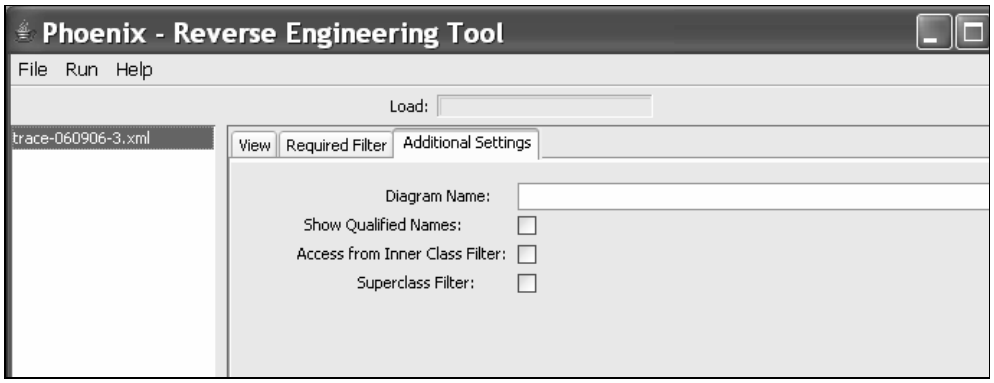

Figura 5.21: Filtros específicos para Java da ferramenta Phoenix.

Durante a reconstrução, se um cenário de caso de uso definido pela *tag* "*Label*" ainda não existe no modelo da aplicação no ambiente Odyssey, um novo caso de uso é criado pela ferramenta Phoenix. Entretanto, os tipos dos objetos (i.e. classes e interfaces) ou os elementos arquiteturais que aparecem no diagrama de seqüência já devem existir no modelo estrutural da aplicação no Odyssey, a fim de manter a consistências entre as visões.

5.3.5 – Avaliação da Arquitetura Recuperada

A abordagem de avaliação de arquiteturas adotada em ArchMine, i.e. **ArqCheck**, conforme já mencionado, se constitui em um processo de inspeção que utiliza como técnica para a detecção de defeitos um *checklist*. Sua execução pode ser manual ou apoiada por ambientes como o apresentado em (KALINOWSKI e TRAVASSOS, 2004). Dessa forma, não foi desenvolvido apoio ferramental à ArqCheck nesta tese.

Entretanto, o documento arquitetural inspecionado é navegado no ambiente Odyssey, para onde são exportadas todas as informações extraídas com o ferramental de apoio à ArchMine. Dessa forma, o Odyssey oferece um apoio à inspeção no sentido de facilitar a visualização e navegação pelos modelos arquiteturais recuperados, permitindo, por exemplo, o fácil acesso aos requisitos funcionais implementados por um elemento arquitetural. As reetruturações sugeridas na arquitetura, através da aplicação de ArqCheck, devem ser realizadas diretamente no ambiente Odyssey.

5.3.6 – Considerações Finais sobre o Ferramental de ArchMine

Esta seção apresentou o ferramental de apoio à execução das atividades de recuperação de arquitetura de ArchMine. Embora o ferramental tenha sido desenvolvido

para analisar sistemas escritos em Java, é possível realizar extensões no ferramental para outras linguagens. A ferramenta de análise estática Ares, por exemplo, possui uma arquitetura flexível a extensões para outras linguagens de programação. Além disso, o Odyssey lê XMI, sendo possível utilizar uma outra ferramenta de análise estática que não a Ares que gere um modelo estático compatível com o XMI do Odyssey. Vale ressaltar que o Odyssey atualmente usa a versão 1.2 da especificação XMI.

Além disso, a TraceMining, atualmente, lê uma base de transações na forma de arquivo serializado, embora seja possível que a sua entrada seja transformada em um arquivo texto, para que ela possa ser usada de forma independente. Ela já gera resultados em forma de arquivo texto, que podem ser lidos em outros ambientes, que não o Odyssey. Vale lembrar que a ferramenta Tracer é independente de ambiente. A Phoenix é a única fortemente acoplada ao Odyssey.

Finalmente, em relação ao ambiente, os resultados gerados por ArchMine no Odyssey podem ser exportados em formato XMI, podendo ser manipulados em outros ambientes de desenvolvimento que leiam XMI compatível com o do Odyssey.

O ferramental de apoio e a sua integração ao Odyssey permitem que ArchMine atenda aos últimos requisitos estabelecidos nesta tese para uma abordagem de recuperação de arquitetura com apoio à reutilização, i.e. apoio ferramental e possibilidade de integração a um ambiente de reutilização. Dessa forma, a Tabela 5.2 apresenta o quadro comparativo final de ArchMine com outras abordagens de recuperação de arquitetura pesquisadas, incluindo os 2 requisitos tratados neste capítulo. Fica, assim, evidente porque houve a necessidade do desenvolvimento de uma nova abordagem de recuperação de arquitetura no contexto desta tese.

Em relação à Tabela 5.2, convém ressaltar que a ArchMine atende parcialmente ao requisito de generalidade por ser voltada à análise de sistemas OO e pelo fato do seu ferramental de apoio ser voltado à análise de sistemas escritos em Java.

Entretanto, algumas adaptações e extensões ainda se fazem necessárias em relação às ferramentas de apoio à ArchMine, a saber:

- **Tracer**: a Tracer deve recuperar a assinatura completa dos métodos, a fim de diferenciar métodos sobrecarregados; deve ainda diferenciar um caso de uso de um cenário de caso de uso nos rastros de execução, a fim de permitir que a Phoenix mapeie cenários adequadamente no Odyssey, uma vez que atualmente a Phoenix mapeia cenários de casos de uso para casos de uso. Vale ressaltar que ambas as soluções já estão

sendo adotadas, no primeiro caso, recuperando a assinatura completa dos métodos através da API do AspectJ e, no segundo caso, criando *tags* específicas no arquivo XML de rastros para caso de uso e cenário.

- **Phoenix**: a Phoenix deve detectar chamadas de métodos em laços e recursivas nos rastros de execução, a fim de reduzir o seu volume. Nesse caso, alguma solução já disponibilizada por outra abordagem de análise dinâmica pode ser adotada (JERDING e RUGABER, 1997; HAMOU-LHADJ e LETHBRIDGE, 2004).

Tabela 5.2: Quadro comparativo entre as abordagens de recuperação de arquitetura e ArchMine.

Abordagens de Recuperação de Arquitetura	Elemento Arquitetural = Conceito Domínio	Nomes de Elementos Arquiteturais com Semântica e Gerados Automáticamente	Rastro para Requisitos Funcionais	Generalidade	Visão Estática e Dinâmica	Abordagem Avaliação	Apoio Ferramental	Ambiente Reutilização
ManSART (HARRIS *et al.*, 1997a)	**Parcial.**	**Não.**	**Não.**	**Não.**	**Não.**	**Não.**	**Sim.**	**Não.**
X-Ray (MENDONÇA e KRAMER, 2001)	**Parcial.**	**Não.**	**Não.**	**Não**	**Não.**	**Não.**	**Sim.**	**Não.**
Pattern-supported (GALL e PINZGER, 2002)	**Parcial.**	**Não.**	**Não.**	**Não.**	**Não.**	**Não.**	**Sim.**	**Não.**
DiscoTect (SCHMERL *et al.*, 2006)	**Parcial.**	**Não.**	**Não.**	**Parcial**	**Não**.	**Sim.**	**Sim.**	**Não.**
Dali (KAZMAN e CARRIÈRE, 1997)	**Sim.**	**Parcial.**	**Não.**	**Parcial**	**Não**	**Sim.**	**Sim.**	**Não.**
High-Level Views for OO Applications (RICHNER e DUCASSE, 1999)	**Sim.**	**Parcial.**	**Não.**	**Parcial**	**Sim.**	**Não.**	**Sim.**	**Não.**
Sincronização entre Visões Estática e Dinâmica (RIVA e RODRIGUEZ, 2002)	**Sim.**	**Parcial.**	**Sim.**	**Parcial**	**Sim.**	**Não.**	**Sim.**	**Não.**
Alborz (SARTIPI e KONTOGIANNIS, 2003)	**Sim.**	**Parcial.**	**Não.**	**Parcial**	**Não.**	**Não.**	**Sim.**	**Não.**
Abordagem direcionado por Casos de Uso (BOJIC e VELASEVIC, 2000)	**Sim.**	**Parcial.**	**Sim.**	**Parcial**	**Não.**	**Não.**	**Sim.**	**Não.**
Focus (DING e MEDVIDOVIC, 2001)	**Parcial.**	**Parcial.**	**Sim.**	**Sim**	**Sim.**	**Não.**	**Não.**	**Não.**
File Names (ANQUETIL e LETHBRIDGE, 1999)	**Parcial.**	**Parcial.**	**Não.**	**Sim**	**Não.**	**Sim.**	**Sim.**	**Não.**
Bunch (MITCHELL e MANCORIDIS, 2006)	**Parcial.**	**Não.**	**Não.**	**Sim**	**Não.**	**Não.**	**Sim.**	**Não.**
ArchMine	**Sim.**	**Parcial.**	**Sim.**	**Parcial**	**Sim.**	**Sim.**	**Sim.**	**Sim.**

- **TraceMining**: a TraceMining deve calcular a solução de compromisso entre suporte e confiança da heurística **H6** e deve ser mais flexível à definição de novas heurísticas. Essa flexibilidade requer a definição de uma infra-estrutura menos amarrada às heurísticas atuais. Além disso, deve ler uma base de transações em um formato menos proprietário que arquivo serializado e definir uma API que permita a sua integração de forma facilitada a outros ambientes.

Finalmente, conforme mencionado, não existe uma abordagem de recuperação de arquitetura que seja adequada a todos os sistemas, com diferentes organizações estruturais. Dessa forma, ainda que ArchMine tenha sido desenvolvida para atender aos requisitos estabelecidos nesta tese, um *plugin* para a exportação dos resultados gerados pela ferramenta Bunch (MITCHELL e MANCORIDIS, 2006) foi desenvolvido para o Odyssey. Esse *plugin* é um módulo do *plugin* de recuperação de arquitetura (ArchRecoveryOdyssey) que incorpora também a TraceMining. Bunch foi escolhida por apoiar uma abordagem de recuperação de arquitetura baseada na análise estática em contrapartida à análise dinâmica proposta por ArchMine. Além disso, seu ferramental se encontra disponível para utilização e extensão.

5.4 – ArchToDSSATool: Comparação de Arquiteturas e Criação de Arquiteturas de Referência

A fim de apoiar a comparação das arquiteturas recuperadas com ArchMine e a criação de arquiteturas de referência de domínio, a ferramenta ArchToDSSATool foi desenvolvida (KÜMMEL, 2007). A ferramenta recebe como entrada arquivos XMI, descrevendo as arquiteturas a serem comparadas, e gera como saída arquivos XMI, descrevendo as arquiteturas de referência de domínio.

A Figura 5.22 apresenta uma visão geral da arquitetura da ArchToDSSATool. A arquitetura é composta de um elemento arquitetural central, o "*phases*", e elementos arquiteturais contendo classes auxiliares, como "*guisupport*", "*components*" e "*utils"*. Quanto aos elementos arquiteturais auxiliares, o pacote denominado "*components*" define uma série de componentes usados pelas classes de interface gráfica, como listas com *checkbox*, árvores com *checkbox*, grupos de *radiobuttons* etc. O pacote

"*guisupport*", por sua vez, implementa partes da interface gráfica de uso geral, que não estão necessariamente atreladas a uma determinada fase da abordagem, mas que são de uso menos genérico que as classes encontradas no pacote "*components*". Por exemplo, o módulo gráfico de configuração da ferramenta fica nesse pacote porque é utilizado para configurações das fases de detecção de equivalências e opcionalidades (*matches*) e de detecção de variabilidades, como mostra a Figura 5.23. Finalmente, no pacote "*utils*", foram implementadas classes utilitárias, como o encapsulamento de repositórios MOF (*Meta Object Facility*) (OMG, 2002a), para a manipulação de arquivos XMI.

O pacote *facade* representa o ponto de acesso à ferramenta, que aciona a execução das três fases da abordagem ArchToDSSA. O pacote *main* contém uma classe para permitir que a ferramenta seja executada de forma independente e o pacote *integration* é responsável pela integração da ferramenta com o Odyssey.

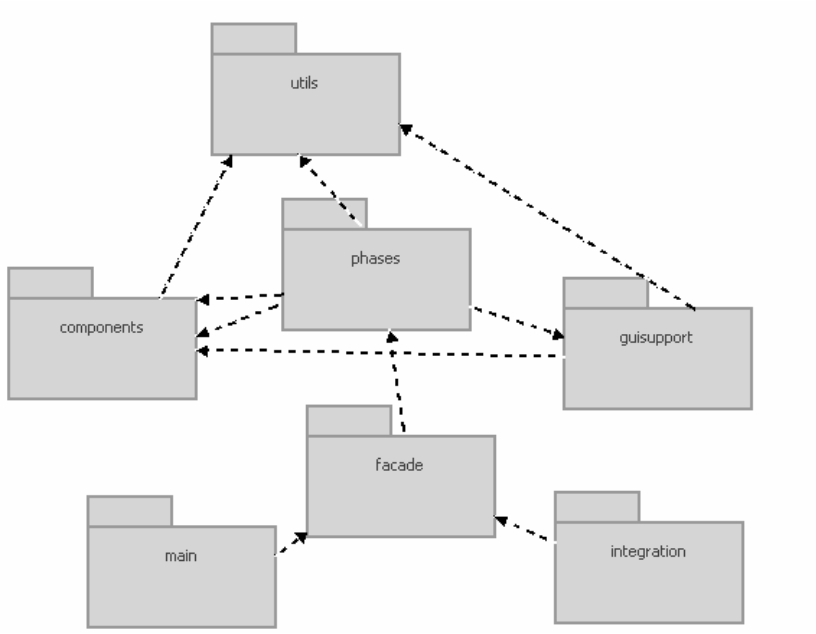

Figura 5.22: Visão geral da arquitetura da ArchToDSSATool.

A arquitetura do elemento arquitetural central da ferramenta foi dividida, de acordo com as funcionalidades da abordagem ArchToDSSA, em 3 elementos arquiteturais distintos, a saber: "*matches*", que cobre a fase de detecção de equivalêsncias e opcionalidades; "*vp*", que cobre a fase de detecção de variabilidades; e "*export*", que cobre a fase de criação de DSSA. As subseções a seguir detalham cada um desses pacotes.

5.4.1 – Pacote *Matches*: Detecção de Opcionalidades

O pacote "*matches*" contém toda a funcionalidade relacionada à detecção de equivalências e opcionalidades, ou seja, inclui tanto as classes de interface gráfica quanto as classes que tratam as regras de negócio dessa fase da abordagem. Como mostra a Figura 5.23, o usuário pode configurar as opções para a detecção de equivalências automática pela ferramenta, que incluem: alimentação automática do dicionário quando o usuário relaciona elementos de nomes diferentes de forma manual (i.e. *Auto Feed Dictionary when Validating Match*); utilização do dicionário, alimentado automaticamente ou de forma manual, quando sugerindo novas equivalências (i.e. *Use Dictionary when Suggesting Matches*); utilização do critério de comparação por *substring* quando sugerindo equivalências (i.e. *Use Substring Comparison when Suggesting Matches*); e comparação apenas de elementos do mesmo tipo, i.e. classes com classes e pacotes com pacotes, quando sugerindo equivalências (i.e. *Compare only Similar Elements when Suggesting Matches*). A ferramenta permite ainda que seja informada a quantidade de arquiteturas onde uma correspondência deve ser encontrada para que ela seja sugerida como uma equivalência.

A Figura 5.24 apresenta a tela principal da ferramenta. Do lado esquerdo, são apresentadas as equivalências, entre elementos das diferentes arquiteturas do domínio de telefonia móvel, utilizado como exemplo, detectadas pela ferramenta e já confirmadas pelo usuário. Uma vez que o usuário alimentou o dicionário com termos sinônimos do domínio e escolheu a comparação por *substrings*, "GerenteEntretenimento" fica equivalente a "GerenteJogos", sendo "Gerente" = "Gerente" e "Entretenimento" sinônimo de "Jogos", como mostra a figura.

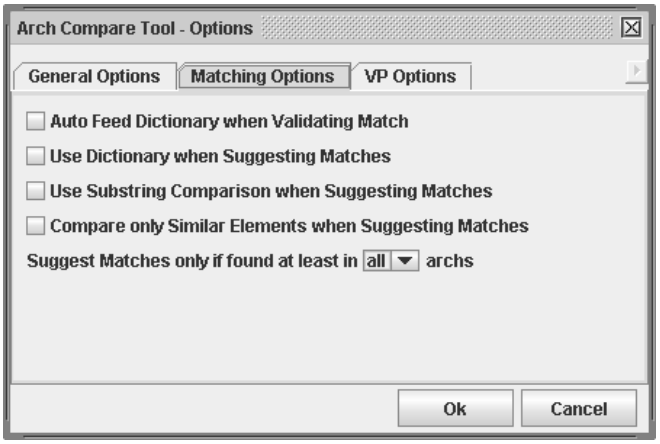

Figura 5.23: Tela de configuração da ArchToDSSATool.

5.4.2 – Pacote *VP*: Detecção de Variabilidades

O pacote "*vp*" contém toda a funcionalidade relacionada à detecção de pontos de variação (i.e. VPs) e suas variantes, ou seja, inclui tanto as classes de interface gráfica quanto as classes que tratam as regras de negócio dessa fase da abordagem. Como mostra a Figura 5.23, existe uma aba no painel de configuração da ferramenta ArchToDSSATool com opções para a configuração da detecção de VPs (i.e. *VP Options*). As opções incluem a determinação do número mínimo de arquiteturas em que uma superclasse ou uma interface deve ser encontrada para que a mesma seja considerada um VP e se tanto superclasse quanto interface devem ser consideradas como candidatos a VPs, conforme sugerido pela abordagem ArchToDSSA.

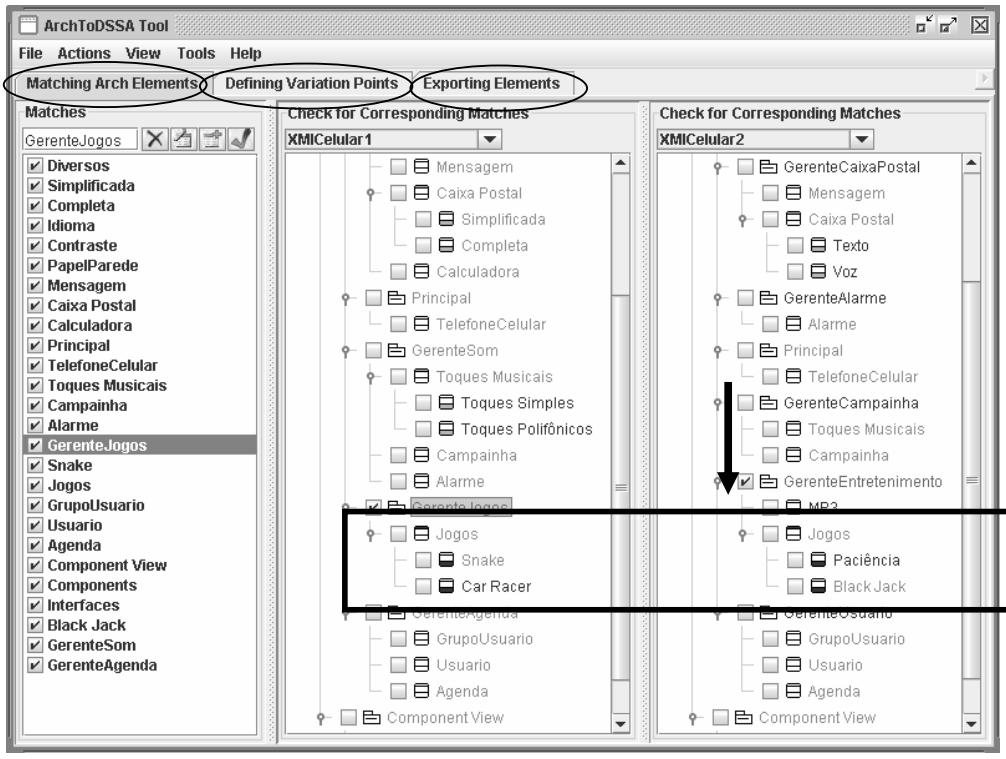

Figura 5.24: Tela principal da ArchToDSSATool.

Dessa forma, uma vez selecionado que superclasse deve ser um candidato a VP se encontrada equivalência em no mínimo 2 arquiteturas, "Jogos" será automaticamente sugerido como um VP pela ferramenta no domínio de telefonia móvel, com as Variantes "*Snake*", "*Car Racer*", "Paciência" e "*Black Jack*", como mostra a Figura 5.24. Embora a comparação nesse exemplo tenha sido entre 3 arquiteturas do domínio, a ferramenta só mostra de 2 em 2 arquiteturas por vez. Nesta fase de definição de pontos de variação, contemplada pelo painel ligado à opção "Defining Variation Points" (Figura 5.24), o

usuário pode aceitar ou rejeitar VPs e variantes sugeridos pela ferramenta, além de selecionar VPs e variantes manualmente.

5.4.3 – Pacote *Export*: Criação de DSSA

O pacote "*export*" contém toda a funcionalidade relacionada à seleção de uma arquitetura base, seleção de elementos opcionais e exportação da DSSA para o formato XMI, ou seja, inclui tanto as classes de interface gráfica quanto as classes que tratam as regras de negócio dessa fase da abordagem. ArchToDSSATool permite a criação de diferentes arquiteturas para o mesmo domínio, através da seleção de diferentes arquiteturas base e elementos opcionais. Entretanto, a exportação é realizada considerando uma arquitetura por vez.

Todos os elementos da arquitetura base, sejam eles opcionais ou mandatórios, são sugeridos para a exportação. O usuário pode desmarcar a seleção de elementos que não sejam de interesse na exportação. Além disso, é possível selecionar elementos opcionais e variantes de outras arquiteturas do conjunto, que não a arquitetura base, sendo que esses vão parcialmente especificados para a arquitetura de referência, ou seja, vão sem os seus relacionamentos originais. A exceção se dá para as variantes com seus VPs, que são ligadas através de herança ou realização de interface, caso contrário não seria possível saber a que VP elas pertencem.

A ArchToDSSATool funciona tanto como *plugin* do Odyssey quanto de forma independente. Dessa forma, as arquiteturas podem ser exportadas para o ambiente Odyssey, que lê arquivo XMI, sendo possível reutilizar as arquiteturas exportadas por ArchToDSSATool na instanciação de aplicações no domínio. A Figura 5.25 apresenta a ArchToDSSATool como *plugin* no ambiente Odyssey.

O modelo exportado para o Odyssey representa uma arquitetura de referência de domínio e fica na *Structural View* do ambiente, como mostra a Figura 5.25. A Figura 5.26 apresenta um diagrama de classes e pacotes, demonstrando parte da arquitetura de referência gerada para o domínio de telefonia móvel. Convém ressaltar que pacotes e classes, embora em níveis de abstração diferentes, estão no mesmo diagrama apenas a título de exemplificação dos elementos gerados. Observa-se que, de acordo com os modelos na Figura 5.24, "Jogos" é um VP, com as variantes "Paciência", "*BlackJack*", "*Car Racer*" e "*Snake*". Como a arquitetura selecionada para exportação foi a primeira, i.e. XMICelular1, "*BlackJack*" e "Paciência" são exportadas como classes não definidas ("*notDefined*"), pois pertencem à arquitetura XMICelular2.

Embora sendo representados como valores chaveados no arquivo XMI, no Odyssey a representação das opcionalidades, variabilidades e da propriedade "não definido" é visualmente apresentada como estereótipos.

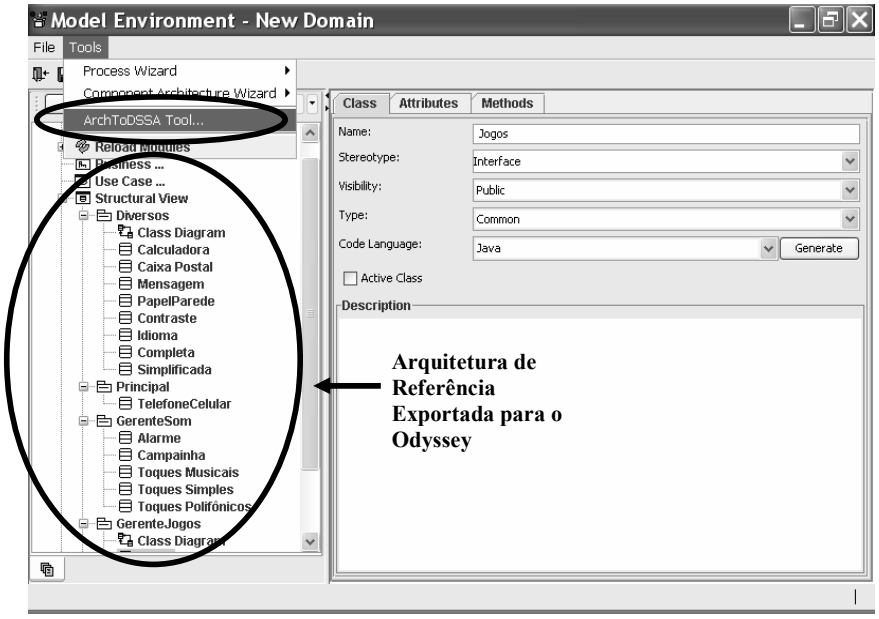

Figura 5.25: ArchToDSSATool como plugin no ambiente Odyssey.

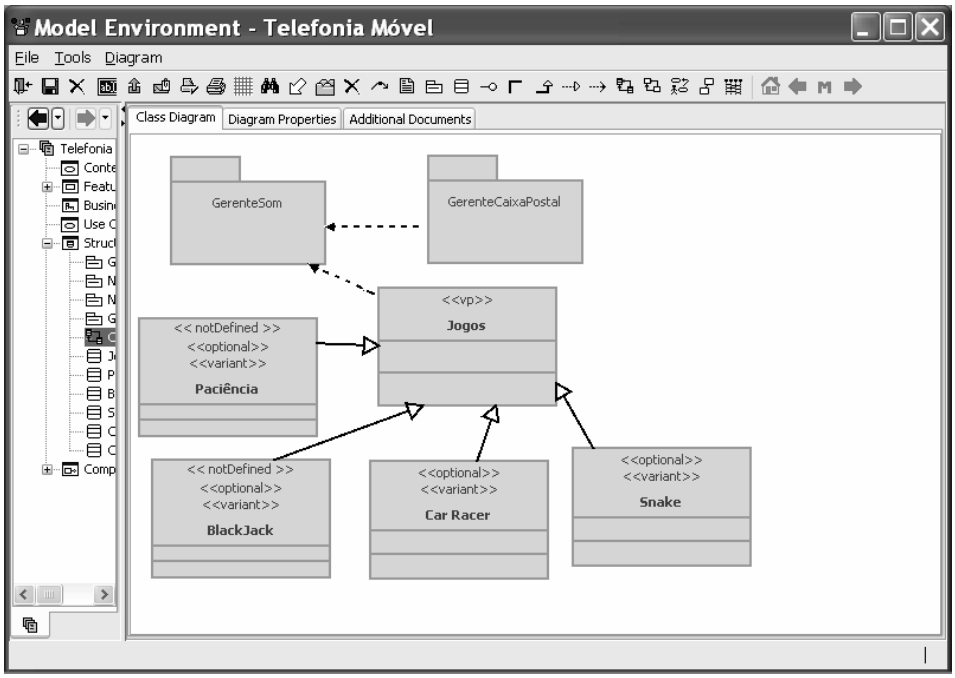

Figura 5.26: Parte da arquitetura de referência para o domínio de telefonia móvel.

A Figura 5.27 apresenta o modelo de características mapeado a partir do modelo arquitetural apresentado na Figura 5.26. Conforme pode ser observado, "Jogos" é um VP, com as suas variantes opcionais. Tanto os pacotes, i.e. elementos arquiteturais,

quanto as classes, são mapeados para características conceituais. Sendo que, quando uma classe faz parte de um pacote, isso é representado através de agregação no diagrama de características, como mostra a relação entre "GerenteSom" e "Campanhia".

O detalhamento das heurísticas de mapeamento do modelo de classes para o modelo de características no Odyssey pode ser obtido no anexo F.

5.4.4 – Considerações Finais sobre ArchToDSSATool

Esta seção apresentou a ferramenta de apoio à abordagem de comparação de arquiteturas e geração de uma arquitetura de referência de domínio parcialmente especificada, a ArchToDSSATool. É importante ressaltar que as arquiteturas de referência geradas por ArchToDSSATool ainda precisam ser refinadas pelo Engenheiro de Domínio. Ele deve especificar os elementos marcados como "não definidos", revisando seus possíveis métodos, atributos e relacionamentos.

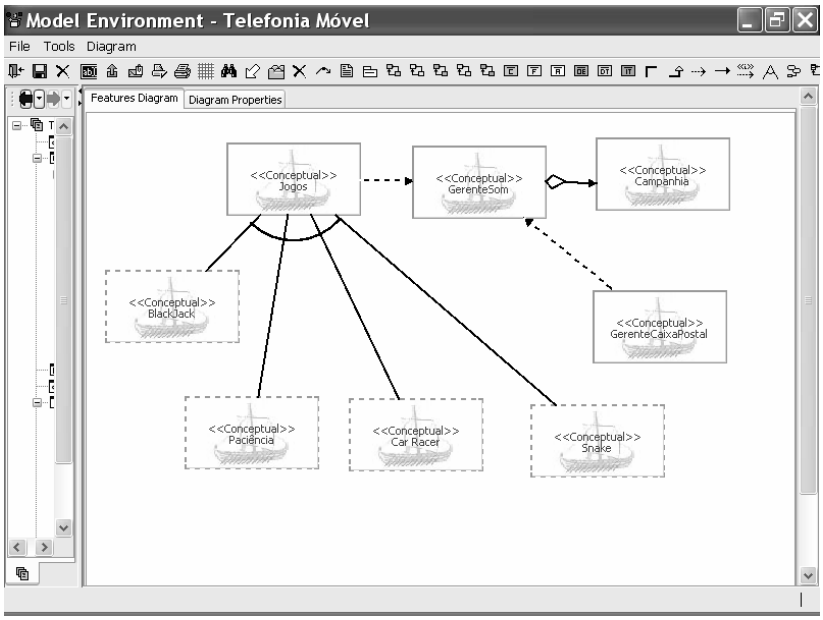

Figura 5.27: Modelo de características mapeado a partir do modelo arquitetural de telefonia móvel.

A infra-estrutura disponibilizada pelo Odyssey permite que mapeamentos sejam gerados do modelo arquitetural para o modelo de características. Dessa forma, é possível que o modelo arquitetural gerado por ArchToDSSATool seja recortado e sirva de base para a instanciação de novas aplicações do domínio.

É importante ressaltar que, além do modelo arquitetural estático, LegaToDSSA prevê a reconstrução do modelo de casos de uso e dinâmico das aplicações. Dessa forma, o ferramental disponibilizado através de ArchToDSSATool deve ser estendido para efetuar comparações também nesses modelos.

ArchToDSSATool apresenta flexibilidade em relação a ambiente de desenvolvimento, uma vez que lê e exporta arquivos XMI. Porém, os ambientes precisam ser compatíveis com a versão do XMI e UML adotadas pela ferramenta, i.e. a versão 1.2 da especificação XMI e a versão 1.4 do metamodelo da UML, as mesmas adotadas pelo Odyssey. Tanto o Odyssey quanto a ArchToDSSATool utilizam o repositório MOF MDR (*Metadata Repository*) (MATULA, 2007), que é compatível com essas versões.

5.5 – Considerações Finais

Este capítulo apresentou o ferramental que suporta a execução das atividades da abordagem de apoio à criação de arquiteturas de referência de domínio a partir de sistema legados, proposta neste tese, a LegaToDSSA. As ferramentas desenvolvidas representam protótipos acadêmicos que visam ser constantemente refinados à medida que novos estudos sejam realizados sobre as abordagens e sobre o próprio ferramental. Porém, vale ressaltar que o ferramental de apoio à abordagem de recuperação de arquitetura ArchMine já foi avaliado através de estudos de caso reais, tendo sido refinado ao longo desse processo. Os estudos demonstraram a viabilidade do ferramental de apoio à ArchMine.

A ferramenta de apoio à execução das atividades de comparação de arquiteturas e geração de arquiteturas de referência, ArchToDSSATool, será avaliada no contexto do trabalho de mestrado de KÜMMEL (2007), podendo sofrer ainda alguns refinamentos.

O ferramental desenvolvido pode ser utilizado no ambiente de reutilização Odyssey, apoiando a modelagem de domínios e a sua instanciação em diferentes aplicações. A ferramenta de comparação de arquiteturas e geração de arquiteturas de referência está baseada na especificação XMI, apresentando certa generalidade em relação a ambiente de desenvolvimento. O ferramental de ArchMine, embora ainda não apresente essa generalidade, apresenta flexibilidade para adaptação a outros ambientes. A ferramenta de análise estática Ares, por exemplo, apresenta uma arquitetura preparada para extensão para diferentes linguagens de programação. A ferramenta de mineração TraceMining, atualmente, lê um arquivo serializado, em formato próprio, e gera arquivo texto, podendo ser utilizada em outros contextos, que não o Odyssey. A sua entrada de dados também está sendo convertida para um formato mais flexível.

O próximo capítulo apresenta os estudos experimentais realizados para avaliar a viabilidade da abordagem e ferramental de ArchMine.

Capítulo 6 – Avaliação da Abordagem de Recuperação de Arquitetura ArchMine

6.1 – Introdução

Este capítulo apresenta os estudos experimentais que foram realizados no contexto desta tese. Antes mesmo de descrever esses estudos, é necessário ressaltar a importância da realização de estudos dessa natureza para a avaliação de um novo método, técnica, processo ou ferramenta na Engenharia de Software. Retomando o histórico da área de Engenharia de Software, é comum observar afirmativas feitas na literatura sem a devida comprovação da teoria que as sustenta ou a transferência de novos métodos ou técnicas imaturos da academia para a indústria sem a devida comprovação da sua eficácia e eficiência em relação aos existentes. Segundo TRAVASSOS *et al.* (2002), a experimentação representa o centro do processo científico, consistindo na única forma de se verificar uma nova teoria. Ela oferece um modo sistemático, disciplinado, computável e controlado para a avaliação de novos métodos, técnicas, linguagens e ferramentas.

Considerando que a Engenharia de Software pode ser vista tanto como engenharia, no contexto da produção de um produto através de um processo disciplinado e economicamente efetivo, quanto como ciência, no sentido da necessidade de melhoria contínua da qualidade do processo e do produto, verifica-se que a realização de estudos experimentais nessa área é essencial para o seu amadurecimento. Segundo AMARAL (2003), novos métodos, técnicas, linguagens e ferramentas não deveriam apenas ser sugeridos, publicados ou apresentados para a venda sem experimentação e avaliação.

De acordo com WOHLIN *et al.* (2000), dependendo do propósito da avaliação e das condições para o estudo experimental (ex: disponibilidade de recursos), há três diferentes estratégias de estudo experimental que podem ser adotadas:

- **Investigação (*survey*)**: representa uma investigação realizada em retrospectiva, quando, por exemplo, uma técnica ou ferramenta foi utilizada por um tempo e se deseja avaliá-la sob algum aspecto. As

principais ferramentas utilizadas para apoiar a investigação são entrevistas e questionários.

- **Estudos de Caso**: são utilizados para monitorar projetos, atividades ou atribuições. Dados são coletados para um propósito específico através do estudo e análises estatísticas podem ser conduzidas com base nos dados coletados. Normalmente, visa rastrear um atributo específico ou estabelecer relacionamentos entre diferentes atributos. O nível de controle é menor do que em um experimento.
- **Experimentos**: são normalmente conduzidos em laboratório, o que provê um alto nível de controle sobre o processo e os fatores ou variáveis que afetam o estudo. O objetivo do experimento é manipular uma ou mais variáveis, mantendo as demais sob controle. Experimentos são apropriados para confirmar teorias, avaliar a predição dos modelos ou validar medidas. Apresenta maior facilidade de repetição.

Conforme pode ser observado pelas definições, as características principais que diferenciam essas estratégias são: o controle de execução, o controle de medição, o custo de investigação e a facilidade de repetição (TRAVASSOS *et al.*, 2002).

Além dessa classificação dos tipos de estudo, é importante ressaltar também os objetivos de um estudo de experimentação. Segundo TRAVASSOS *et al.* (2002), os objetivos relacionados à execução de experimentos em Engenharia de Software são a caracterização, avaliação, previsão, controle e melhoria a respeito de produtos, processos, recursos, modelos, teorias entre outros. A importância e o esforço aumentam de um experimento com o objetivo de "caracterização" a um experimento com o objetivo de "melhoria".

Estudos de viabilidade costumam objetivar a caracterização de uma tecnologia, verificando se ela realmente faz o que se propõe a fazer e se é viável continuar a despender recursos para desenvolvê-la. SHULL *et al.* (2001) destacam que revisões nestas questões provocam as maiores alterações na tecnologia, por isso elas devem ser tratadas no início do processo de avaliação. Dessa forma, estudos desse tipo costumam ser os primeiros a serem conduzidos em um processo de avaliação de uma nova tecnologia ou abordagem.

Diante do contexto exposto, quatro estudos de viabilidade foram conduzidos no decorrer desta tese. Eles se enquadram na categoria de estudos de caso, estratégia de estudo experimental selecionada devido a uma série de fatores, como: escassez de

tempo e recursos (ex: participantes) para o planejamento e execução dos estudos; tipo de avaliação que se deseja realizar (ou seja, a viabilidade e eficácia da abordagem de recuperação de arquitetura proposta); impossibilidade de controle total sobre as variáveis dos estudos (ex: perfil dos participantes e arquitetura esperada, a qual é avaliada por especialistas das aplicações).

Em relação aos seus objetivos, os estudos realizados podem ser apresentados na seguinte ordem: (1) o primeiro estudo representa um estudo de viabilidade que visa caracterizar a abordagem proposta de recuperação de arquitetura, ArchMine, quanto à sua aplicabilidade em recuperar modelos arquiteturais compreensíveis para sistemas legados OO; (2) a fim de avaliar a viabilidade de ArchMine, o segundo estudo representa um estudo de viabilidade que visa caracterizar a eficácia de ArchMine em comparação com uma outra abordagem de recuperação de arquitetura; (3) visto que o processo de avaliação de arquitetura em ArchMine vinha sendo realizado de forma *ad-hoc* e exigindo grande esforço do especialista da aplicação, o objetivo deste terceiro estudo de viabilidade é caracterizar a aplicabilidade da abordagem proposta para a avaliação das arquiteturas recuperadas com ArchMine; (4) o objetivo deste último estudo de caso é avaliar se a etapa de avaliação de ArchMine de fato pode ser melhorada, em relação ao esforço de avaliação e qualidade final da arquitetura recuperada, com a utilização da abordagem avaliada no terceiro estudo de caso.

Convém ressaltar que foram realizados estudos apenas para a avaliação da abordagem de recuperação de arquitetura ArchMine. A avaliação da abordagem de comparação de arquiteturas e geração de arquiteturas de referência de domínio, i.e. ArchToDSSA, é objeto do trabalho de mestrado de KÜMMEL (2007).

Em função do contexto exposto, o capítulo está organizado da seguinte forma: a seção 6.2 apresenta o primeiro estudo de viabilidade de ArchMine; a seção 6.3 apresenta o estudo de verificação da eficácia de ArchMine em comparação com uma outra abordagem de recuperação de arquitetura; a seção 6.4 apresenta o estudo de viabilidade sobre a abordagem para a avaliação de arquitetura; a seção 6.5 apresenta o último estudo de viabilidade de ArchMine, tendo como foco a caracterização da viabilidade em aplicar a abordagem de avaliação de arquitetura para a avaliação das arquiteturas recuperadas com ArchMine; e, por fim, a seção 6.6 apresenta as considerações finais acerca dos estudos experimentais realizados no contexto desta tese.

6.2 – Estudo de Viabilidade de ArchMine

Neste primeiro estudo de caso, a viabilidade da abordagem de recuperação de arquitetura ArchMine foi observada através da recuperação da arquitetura de 3 aplicações em ambiente acadêmico. A escolha do ambiente acadêmico para a realização dos estudos iniciais se deve ao fato de que a inserção de uma tecnologia em um ambiente industrial envolve altos custos e riscos (ex: disponibilidade e horas de trabalho dos profissionais, impacto nos projetos em andamento etc.) que devem ser minimizados para que ela possa ser aplicada com sucesso (BARCELOS, 2006). Assim, para que uma tecnologia seja inserida em ambiente industrial, o ideal é que ela já tenha sido avaliada e amadurecida em ambiente acadêmico.

Dessa forma, no contexto da avaliação da abordagem de recuperação de arquitetura proposta nesta tese, foi realizado um estudo inicial de viabilidade, visando verificar a sua aplicabilidade na reconstrução de um modelo arquitetural, onde elementos arquiteturais representem conceitos do domínio.

6.2.1 – Planejamento do Estudo

Objetivo Global

O propósito desse estudo, portanto, é avaliar a viabilidade da abordagem de recuperação de arquitetura de sistemas legados orientados a objetos escritos em Java, ArchMine, na recuperação de um modelo arquitetural, onde os elementos arquiteturais representam conceitos do domínio e implementam um conjunto de serviços coesos.

Objetivo do Estudo

Analisar a abordagem de recuperação de arquitetura ArchMine

Com o propósito de caracterizar

Com respeito à viabilidade em recuperar um modelo arquitetural compreensível para a aplicação, onde os elementos arquiteturais representem conceitos do domínio e implementem um conjunto de serviços coesos

Do ponto de vista de engenheiros de software

No contexto da recuperação da arquitetura de aplicações acadêmicas, para as quais um especialista se encontra disponível para avaliar a arquitetura recuperada.

Questões

A fim de avaliar os resultados produzidos pela abordagem ArchMine, o paradigma GQM (*Goal Question Metric*) de BASILI *et al.* (1994) é adotado e métricas tradicionais de recuperação da informação, i.e. precisão e revocação (BAEZA-YATES e RIBEIRO-NETO, 1999), são utilizadas. Precisão (*Precision*) pode ser definida como a fração dos elementos recuperados que são relevantes. Revocação (*Recall*), por outro lado, representa a fração dos elementos relevantes que são recuperados.

Q1: **A arquitetura recuperada é precisa?**

M1:

$$\textbf{PrecisãoArquitetura} = \frac{|\text{RelevantesEls} \cap \text{RecuperadosEls}|}{|\text{RecuperadosEls}|}$$

A fim de calcular a precisão da arquitetura recuperada, os seguintes valores são considerados:

RecuperadosEls = todos os elementos arquiteturais recuperados pela abordagem ArchMine.

RelevantesEls ∩ RecuperadosEls = elementos arquiteturais recuperados pela abordagem ArchMine que representam um conceito ou um conjunto de serviços coesos do domínio, sob o ponto de vista do especialista.

Q2: **Os elementos arquiteturais recuperados são precisos, i.e. estão corretos?**

M2:

$$\textbf{PrecisãoElemento} = \frac{|\text{RelevantesClasses} \cap \text{RecuperadosClasses}|}{|\text{RecuperadosClasses}|}$$

Neste caso, o cálculo de precisão é realizado para cada um dos elementos arquiteturais recuperados, considerando-se a sua estrutura interna, como se segue:

RecuperadosClasses = todos os módulos internos ou classes recuperados no elemento arquitetural.

RelevantesClasses ∩ RecuperadosClasses = módulos internos ou classes recuperados no elemento arquitetural e que de fato devem compor este elemento arquitetural, em função do conceito que ele representa, sob o ponto de vista do especialista.

Q3: **A arquitetura recuperada tem uma boa taxa de revocação, i.e. apresenta uma boa cobertura em relação aos elementos arquiteturais que deveriam estar presentes?**

M3:

$$\textbf{RevocaçãoArquitetura} = \frac{|\text{RelevantesEls} \cap \text{RecuperadosEls}|}{|\text{RelevantesEls}|}$$

A fim de calcular a revocação da arquitetura recuperada, os seguintes valores são considerados:

RelevantesEls = todos os elementos arquiteturais que devem estar presentes na arquitetura sob o ponto de vista do especialista.

RelevantesEls ∩ RecuperadosEls = elementos arquiteturais recuperados pela abordagem ArchMine e que fazem parte do conjunto dos elementos arquiteturais que devem estar presentes na arquitetura, sob o ponto de vista do especialista.

Q4: **Os elementos arquiteturais recuperados têm uma boa taxa de revocação, i.e. apresentam uma boa cobertura em relação às classes que deveriam estar presentes?**

M4:

$$\textbf{RevocaçãoElemento} = \frac{|\text{RelevantesClasses} \cap \text{RecuperadosClasses}|}{|\text{RelevantesClasses}|}$$

RelevantesClasses = todos os módulos internos ou classes que devem compor este elemento arquitetural, em função do conceito que ele representa, sob o ponto de vista do especialista.

RelevantesClasses ∩ RecuperadosClasses = módulos internos recuperados no elemento arquitetural e que fazem parte do conjunto dos módulos que devem compor este elemento arquitetural, em função do conceito que ele representa, sob o ponto de vista do especialista.

A fim de verificar se os valores de precisão e revocação obtidos para os elementos arquiteturais recuperados são razoáveis, os valores obtidos são comparados com valores da literatura de ER. Essa comparação deve ser realizada através da média harmônica (BAEZA-YATES e RIBEIRO-NETO, 1999), que estabelece uma relação de compromisso entre os valores de precisão e revocação. Quanto mais o valor da média

harmônica se aproxima de 1, melhor é essa relação de compromisso. A média harmônica é calculada da seguinte forma:

$$\textbf{MédiaHarmônica} = \frac{2}{\frac{1}{\text{Revocação}} + \frac{1}{\text{Precisão}}}$$

Q5: **Os elementos arquiteturais recuperados apóiam a compreensão da aplicação?**

M5: Número de especialistas que respondeu que os elementos arquiteturais recuperados e seus relacionamentos representam um possível modelo arquitetural da aplicação.

Seleção do Contexto

O estudo é realizado em ambiente acadêmico e a recuperação da arquitetura é executada para três aplicações acadêmicas de tamanhos e complexidades diferenciados. As aplicações foram selecionadas levando-se em conta o fato de terem sido desenvolvidas em Java e o fato de haver um especialista disponível para apoiar o estudo.

A primeira das aplicações é uma ferramenta de evolução de rastros entre elementos arquiteturais e elementos do código fonte, desenvolvida em parceria entre a COPPE/UFRJ e a Universidade da Califórnia em Irvine (UCI), denominada ArchTrace (MURTA *et al.*, 2006). A segunda é um jogo baseado em simulação para treinamento de gerentes de projeto, denominada TIM (*The Incredible Manager*) (DANTAS *et al.*, 2006). A terceira é um ferramenta para a elaboração, verificação e validação de restrições OCL (*Object Constraint Language*) anexadas a modelos UML (CORREA e WERNER, 2006), i.e. a Odyssey-PSW. A Tabela 6.1 resume as características das aplicações sob estudo.

Tabela 6.1: Aplicações objetos de estudo do primeiro estudo de viabilidade.

Aplicação	Classes	Ligações entre Classes	LOC	Estilos Arquiteturais	Cenários de Caso de Uso
ArchTrace	80	308	4 695	Modelo-Visão-Controlador (*Model-View-Controller* -MVC)	14
TIM	51	168	5724	MVC	15
Odyssey-PSW	596	1963	65 273	MVC, Camadas, Dutos e Filtros	55

Seleção dos Participantes

O próprio pesquisador aplicou a sua abordagem de recuperação de arquitetura sobre três aplicações das quais não tinha conhecimento. Como os especialistas conhecem a arquitetura da aplicação, eles não poderiam aplicar a abordagem de recuperação de arquitetura, o que representaria um viés para o experimento. Os especialistas participaram do estudo para apoiar o pesquisador na preparação dos dados e para realizar a avaliação da arquitetura recuperada.

Definição das Hipóteses

Hipótese Nula (H0): Não é possível recuperar um modelo arquitetural compreensível, onde os elementos arquiteturais implementem um conjunto de serviços coesos que representem conceitos do domínio, para aplicações orientadas a objeto escritas em Java, aplicando a abordagem de recuperação de arquitetura baseada na mineração de rastros de execução ArchMine.

Variáveis

Durante o planejamento de um estudo experimental, as variáveis relacionadas devem ser definidas. Variáveis independentes são aquelas que podem ser controladas e manipuladas no estudo experimental e que têm impacto nos valores das variáveis dependentes, representando a causa que afeta o resultado do processo de experimentação (WOHLIN *et al.*, 2000). Algumas variáveis independentes têm o seu valor fixado e outras são avaliadas quanto ao seu impacto no resultado, as quais neste caso são chamadas de "fatores" e seus valores de "tratamentos". As variáveis independentes para esse estudo são apresentadas na Tabela 6.2.

Tabela 6.2: Variáveis independentes do primeiro estudo de viabilidade.

Variável	Valores	Descrição
APL	ArchTrace, TIM e Odyssey-PSW	Aplicações para as quais as arquiteturas são recuperadas.
CEN	Cenários de casos de uso das aplicações	Cenários especificados e monitorados para as aplicações ArchTrace, TIM e Odyssey-PSW.
CNF	Real	Valor de confiança mínima aplicado na mineração dos rastros de execução.
SUP	Real	Valor de suporte mínimo aplicado na mineração dos rastros de execução.
ANT	Antecedentes	Classes selecionadas como antecedentes para a mineração.
ARC	ArchMine	Abordagem de recuperação de arquitetura utilizada.

Nesse estudo, o único tratamento é a abordagem de recuperação de arquitetura ArchMine. O valor de confiança mínima para a mineração foi ajustado com especialistas, chegando-se a um valor razoável de 60%. Esse valor não foi variado no estudo para se analisar os impactos dessa variação. O suporte mínimo adotado foi de 0%, uma vez que se desejava agrupar todas as classes monitoradas em elementos arquiteturais durante os ciclos de mineração. Esse valor também não foi variado. Os cenários de casos de uso da aplicação foram definidos junto aos especialistas e os antecedentes para a mineração selecionados aleatoriamente.

O efeito do(s) tratamento(s) é medido através das variáveis dependentes (WOHLIN *et al.*, 2000), que representam o resultado do estudo. As variáveis dependentes que apresentam o efeito da aplicação do tratamento são apresentadas na Tabela 6.3.

As aplicações para as quais a arquitetura é recuperada representam ao mesmo tempo objetos do estudo, sobre os quais o tratamento é aplicado, e variáveis independentes em função do impacto que as suas características podem causar no resultado.

Tabela 6.3: Variáveis dependentes do primeiro estudo de viabilidade.

Variável	Valores	Descrição
PRCE	Real	Medida de precisão dos elementos arquiteturais recuperados para as aplicações ArchTrace, TIM e Odyssey-PSW, aplicando a abordagem ArchMine.
PRCA	Real	Medida de precisão da arquitetura recuperada para as aplicações ArchTrace, TIM e Odyssey-PSW, aplicando a abordagem ArchMine.
RECE	Real	Medida de revocação dos elementos arquiteturais recuperados para as aplicações ArchTrace, TIM e Odyssey-PSW, aplicando a abordagem ArchMine.
RECA	Real	Medida de revocação da arquitetura recuperada para as aplicações ArchTrace, TIM e Odyssey-PSW, aplicando a abordagem ArchMine.
MHA	Real	Medida de compromisso entre a precisão e a revocação dos elementos arquiteturais recuperados para as aplicações ArchTrace, TIM e Odyssey-PSW, aplicando a abordagem ArchMine.
NCP	Inteiro	Número de especialistas que respondeu que os elementos arquiteturais recuperados e seus relacionamentos representam um possível modelo arquitetural da aplicação.
TMP	Real	Tempo de recuperação da arquitetura.

Validade do Estudo

As seguintes ameaças à validade do estudo foram identificadas:

Validade Interna

Refere-se ao risco de um outro fator, que não o tratamento (i.e. a aplicação da abordagem ArchMine), ter influenciado nos resultados do estudo.

Esforço na Avaliação: o fato da abordagem envolver um grande número de checagens a serem realizadas na arquitetura leva a um desgaste dos especialistas quando o documento arquitetural é extenso, o que acaba prejudicando a sua avaliação. As arquiteturas avaliadas não eram extensas, com exceção da arquitetura da aplicação Odyssey-PSW, para a qual o especialista respondeu algumas questões realizando marcações na própria arquitetura.

Viés da Arquitetura Recuperada: se o especialista não tiver um minucioso conhecimento da aplicação, ele pode ser levado a achar que a arquitetura recuperada está correta. Por isso, os especialistas convidados foram os próprios autores das aplicações em questão.

Viés da Proximidade dos Participantes: os participantes do estudo são colegas de trabalho, do mesmo grupo de pesquisa, do pesquisador. Dessa forma, pode haver um viés na sua avaliação da arquitetura recuperada, visando agradar o pesquisador. Porém, através da análise dos resultados obtidos neste estudo, em comparação com dados da literatura, é possível obter indícios de que esse risco possivelmente não tenha se manifestado.

Validade de Construção

Refere-se à ameaça de que o projeto do estudo de fato reflita os seus objetivos, ou seja, o tratamento deve refletir de fato a causa do estudo e as saídas (i.e. variáveis dependentes) devem refletir o seu efeito.

Não foram identificadas ameaças a esse respeito.

Validade Externa

Está relacionada à ameaça dos resultados do estudo não serem generalizáveis a outros contextos, que não aquele em que o estudo foi realizado.

Viés da origem das aplicações: o estudo experimental utiliza aplicações acadêmicas desenvolvidas por um mesmo grupo de pesquisa, i.e. o grupo de reutilização da COPPE/UFRJ, utilizando estilos arquiteturais e padrões de projeto e de programação similares. Portanto, o estudo pode não ser representativo da teoria sob a qual se sustenta,

havendo a necessidade de avaliar a abordagem em aplicações desenvolvidas em outros contextos.

Interação entre Ambiente e Tratamento: o fato das aplicações utilizadas no estudo terem sido desenvolvidas no mesmo ambiente universitário e pelo mesmo grupo de pesquisa que a abordagem ArchMine pode representar uma ameaça à generalização dos resultados do estudo para um contexto industrial.

Validade de Conclusão

Refere-se ao risco em traçar conclusões corretas sobre as relações entre o tratamento e os resultados do estudo experimental.

Comparação com Dados da Literatura: a análise de viabilidade de ArchMine a partir da comparação com valores de precisão e revocação da literatura é um risco, pois a comparação não leva em conta o tamanho, a complexidade e o domínio das aplicações testadas para as diferentes abordagens. Há a necessidade de se realizar estudos aplicando ArchMine e outras abordagens de recuperação de arquitetura sobre as mesmas aplicações, a fim de se traçar conclusões mais significativas sobre a sua eficácia.

6.2.2 – Execução do Estudo Experimental

Preparação

Nesta etapa, os especialistas das aplicações preencheram o formulário de consentimento (Anexo A/A.1) para apoio no estudo. Eles apóiam tanto a fase de preparação dos dados quanto a avaliação da arquitetura recuperada. A fim de permitir a avaliação das arquiteturas recuperadas, os especialistas receberam instruções (Anexo A/A.4) para o preenchimento dos formulários de avaliação da arquitetura recuperada (Anexo A/A.2).

Durante a preparação dos dados, o pesquisador executou as aplicações junto aos especialistas, especificando os seus cenários de casos de uso. A Tabela 6.1 lista a quantidade de cenários de casos de uso identificada para cada uma das aplicações.

Os cenários de caso de uso foram definidos avaliando-se itens do menu principal e menus pop-up das aplicações. Opções de menu semanticamente equivalentes, como Save e Save As, originaram apenas um cenário de caso de uso. Cenários de exceção não foram considerados, embora cenários disparados de formas não convencionais (ex: clique em um link na interface) tenham sido ressaltados pelos especialistas.

Uma vez definidos os cenários de casos de uso, os rastros de execução das aplicações foram coletados com o apoio da ferramenta Tracer, descrita na seção 5.3.3. Além dos rastros de execução, o modelo estático das aplicações também representa um artefato de entrada para a recuperação da arquitetura. Dessa forma, foi feita a Engenharia Reversa (ER) estática de cada uma das aplicações no ambiente OdysseyLight, utilizando a sua ferramenta de ER, a Ares, descrita na seção 5.3.1.

Operação

Uma vez preparados os artefatos de entrada para a recuperação da arquitetura, a mineração dos rastros de execução para apoiar a recuperação da arquitetura de software foi realizada pelo pesquisador. Essa etapa foi realizada com o apoio da ferramenta TraceMining, descrita na seção 5.3.4, funcionando como um *plugin* para o ambiente OdysseyLight. A mineração dos rastros de execução e o agrupamento de classes em elementos arquiteturais são atividades semi-automatizadas, requerendo intervenção manual. Entretanto, com base nas heurísticas definidas no processo ArchMine, apresentadas na seção 4.2.5.2, o pesquisador pôde se guiar nesta etapa, gerando o modelo arquitetural resultante no ambiente OdysseyLight.

Com base na heurística H1, o pesquisador utilizou um valor de suporte baixo, adotando um valor de 0%, uma vez que o objetivo era que todas ou a maior parte das classes monitoradas fossem agrupadas em elementos arquiteturais durante os ciclos de mineração. A confiança mínima adotada foi de 60%, valor esse que foi calibrado com os participantes do estudo, i.e. especialistas das aplicações. Os antecedentes foram selecionados dos maiores para os menores valores de suporte, conforme sugere a heurística H3, sendo iniciada a seleção no suporte máximo de 100%. Esse suporte foi sendo reduzido em 10% até que não houvesse mais sugestão de antecedente para minerar ou que não houvesse sugestão de grupos de classes ou elementos arquiteturais a serem formados.

A cada ciclo de mineração, várias classes eram retornadas nas regras de associação dos antecedentes. A ferramenta TraceMining calcula as interseções entre classes resultantes nas regras de diferentes antecedentes. Dessa forma, de acordo com a heurística H5, primeiramente eram formados grupos a partir das interseções e, caso restassem classes não agrupadas, elas eram agrupadas junto com as outras classes que apareciam em regras do mesmo antecedente. As classes nas interseções eram filtradas desses grupos. Vale lembrar que se havia interseção entre 3 antecedentes e entre 2 antecedentes que representassem um subconjunto desses 3, a prioridade era para a

formação de grupos das interseções entre o maior número possível de antecedentes, ou seja, entre os 3 antecedentes, nesse caso. Porém, o pesquisador evitou compor grupos muito pequenos, ou seja, com 2 ou 3 elementos, combinando, em alguns casos, resultados das interseções com antecedentes em comum.

Grupos de classes já formados eram sempre eliminados dos ciclos de mineração subseqüentes, conforme sugere a heurística H4. De vez em quando, o pesquisador perguntava à ferramenta TraceMining quantas classes ainda não tinham sido agrupadas e, quando havia um percentual baixo de classes não agrupadas (entre 10 e 20%), o pesquisador aplicava a heurística H6, referente ao agrupamento de classes não agrupadas.

Os elementos arquiteturais recuperados foram apresentados aos especialistas para que os mesmos avaliassem as arquiteturas recuperadas e preenchessem os formulários de avaliação. As arquiteturas foram geradas tanto no OdysseyLight quanto em documento texto.

6.2.3 – Análise dos Resultados Obtidos

Durante o planejamento desse estudo, foi definido como objetivo principal avaliar a viabilidade da abordagem de recuperação de arquitetura ArchMine. Essa viabilidade, conforme os questionamentos estabelecidos, é avaliada através da análise dos valores de precisão e revocação da arquitetura recuperada, comparando-se esses valores com dados da literatura de ER, e através da análise das questões subjetivas respondidas pelos especialistas das aplicações.

Convém ressaltar que os especialistas foram instruídos a avaliar a arquitetura recuperada sob o ponto de vista de elementos arquiteturais que representem um conjunto de serviços coesos ou conceitos do domínio.

A arquitetura e os elementos arquiteturais recuperados são precisos, i.e. estão corretos?

A precisão foi calculada com base nas métricas descritas na questões Q1 e Q2. Duas formas de cálculo de precisão foram aplicadas: um cálculo de precisão global (i.e. PrecisãoArquitetura), avaliando a arquitetura recuperada no nível mais alto de abstração, e um cálculo de precisão por elemento arquitetural (i.e. PrecisãoElemento), avaliando a estrutura interna de cada elemento arquitetural em um nível de projeto detalhado. A precisão global foi calculada extraindo-se o valor de "RecuperadosEls" da resposta da questão Q1 do formulário 1 (Anexo A/A.2), i.e. a quantidade de elementos

arquiteturais recuperados. O valor de "RelevantesEls" na fórmula foi extraído da resposta à questão Q2, i.e. elementos arquiteturais recuperados que correspondem a elementos arquiteturais da aplicação. A Tabela 6.4 apresenta os valores de precisão globais para cada uma das aplicações.

Convém ressaltar, entretanto, que o especialista da Odyssey-PSW respondeu apenas à questão Q8 do formulário 1, não indicando elementos arquiteturais reconhecidos, não reconhecidos ou ausentes na arquitetura recuperada. Ao invés disso, o especialista realizou uma marcação na própria arquitetura recuperada, indicando, para cada classe de um elemento arquitetural, o elemento arquitetural em que ela deveria estar alocada, quando ela não estivesse alocada adequadamente.

Tabela 6.4: Precisão global da arquitetura recuperada.

Aplicação	RecuperadosEls ∩ RelevantesEls	RecuperadosEls	Precisão da Arquitetura Recuperada
ArchTrace	4	4	100%
TIM	3	5	60%

Após calculada a precisão global, foi calculada a precisão por elemento arquitetural (i.e. PrecisãoElemento). Nesse caso, foi considerado o número de classes recuperadas no elemento arquitetural e dentre elas as que o especialista considerou relevantes nesse elemento. A título de simplificação, foi extraída uma média de precisão dos elementos arquiteturais para cada aplicação, a qual é apresentada na Tabela 6.5. Entretanto, a Tabela 6.8 apresenta esses valores de precisão por elemento arquitetural, conforme determina a métrica M2.

Tabela 6.5: Média de precisão dos elementos arquiteturais recuperados.

Aplicação	Média de Precisão dos Elementos Arquiteturais Recuperados
ArchTrace	70%
TIM	76%
Odyssey-PSW	73%

A arquitetura e os elementos arquiteturais recuperados têm uma boa taxa de revocação?

A revocação foi calculada com base nas métricas descritas nas questões Q3 e Q4. Assim como para a precisão, 2 valores de revocação foram calculados: um global, da arquitetura no nível mais alto de abstração (i.e. RevocaçãoArquitetura), e um por elemento arquitetural (i.e. RevocaçãoElemento), em um nível de projeto detalhado.

Convém ressaltar que para a Odyssey-PSW, em função da falta de informação, apenas a revocação por elemento arquitetural foi calculada.

A revocação global foi calculada verificando-se a resposta à questão Q2 do formulário 1 (Anexo A/A.2), i.e. elementos arquiteturais recuperados relevantes, e a resposta à questão 4, i.e. elementos arquiteturais que estão faltando. Somando-se os elementos arquiteturais recuperados relevantes com aqueles que estão faltando, é possível se chegar ao valor total dos elementos arquiteturais, i.e. elementos relevantes na fórmula. A Tabela 6.6 apresenta os percentuais de revocação globais.

Tabela 6.6: Revocação global da arquitetura recuperada.

Aplicação	RecuperadosEls ∩ RelevantesEls	RelevantesEls	Revocação da Arquitetura Recuperada
ArchTrace	4	5	80%
TIM	3	5	60%

Para o cálculo da revocação por elemento arquitetural, foi levado em conta o número de classes recuperadas no elemento arquitetural que eram relevantes e o número de classes que estariam faltando para completar o elemento. Essas classes que estariam faltando poderiam tanto estar alocadas em um outro elemento arquitetural quanto não terem sido monitoradas. A Tabela 6.7 apresenta a média de revocação dos elementos arquiteturais para cada uma das aplicações. A Tabela 6.8 apresenta o valor de revocação por elemento arquitetural, conforme determina a métrica M4.

Tabela 6.7: Média de revocação dos elementos arquiteturais recuperados.

Aplicação	Média de Revocação dos Elementos Arquiteturais Recuperados
ArchTrace	60,5%
TIM	69%
Odyssey-PSW	46%

A Tabela 6.8 apresenta o detalhamento dos valores de precisão e revocação por elemento arquitetural recuperado (i.e. PrecisãoElemento e RevocaçãoElemento). Os nomes dos elementos arquiteturais iniciados por EA foram atribuídos por ArchMine e o segundo nome foi atribuído pelo especialista (i.e. EAx:nome especialista). Convém ressaltar que na Odyssey-PSW mais de um mesmo elemento arquitetural recuperado correspondia ao mesmo elemento arquitetural da aplicação.

Os elementos arquiteturais recuperados apóiam a compreensão da aplicação?

A fim de responder a esta pergunta, foram consultadas as respostas dadas pelos especialistas à questão Q9 do formulário 2 (Anexo A/A.2), que visa verificar se os especialistas consideram que a arquitetura recuperada representa um possível modelo arquitetural para a aplicação. Neste ponto, um dos especialistas, o da ArchTrace, respondeu que sim e os outros dois especialistas, i.e. os especialistas da TIM e da Odyssey-PSW, responderam que parcialmente.

O especialista da ArchTrace considerou que o resultado foi bastante próximo do idealizado na concepção da aplicação. O especialista da Odyssey-PSW considerou que a arquitetura reflete bem os elementos arquiteturais que ele chama de "elementos de mais alto nível", ou seja, com funcionalidades mais específicas, e que os elementos mais gerais, "da base do software", como o especialista denomina, acabaram ficando muito espalhados em vários elementos arquiteturais recuperados. Isso se deve ao fato desses elementos mais gerais apresentarem classes que são utilizadas por outros elementos em situações específicas. Por fim, o especialista da TIM julgou que a arquitetura recuperada representa parcialmente a aplicação, porque algumas classes foram alocadas incorretamente nos elementos arquiteturais recuperados.

Vale ressaltar que a perspectiva de projeto considerada foi a de elementos arquiteturais que representassem conceitos da aplicação, ou seja, os especialistas foram instriuídos a avaliar cada aplicação sob esse ponto de vista.

Considerações em Relação à Hipótese Nula

Os valores de precisão e revocação obtidos com ArchMine foram comparados com valores de precisão e revocação relatados na literatura de ER (ANQUETIL *et al.*, 1999; SARTIPI, 2003; SARTIPI e KONTOGIANNIS, 2003).

Em (SARTIPI, 2003; SARTIPI e KONTOGIANNIS, 2003), a média de precisão dos elementos arquiteturais recuperados em dois estudos é de 68,5%, enquanto a média de revocação é de 66,3%. O autor argumenta que esses valores são promissores. Extraindo-se uma média dos valores de precisão entre os elementos arquiteturais da Tabela 6.8, obtém-se 72,6% de precisão dos elementos arquiteturais recuperados com ArchMine contra 50,1% de revocação. A fim de comparar os valores obtidos com a utilização de ArchMine com os valores relatados em (SARTIPI, 2003), a média

harmônica (BAEZA-YATES e RIBEIRO-NETO, 1999) entre precisão e revocação é calculada, da seguinte forma:

$$\textbf{MédiaHarmônicaArchMine} = \frac{2}{\frac{1}{0,5} + \frac{1}{0,72}} \cong 0,6$$

$$\textbf{MédiaHarmônicaSartipi} = \frac{2}{\frac{1}{0,68} + \frac{1}{0,66}} \cong 0,67$$

Quanto mais a média harmônica se aproxima de 1, melhor é o compromisso entre precisão e revocação. Através da média harmônica obtida, percebe-se que ArchMine gera bons resultados quando comparada a outras abordagens, como a abordagem de SARTIPI (2003), embora com um objetivo e visão arquitetural recuperados diferentes.

Em (ANQUETIL *et al.*, 1999), vários algoritmos de *clustering* hierárquicos são avaliados e a precisão e revocação dos grupos gerados é calculada. Entretanto, o cálculo de precisão e revocação é feito de forma diferente ao realizado nesta tese, i.e. levando-se em conta pares de entidades presentes nos grupos (*clusters*) recuperados. Dessa forma, não é possível realizar uma comparação dos valores de precisão e revocação obtidos em (ANQUETIL *et al.*, 1999) com os valores obtidos neste trabalho. Porém, duas considerações são relevantes: como o nível de granularidade dos grupos (*clusters*) obtidos nos experimentos de (ANQUETIL *et al.*, 1999) é menor do que na decomposição modular do sistema fornecida por um especialista, os autores ressaltam que os valores de precisão são sempre melhores que os de revocação; e, ainda, existe uma tendência de queda nos valores de revocação à medida que a precisão aumenta.

As mesmas tendências podem ser observadas neste primeiro estudo de viabilidade, principalmente para a aplicação Odyssey-PSW, a maior das aplicações testadas, conforme mostra o gráfico da Figura 6.1. Os elementos arquiteturais obtidos tendem a apresentar granularidade menor do que os considerados pelo especialista. Dessa forma, o gráfico da Figura 6.1 mostra que a precisão tende a se manter mais alta que a revocação, e que quando a precisão chega a 100%, a revocação tende a cair, sendo observado o mesmo comportamento entre essas duas medidas quando a revocação atinge seu cume de 100%.

Tabela 6.8: Precisão e revocação dos elementos arquiteturais recuperados.

Aplicação	Elemento Arquitetural	Nº Classes	Precisão	Revocação
ArchTrace	EA1: central classes	14	67%	71%
	EA2: policy control	20	67%	70%
	EA3: config	24	90%	38%
	EA4: repository conf. control	8	56%	63%
TIM	EA1: central classes	12	50%	54%
	EA2: planejamento	21	100%	69%
	EA4: controle	13	79%	83%
Odyssey-PSW	EA1: C1	91	67%	11%
	EA2: UIG	32	63%	63%
	EA3: R1	11	84%	100%
	EA4: C3	60	64%	36%
	EA5: C1	91	94%	32%
	EA6: C3	60	30%	16%
	EA7: C8	6	33%	16%
	EA8: Q1	21	80%	57%
	EA9: C5	25	89%	62%
	EA10: C6	4	36%	100%
	EA11: C3	60	44%	18%
	EA12: S1	10	100%	60%
	EA13: X1	28	79%	67%
	EA14: T1	19	100%	94%
	EA15: P1	6	50%	50%
	EA16: C8	6	100%	67%
	EA17: C1	91	100%	10%
	EA18: C1	91	68%	59%
	EA19: C3	60	53%	26%
	EA20: X1	28	70%	25%
	EA21: I1	11	100%	36%
	EA22: Q1	21	67%	10%
	EA23: S1	10	100%	40%

Dessa forma, os resultados obtidos por ArchMine em termos de precisão e revocação fornecem indícios da sua viabilidade se comparados a outras abordagens da literatura, embora novos estudos para confirmar essa viabilidade sejam necessários.

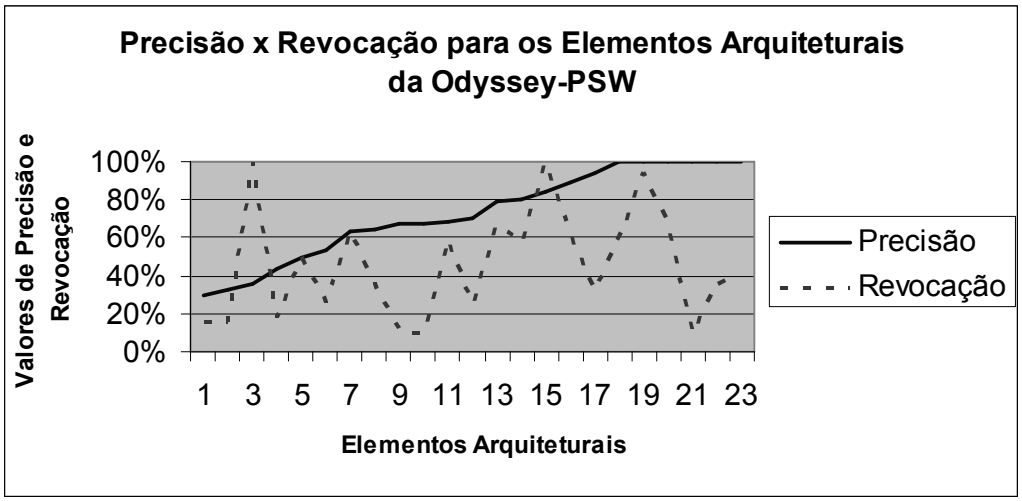

Figura 6.1: Precisão x revocação dos elementos arquiteturais recuperados para a Odyssey-PSW.

6.2.4 – Lições Aprendidas

Esse estudo preliminar permitiu obter *feedback* dos especialistas, que, juntamente com o pesquisador, identificaram pontos de melhoria, contribuições e limitações de ArchMine. Esses aspectos foram identificados a partir das respostas dadas pelos especialistas às questões do formulário 2 e através de conversas informais entre o pesquisador e os especialistas durante a avaliação dos resultados, na qual o pesquisador explicou pra os especialistas o funcionamento e heurísticas de ArchMine.

Dentre os aspectos onde ArchMine poderia ser melhorada, considerando-se o seu estágio naquele momento, destacaram-se:

- O mapeamento de *threads* para transações deveria ser configurável e não fixo como se encontrava nesse momento. Ou seja, a abordagem deveria permitir as seguintes opções de configuração: mapear diferentes *threads* que são executadas em um mesmo cenário de caso de uso para diferentes transações; incluir todas as *threads* que são executadas em um mesmo cenário de caso de uso em uma mesma transação que representa o cenário; ou eliminar as *threads* secundárias do processo de mineração e agrupar as classes que só participam dessas *threads* separadamente, sendo essa a única opção disponível no momento da realização do estudo.
- Algumas superclasses e interfaces da aplicação não foram monitoradas e, portanto, não foram alocadas na arquitetura. Elas deveriam ser alocadas no elemento que apresentasse o maior número de subclasses, ou classes que a

realizam, para o caso de interfaces. Dessa forma, a análise estática complementaria a dinâmica na sugestão de agrupamentos de classes.

- Para fins de apoio à manutenção, alguns especialistas apontaram a necessidade de relacionar aos elementos arquiteturais recuperados os cenários de caso de uso implementados por suas classes.
- Além disso, os nomes dos elementos arquiteturais não possuíam semântica, pois eram compostos pelo nome "ElementoArquitetural" ou "EA" mais um número seqüencial. Assim, foi sugerido que nomes com maior semântica fossem gerados.

Em relação às limitações, os seguintes aspectos foram constatados:

- A intervenção manual na coleta dos cenários pode prejudicar a escalabilidade da abordagem.
- A seleção dos cenários tem um grande impacto na arquitetura resultante. Uma seleção imprópria de cenários pode levar a uma arquitetura pobre. Além disso, os cenários devem cobrir todos ou a maior parte das possibilidades de uso da aplicação. Foi ressaltado também pelos especialistas que cenários referentes a requisitos não-funcionais, como "Salvar", podem ter um impacto negativo na recuperação da arquitetura se os dados de entrada privilegiarem uma ou outra entidade da aplicação. Os dados de entrada dos cenários também exercem forte impacto na arquitetura recuperada e, quanto maior o conhecimento da abordagem ArchMine por parte de quem define e monitora os cenários, melhor é a qualidade da arquitetura recuperada.

Nesse estudo, a mineração dos rastros de execução e reconstrução dos elementos arquiteturais levaram em torno de 4 horas para a ArchTrace, 2 horas para a TIM e 30 horas para a Odyssey-PSW. A fim de reduzir esse esforço, uma maior automação do processo de mineração se fazia necessária.

Quanto às contribuições observadas nesse estudo, ressaltaram-se:

- A recuperação de uma decomposição funcional da aplicação, a qual revela elementos arquiteturais coesos que refletem a implementação de conceitos do domínio, sendo benéfica à reutilização. Para arquiteturas que seguem os estilos arquiteturais de camadas ou MVC (*Model-View-Controller*), por exemplo, os elementos arquiteturais tendem a ser gerados de forma perpendicular às camadas, representando conceitos do domínio.
- Entretanto, em algumas circunstâncias, mesmo arquiteturas em camadas podem ser corretamente recuperadas pela abordagem ArchMine. Essas circunstâncias

ocorrem quando há uma intensa troca de mensagens entre classes de uma mesma camada e poucas mensagens cruzando as fronteiras das diferentes camadas. Casos onde as camadas possuem um ponto único de acesso, como uma Fachada, podem refletir essa situação.

- Em alguns casos, a análise da arquitetura recuperada com o especialista levou à reestruturação da própria arquitetura original da aplicação. Através da arquitetura recuperada, o especialista pôde identificar classes que não haviam sido corretamente alocadas na arquitetura original da aplicação, em função dos serviços que ela oferece.

Após esse primeiro estudo de viabilidade, refinamentos foram realizados na abordagem ArchMine em função das lições aprendidas. Resumidamente, esses refinamentos envolveram: mapeamento de *threads* configurável; análise estática complementando a dinâmica; registro na arquitetura dos cenários de casos de uso que dão origem ao elemento arquitetural; geração dos nomes dos elementos arquiteturais com base em *substrings* comuns em suas classes; maior automação do processo de mineração.

Em função desses refinamentos, o ideal seria re-executar ArchMine sobre as mesmas aplicações, a fim de verificar o quanto os refinamentos de fato melhoram a qualidade da arquitetura recuperada sob o ponto de vista dos objetivos estabelecidos neste estudo. Entretanto, ao invés de repetir esse primeiro estudo, optou-se pela realização de um segundo estudo de viabilidade comparando ArchMine com uma outra abordagem de recuperação de arquitetura. Porém, a aplicação ArchTrace (ver Tabela 6.1), também foi utilizada nesse segundo estudo, o que nos permite tecer algumas considerações quanto à melhoria alcançada.

Além desse *feedback* em relação à mineração e recuperação dos elementos arquiteturais, um retorno dos especialistas em relação ao processo de avaliação da arquitetura também foi obtido. O formulário apresentado no Anexo A/A.3 visa avaliar a clareza dos demais formulários. Nesse contexto, um dos especialistas considerou as questões Q9 a Q13 do formulário 2 redundantes e um dos especialistas, o da Odyssey-PSW (ver Tabela 6.1), reclamou que o preenchimento da tabela da questão Q6 do formulário 1 se torna árduo, exigindo alto esforço, quando a arquitetura é muito grande.

Com base nessas respostas, os formulários de avaliação da arquitetura recuperada foram refinados (Anexo B/B.2). Questões redundantes, como a Q9 e Q10, Q11 e Q12 no formulário do Anexo A/A.2, foram reduzidas a uma única questão.

6.3 – Estudo Comparativo de ArchMine

A fim de obter mais indícios da viabilidade de ArchMine, um estudo comparativo em relação a uma outra abordagem de recuperação de arquitetura, destacada na literatura de ER, a abordagem implementada na ferramenta Bunch (MITCHELL e MANCORIDIS, 2006), foi realizado. A escolha de Bunch, cuja abordagem relacionada foi descrita na seção 3.2.3.4, se deu em função dessa já ter sido utilizada em vários outros estudos (MANCORIDIS *et al.*, 1999; ANQUETIL e LETHBRIDGE, 2003; WEN e TZERPOS, 2004; MITCHELL e MANCORIDIS, 2006), e estar sendo refinada ao longo do tempo, o que oferece indícios da sua maturidade, além de possuir ferramental disponível. Assim, a arquitetura de duas aplicações acadêmicas foi recuperada aplicando Bunch e ArchMine.

Como Bunch é automatizada, somente a eficácia das abordagens pode ser comparada (i.e. qualidade da arquitetura recuperada), não sendo possível realizar uma comparação em termos de custo/eficiência (i.e. qualidade/esforço), uma vez que ArchMine é semi-automatizada. Por fim, Bunch analisa um grafo que representa o modelo estático da aplicação, o qual contém nós, representando as entidades do código fonte a serem agrupadas em elementos arquiteturais, e arcos, representando os relacionamentos entre elas. Dessa forma, Bunch não está limitada a analisar aplicações OO escritas em Java, como ArchMine. Porém, a fim de comparar as duas abordagens, aplicações com esse perfil foram selecionadas para o estudo.

6.3.1 – Planejamento do Estudo

Objetivo Global

O propósito deste estudo é comparar a abordagem de recuperação de arquitetura ArchMine refinada com a abordagem de recuperação de arquitetura implementada pela ferramenta Bunch, em termos da sua eficácia para apoio à recuperação da arquitetura de sistemas legados OO escritos em Java.

Objetivo do Estudo

Analisar as abordagens de recuperação de arquitetura ArchMine refinada e Bunch

Com o propósito de caracterizar

Com respeito à eficácia em apoiar a recuperação da arquitetura de software orientado a objetos escrito em Java

Do ponto de vista de engenheiros de software

No contexto da recuperação da arquitetura de aplicações acadêmicas, para as quais um especialista se encontra disponível para avaliar as arquiteturas recuperadas.

Questões

Q1: **A eficácia da abordagem ArchMine na recuperação da arquitetura é melhor que a de Bunch, quando ambas são aplicadas sobre os mesmos sistemas e avaliadas pelos mesmos especialistas?**

Métricas: a eficácia de uma abordagem de recuperação de arquitetura pode ser obtida em função do compromisso atingido entre a precisão e a revocação dos elementos arquiteturais recuperados, que pode ser expresso através da média harmônica (BAEZA-YATES e RIBEIRO-NETO, 1999). O ideal é que os elementos arquiteturais estejam corretos, com o menor número de classes equivocadas quanto possível (i.e. precisão) e, ao mesmo tempo, com a maior cobertura de classes quanto possível (i.e. revocação).

Nesse sentido, devem ser extraídas métricas de precisão e revocação para os elementos arquiteturais recuperados por cada uma das abordagens, a fim de permitir efetuar a comparação, calculadas da seguinte forma:

$$\textbf{PrecisãoArchMine} = \frac{\sum_{i=1}^{els\,\mathrm{Re}\,cuperadosArchMine} \frac{|\,ClassesElArqArchMine\_i \cap \mathrm{Re}\,levantes\,|}{|\,ClassesElArqArchMine\_i\,|}}{els\,\mathrm{Re}\,cuperadosArchMine}$$

$$\textbf{PrecisãoBunch} = \frac{\sum_{i=0}^{els\,\mathrm{Re}\,cuperadosBunch} \frac{|\,ClassesElArqBunch\_i \cap \mathrm{Re}\,levantes\,|}{|\,ClassesElArqBunch\_i\,|}}{els\,\mathrm{Re}\,cuperadosBunch}$$

$$\textbf{RevocaçãoArchMine} = \frac{\sum_{i=0}^{els\,\mathrm{Re}\,cuperadosArchMine} \frac{|\,ClassesElArchMine\_i \cap \mathrm{Re}\,levantes\,|}{|\,\mathrm{Re}\,levantes\,|}}{els\,\mathrm{Re}\,cuperadosArchMine}$$

$$\textbf{RevocaçãoBunch} = \frac{\sum_{i=0}^{els\,\mathrm{Re}\,cuperadosBunch} \frac{|\,ClassesElArqBunch\_i \cap \mathrm{Re}\,levantes\,|}{|\,\mathrm{Re}\,levantes\,|}}{els\,\mathrm{Re}\,cuperadosBunch}$$

Onde:

elsRecuperadosArchMine = número total de elementos arquiteturais recuperados pela abordagem ArchMine;

$ClassesElArqArchMine\_i \cap Relevantes$ = classes recuperadas por ArchMine no elemento arquitetural *i* que são relevantes do ponto de vista do especialista;

$ClassesElArqArchMine\_i$ = classes recuperadas por ArchMine no elemento arquitetural *i*;

elsRecuperadosBunch = número total de elementos arquiteturais recuperados pela abordagem implementada em Bunch;

$ClassesElArqBunch\_i \cap Relevantes$ = classes recuperadas por Bunch no elemento arquitetural *i* que são relevantes do ponto de vista do especialista;

$ClassesElArqBunch\_i$ = classes recuperadas por Bunch no elemento arquitetural *i*;

Relevantes = classes que deveriam compor o elemento arquitetural recuperado do ponto de vista do especialista.

Conforme observado nas fórmulas, a precisão e revocação são obtidas através da média de precisão e revocação dos elementos arquiteturais recuperados por cada uma das abordagens. Uma vez obtidas a precisão e revocação, a média harmônica é então calculada, representando a eficácia das abordagens, como se segue.

$$\textbf{MédiaHarmônicaArchMine} = \frac{2}{\frac{1}{Precisão ArchMine} + \frac{1}{Revocação ArchMine}}$$

$$\textbf{MédiaHarmônicaBunch} = \frac{2}{\frac{1}{Precisão Bunch} + \frac{1}{Revocação Bunch}}$$

Seleção do Contexto

O estudo é realizado em ambiente acadêmico e a recuperação da arquitetura é executada para duas aplicações acadêmicas de diferentes tamanhos. Uma das aplicações é a ArchTrace (MURTA *et al.*, 2006), já utilizada no estudo anterior. A outra aplicação é o ambiente de reutilização de software OdysseyLight (ODYSSEY, 2007). A Tabela 6.9 resume as características das aplicações sob estudo.

Essas duas aplicações foram selecionadas levando-se em conta o fato de terem sido desenvolvidas em Java, terem tamanhos diferenciados, e haver um especialista disponível para apoiar o estudo.

Tabela 6.9: Características das aplicações objetos do segundo estudo de viabilidade.

Aplicação	Classes	Ligações entre Classes	LOC	Estilos Arquiteturais	Cenários de Casos de Uso	Versão
ArchTrace	80	308	4 695	Model-View-Controller (MVC)	14	1.0.0
OdysseyLight	595	3610	58 690	Microkernel	146	1.5.0

Seleção dos Participantes

O próprio pesquisador aplica as abordagens de recuperação de arquitetura, i.e. ArchMine e Bunch, sobre as duas aplicações. Os especialistas das aplicações atuam na etapa de avaliação das arquiteturas recuperadas.

Definição das Hipóteses

Hipótese Nula (H0): Não existe diferença de eficácia na recuperação da arquitetura pelas abordagens ArchMine e Bunch.

H0:

$MédiaHarmônicaArchMine = MédiaHarmônicaBunch$

$$\textbf{MédiaHarmônicaArchMine} - \textbf{MédiaHarmônicaBunch} = 0$$

Hipótese Alternativa (H1): ArchMine apresenta melhor eficácia na recuperação da arquitetura que a abordagem implementada em Bunch.

H1:

$MédiaHarmônicaArchMine \neq MédiaHarmônicaBunch$

$$\textbf{MédiaHarmônicaArchMine} - \textbf{MédiaHarmônicaBunch} > 0$$

Hipótese Alternativa (H2): Bunch apresenta melhor eficácia na recuperação da arquitetura que ArchMine.

H2:

$MédiaHarmônicaBunch \neq MédiaHarmônicaArchMine$

$$\textbf{MédiaHarmônicaBunch} - \textbf{MédiaHarmônicaArchMine} > 0$$

Variáveis

Neste estudo, Bunch é executada com o conjunto de parâmetros padrão sugeridos pela ferramenta, dentre eles, o método de *clustering Hill Climbing* (MANCORIDIS et al., 1999). A ferramenta Bunch aceita como entrada um grafo de

entidades e relacionamentos representando a estrutura estática do código fonte. A ferramenta empregada para extrair esse grafo a partir do código tem influência na qualidade dos resultados produzidos por Bunch. Além disso, Bunch aceita que sejam atribuídos pesos aos relacionamentos no grafo, o que também impacta a qualidade dos resultados produzidos. Neste estudo, pesos são atribuídos aos relacionamentos do grafo através da métrica de acoplamento CBO proposta por CHIDAMBER e KEMERER (1994). A CBO foi escolhida por ser a métrica de acoplamento entre classes mais comumente utilizada em trabalhos acadêmicos.

A confiança mínima adotada em ArchMine foi de 60% e o suporte mínimo de 0%, como no primeiro estudo de viabilidade, os cenários das aplicações foram definidos junto aos especialistas e os antecedentes para a mineração sugeridos em função das heurísticas propostas na abordagem. Variações nos valores dessas variáveis não são investigadas, apenas variação nas abordagens de recuperação de arquitetura empregadas, as quais representam os tratamentos do estudo. As variáveis independentes desse estudo são apresentadas na Tabela 6.10 e as variáveis dependentes na Tabela 6.11.

Tabela 6.10: Variáveis independentes do segundo estudo de viabilidade.

Variável	Valores	Descrição
APL	ArchTrace e OdysseyLight	Aplicações para as quais as arquiteturas são recuperadas.
CEN	Cenários	Cenários especificados para as aplicações ArchTrace e OdysseyLight.
ARCREC	ArchMine e Bunch	Abordagens de recuperação de arquitetura utilizadas, as quais representam os tratamentos nesse estudo.
CNF	Real	Valor de confiança mínima aplicado em ArchMine.
SUP	Real	Valor de suporte mínimo aplicado em ArchMine.
ANT	Antecedentes	Classes selecionadas como antecedentes para a mineração em ArchMine.
MCL	Método de *Clustering*	Método de *clustering* selecionado na ferramenta Bunch.
PMR	Parâmetros Algoritmo *Clustering*	Parâmetros para o algoritmo de *clustering* utilizado em Bunch.
GRF	Grafo Extraído por uma Ferramenta de Análise Estática	Grafo extraído por uma ferramenta de análise estática, alimentado na ferramenta Bunch.

Tabela 6.11: Variáveis dependentes do segundo estudo de viabilidade.

Variável	Valores	Descrição
PRC	Real	Média de precisão dos elementos arquiteturais recuperados para a ArchTrace e OdysseyLight, através da ArchMine e Bunch.
REC	Real	Média de revocação dos elementos arquiteturais recuperados para a ArchTrace e OdysseyLight, através da ArchMine e Bunch.
EFC	Real	Eficácia na recuperação da arquitetura pelas abordagens ArchMine e Bunch, obtida através da média harmônica.
TMPAvaliacao	Real	Tempo de avaliação das arquiteturas recuperadas com Bunch e ArchMine pelos especialistas.

Validade do Estudo

Validade de Conclusão

Confiabilidade na Aplicação do Tratamento ArchMine: o pesquisador realizou manutenções na aplicação OdysseyLight e, portanto, tem algum conhecimento sobre a sua estrutura. Dessa forma, como a ArchMine é uma abordagem semi-automatizada, este conhecimento do pesquisador pode representar um viés no emprego dessa abordagem para a recuperação da arquitetura da aplicação OdysseyLight. A fim de minimizar esta ameaça à validade do estudo, o pesquisador buscou aplicar as heurísticas relacionadas à abordagem de forma "cega", ou seja, seguindo a risca as suas diretrizes. O mesmo foi feito na recuperação da arquitetura de ArchTrace, para a qual algum conhecimento da estrutura também já existia em função da sua utilização no primeiro estudo de viabilidade.

Confiabilidade na Aplicação do Tratamento Bunch: os resultados produzidos por Bunch são impactados pela qualidade do grafo extraído do código com a ferramenta de análise estática. Dessa forma, o ideal seria executar a Bunch com diferentes modelos de entrada, a fim de verificar a sua eficácia em diferentes situações. O mesmo vale para a ArchMine em relação à variação dos cenários, confiança e suporte mínimos e antecedentes para a mineração. Entretanto, avaliar a variação nos valores dessas variáveis dependentes não é objetivo deste estudo, sendo necessária a realização de estudos específicos com essa finalidade.

Validade de Construção

Viés da Origem das Aplicações: assim como no primeiro estudo de viabilidade,

este estudo experimental utiliza aplicações acadêmicas desenvolvidas pelo mesmo grupo de pesquisa do pesquisador, i.e. o grupo de reutilização da COPPE/UFRJ. Portanto, o estudo pode não ser representativo da teoria sob a qual se sustenta, uma vez que pode não ficar claro se a causa dos resultados do estudo é decorrente do fato do pesquisador pertencer ao mesmo grupo de pesquisa onde as aplicações foram desenvolvidas, ou se o resultado de fato se deve à eficácia dos diferentes tratamentos. Portanto, um estudo com aplicações desenvolvidas em outro contexto se faz necessário.

Viés do Número de Aplicações: duas aplicações é um número pequeno para traçar conclusões definitivas a respeito da comparação entre a eficácia das abordagens. Pode não ficar claro se a causa dos resultados do estudo é decorrente da aplicação dos tratamentos ou de características específicas das aplicações.

Validade Externa

Interação entre Ambiente e Tratamento: o fato das aplicações, objetos da recuperação de arquitetura, terem sido desenvolvidas em ambiente acadêmico pode representar uma ameaça à generalização dos resultados do estudo para o ambiente industrial.

6.3.2 – Execução do Estudo Experimental

Preparação

Nesta etapa, os especialistas das aplicações preencheram o formulário de consentimento (Anexo B/B.1) para apoio no estudo. Eles apoiaram a fase de avaliação das arquiteturas recuperadas.

Durante a preparação dos dados para a abordagem ArchMine, o participante do estudo (i.e. o próprio pesquisador) definiu os cenários de casos de uso para a aplicação OdysseyLight e os executou, coletando os rastros de execução. O pesquisador tem conhecimento suficiente da aplicação para realizar essa etapa sem o apoio do especialista. Uma ampla cobertura em relação às funcionalidades da aplicação OdysseyLight e aos caminhos possíveis para executá-las foi buscada nesse momento (ver Tabela 6.9).

Em relação ao ArchTrace, somente alguns cenários foram redefinidos e re-executados com base nas lições aprendidas no primeiro estudo de viabilidade.

O modelo estático das aplicações foi recuperado no ambiente OdysseyLight através da ferramenta Ares. Como a ferramenta Bunch aceita que pesos sejam atribuídos aos relacionamentos para a recuperação da arquitetura, a ferramenta Odyssey-Metrics

foi utilizada para calcular a métrica CBO, atribuindo pesos aos relacionamentos do grafo extraído com a Ares. Dessa forma, como determina a métrica CBO, foi contabilizado o número de relacionamentos diretos entre cada par de classes, somando-se a esses os relacionamentos herdados das suas superclasses. Com base nessa contagem, um peso foi atribuído ao acoplamento entre cada par de classes.

Operação

Uma vez preparados os artefatos de entrada para a recuperação da arquitetura, a reconstrução dos elementos arquiteturais com ArchMine foi realizada pelo pesquisador com o apoio da ferramenta TraceMining, evoluída em função dos refinamentos realizados na abordagem ArchMine entre o 1º e o 2º estudo de viabilidade.

Posteriormente, foi realizada a recuperação da arquitetura na ferramenta Bunch. A ferramenta Bunch recebeu como entrada o grafo gerado pela Ares para cada uma das aplicações, contendo pesos nos relacionamentos atribuídos através da métrica CBO. Como Bunch apresenta um algoritmo de agrupamento hierárquico, elementos arquiteturais foram selecionados em um nível da hierarquia que fosse similar ao nível de granularidade atingido por ArchMine. Esse nível de granularidade foi comparado levando-se em conta o número de elementos arquiteturais nos modelos gerados pelas 2 ferramentas, que deveria ser similar. Dessa forma, a comparação das arquiteturas pôde ser realizada para modelos no mesmo nível de abstração.

Os modelos arquiteturais gerados pela TraceMining (abordagem ArchMine) e pela Bunch foram exportados para o ambiente OdysseyLight e apresentados aos especialistas. A fim de evitar viés, os especialistas não foram informados sobre que arquitetura havia sido gerada por que abordagem. Eles avaliaram as arquiteturas verificando a coerência dos elementos arquiteturais recuperados, através da coesão das suas classes. Os especialistas preencheram os formulários de avaliação (Anexo B/B.2).

6.3.3 – Análise dos Resultados Obtidos

Durante o planejamento desse estudo, foi definido como objetivo principal comparar as abordagens de recuperação de arquitetura ArchMine e Bunch em relação à sua eficácia na recuperação da arquitetura de aplicações OO em Java. Essa comparação, conforme os questionamentos identificados, foi realizada através da análise dos valores de precisão e revocação dos elementos arquiteturais recuperados.

A eficácia da abordagem ArchMine na recuperação da arquitetura é melhor que a de Bunch, quando ambas são aplicadas sobre os mesmos sistemas e avaliadas pelos mesmos especialistas?

O cálculo da eficácia das abordagens na recuperação da arquitetura foi realizado calculando-se a média de precisão e revocação dos elementos arquiteturais recuperados e extraindo-se desses resultados a média harmônica.

O cálculo de precisão individual de cada elemento arquitetural foi efetuado verificando-se as classes recuperadas que são relevantes (classes recuperadas no elemento arquitetural, subtraindo-se aquelas alocadas incorretamente, informadas na última coluna da tabela da questão Q2 do formulário 1 – Anexo B/B.2) e dividindo este número pelas classes recuperadas no elemento arquitetural. O cálculo individual de revocação de cada elemento arquitetural foi efetuado verificando-se as classes recuperadas que são relevantes (obtidas da mesma forma explicada no cálculo de precisão) e dividindo este número pelo total de classes relevantes do elemento arquitetural (classes recuperadas no elemento, subtraindo-se as que estão alocadas incorretamente e somando aquelas que estão faltando, informadas na última coluna da tabela da questão Q3 do formulário 1 – Anexo B/B.2).

A Tabela 6.12 apresenta os resultados da média de precisão e revocação dos elementos arquiteturais para cada uma das aplicações, empregando cada uma das abordagens.

Tabela 6.12: Média de precisão e revocação dos elementos arquiteturais recuperados no segundo estudo de viabilidade.

Aplicação	Precisão Els. Arqs.		Revocação Els. Arqs.	
	ArchMine	Bunch	ArchMine	Bunch
OdysseyLight	75%	35%	48%	24%
ArchTrace	87%	62%	67%	69%

Com base nos valores da Tabela 6.12, a média harmônica é calculada conforme se apresenta a seguir. Vale ressaltar que as métricas definidas no Planejamento do Estudo são calculadas para cada uma das aplicações analisadas, i.e. OdysseyLight e ArchTrace.

$$\textbf{\textit{MédiaHarmônicaArchMine\_OdysseyLight}} = \frac{2}{\frac{1}{\text{PrecisãoArchMine\_OdysseyLight}} + \frac{1}{\text{RevocaçãoArchMine\_OdysseyLight}}} =$$

$$\frac{2}{\frac{1}{0,75} + \frac{1}{0,48}} \cong 0,58$$

$$\textbf{\textit{MédiaHarmônicaBunch\_OdysseyLight}} = \frac{2}{\frac{1}{\text{PrecisãoBunch\_OdysseyLight}} + \frac{1}{\text{RevocaçãoBunch\_OdysseyLight}}} =$$

$$\frac{2}{\frac{1}{0,35} + \frac{1}{0,24}} \cong 0,28$$

$$\textbf{\textit{MédiaHarmônicaArchMine\_ArchTrace}} = \frac{2}{\frac{1}{\text{PrecisãoArchMine\_ArchTrace}} + \frac{1}{\text{RevocaçãoArchMine\_ArchTrace}}} =$$

$$\frac{2}{\frac{1}{0,87} + \frac{1}{0,67}} \cong 0,76$$

$$\textbf{\textit{MédiaHarmônicaBunch\_ArchTrace}} = \frac{2}{\frac{1}{\text{PrecisãoBunch\_ArchTrace}} + \frac{1}{\text{RevocaçãoBunch\_ArchTrace}}} =$$

$$\frac{2}{\frac{1}{0,62} + \frac{1}{0,69}} \cong 0,66$$

Pela análise dos resultados, percebe-se que ArchMine atingiu melhores resultados que Bunch para o OdysseyLight e ArchTrace. Os baixos valores de precisão e revocação obtidos por Bunch para o OdysseyLight se devem ao fato do OdysseyLight estar organizado em elementos arquiteturais funcionalmente coesos, porém fortemente acoplados. Bunch privilegia elementos arquiteturais fracamente acoplados.

Considerações em Relação às Hipóteses

A média harmônica obtida na recuperação das arquiteturas foi melhor para a ArchMine do que para a Bunch. Dessa forma, há indícios de que seria possível refutar a hipótese nula e confirmar a hipótese alternativa H(1), i.e. que a eficácia de ArchMine é melhor que a de Bunch na recuperação de arquitetura. Porém, estudos com novas aplicações em diferentes domínios e desenvolvidas por diferentes equipes se fazem necessários. As 2 aplicações analisadas nesse estudo foram desenvolvidas pela mesma equipe que a abordagem ArchMine, o que, conforme ressaltado, representa um possível viés no estudo. Além disso, conforme mencionado no capítulo 3, não existe uma abordagem de recuperação de arquitetura que seja adequada a todos os sistemas, estilos arquiteturais, linguagens de programação ou domínios. O que esse estudo indica é que ArchMine é capaz de oferecer resultados interessantes em determinados contextos, assim como outras abordagens de ER.

6.3.4 – Lições Aprendidas

A partir da análise dos resultados do estudo junto aos especialistas, algumas conclusões puderam ser traçadas:

- Há a necessidade de um mecanismo para verificar a cobertura de classes atingida por ArchMine, pois algumas classes presentes na arquitetura recuperada com a ferramenta Bunch não estavam presentes na arquitetura recuperada com ArchMine. Essa análise de cobertura pode indicar a necessidade de definição de um maior número de cenários.
- Um dos especialistas, o que avaliou a arquitetura do OdysseyLight, que possui 595 classes, argumentou que o processo de avaliação é extremamente cansativo e exige um alto esforço. Na realidade, a situação ideal para o cálculo de precisão e revocação é quando existe uma arquitetura de referência para se comparar com a arquitetura recuperada (MITCHELL e MANCORIDIS, 2006). O especialista gastou em média 5 horas em cada uma das avaliações. Esse mesmo problema tinha sido apontado pelo especialista da aplicação Odyssey-PSW (tamanho de 596 classes) no primeiro estudo de viabilidade. Dessa forma, um novo instrumento de avaliação para a abordagem ArchMine foi desenvolvido e a sua aplicabilidade para a avaliação das arquiteturas recuperadas é avaliada nos dois próximos estudos de viabilidade.

- A precisão e revocação dos elementos arquiteturais recuperados com ArchMine refinada melhoraram em relação ao primeiro estudo de viabilidade para a aplicação ArchTrace, i.e. 87% de precisão contra 70% no primeiro estudo; e 67% de revocação contra 60,5% do primeiro estudo. Essa melhoria foi influenciada pelos refinamentos em ArchMine e pela redefinição de alguns cenários de ArchTrace junto ao especialista, que já possuía algum conhecimento da abordagem ArchMine. Vale lembrar que o pesquisador procurou recuperar novamente a arquitetura da ArchTrace aplicando as heurísticas definidas em ArchMine de forma "cega".

Convém ressaltar que seria interessante repetir esse estudo comparando ArchMine com outras abordagens de recuperação de arquitetura. Nesse estudo, ArchMine foi comparada com uma abordagem automática, que se baseia em princípios de "bom" projeto de software, i.e. coesão e acoplamento, para direcionar a reconstrução dos elementos arquiteturais. Seria interessante comparar ArchMine também com abordagens baseadas em requisitos funcionais, como a de BOJIC e VELASEVIC (2000), que tendem a recuperar elementos arquiteturais mais semelhantes aos recuperados com ArchMine. Entretanto, as abordagens, em geral, não apresentam ferramental disponível para uso. A abordagem de BOJIC e VELASEVIC (2000), além disso, está voltada à análise de sistemas implementados em C++.

Finalmente, os elementos arquiteturais recuperados com Bunch podem representar conceitos do domínio, dependendo de como se dá o acoplamento entre entidades do código.

6.4 – Estudo de Viabilidade das Extensões realizadas na Abordagem de Avaliação de Arquitetura

A fim de apoiar a avaliação das arquiteturas recuperadas por ArchMine, foram realizadas extensões no *checklist* da abordagem ArqCheck (BARCELOS, 2006) para a avaliação do atributo de qualidade Reusabilidade, conforme descrito na seção 4.2.6. Uma vez que a abordagem ArqCheck já tinha sido avaliada através de dois estudos de caso (BARCELOS, 2006), o que fornece indícios da sua viabilidade em detectar defeitos em documentos arquiteturais, o objetivo desse estudo é avaliar apenas se as extensões realizadas em ArqCheck possibilitam verificar o atendimento ao requisito de Reusabilidade.

6.4.1 – Planejamento do Estudo

O propósito desse estudo é avaliar a viabilidade das extensões realizadas no *checklist* da abordagem ArqCheck na detecção de discrepâncias na arquitetura que possam vir a comprometer a Reusabilidade dos elementos arquiteturais. Essas discrepâncias são denominadas "defeitos de Reusabilidade" no contexto deste estudo.

Objetivo do Estudo

Analisar as extensões realizadas no *checklist* da abordagem de inspeção arquitetural ArqCheck para a avaliação de elementos arquiteturais quanto ao atendimento do atributo de qualidade Reusabilidade

Com o propósito de caracterizar

Com respeito à viabilidade em identificar discrepâncias na arquitetura que possam comprometer a Reusabilidade dos elementos arquiteturais

Do ponto de vista de inspetores de artefatos de software

No contexto da inspeção de um documento arquitetural que atende aos requisitos mínimos definidos em ArqCheck para viabilizar a inspeção.

Questões

Q1: **Os itens de avaliação que compõem o *checklist* auxiliam os inspetores na identificação de discrepâncias na arquitetura que possam comprometer a Reusabilidade dos elementos arquiteturais?**

M1: para cada item, quantidade de participantes que o utilizou para identificar defeitos de Reusabilidade.

Q2: **O apoio dos itens do *checklist* na identificação de defeitos de Reusabilidade é maior do que na identificação de falso-positivos na arquitetura?**

M2: para cada item do *checklist*, a quantidade de defeitos e a quantidade de falso-positivos identificados.

Seleção do Contexto

O estudo é realizado em ambiente acadêmico e a abordagem é aplicada sobre um documento que representa uma infra-estrutura de Instanciação e Execução de Ambientes de Software, o eSEE (NETO *et al.*, 2004), que foi construído pelo grupo de pesquisa ESE/COPPE/UFRJ, i.e. o grupo de engenharia de software experimental da

COPPE/UFRJ (ESE, 2007). Este documento foi construído de acordo com as recomendações da IEEE 1471, e, portanto, representa a arquitetura a partir de diferentes perspectivas, além de ressaltar o rastreamento entre requisitos funcionais e não-funcionais e os elementos arquiteturais. Apresenta, dessa forma, os requisitos necessários para que a ArqCheck possa ser aplicada.

Seleção dos Participantes

Foram selecionados 5 inspetores com dois perfis, a saber: inspetores com e sem conhecimento do domínio. Os inspetores são estudantes de pós-graduação da COPPE/UFRJ de dois grupos de pesquisa distintos, i.e. engenharia de software experimental (ESE, 2007) e reutilização (REUSE, 2007). Foram selecionados 2 inspetores do grupo de engenharia de software experimental, os quais julgava-se possuir conhecimento do domínio, e 3 inspetores do grupo de reutilização, os quais julgava-se ter menor probabilidade em possuir conhecimento do domínio. Os inspetores de cada grupo foram selecionados por conveniência, em função da sua disponibilidade para participação no estudo. Todos os inspetores já tiveram curso sobre inspeção de software na pós-graduação, sendo que alguns já haviam participado de outras inspeções de documentos de software. Um dos inspetores do grupo de engenharia de software experimental já havia aplicado a abordagem ArqCheck em um outro estudo.

Definição das Hipóteses

Hipótese Nula (H0): não é possível identificar, em um documento arquitetural, defeitos arquiteturais relacionados ao atendimento do atributo de qualidade Reusabilidade utilizando as extensões do *checklist* da abordagem ArqCheck.

Variáveis

As variáveis independentes e dependentes são descritas nas Tabelas 6.13 e 6.14.

Tabela 6.13: Variáveis independentes do terceiro estudo de viabilidade.

Variável	Valores	Descrição
DOC1	Arquitetura do ambiente eSEE	Documento arquitetural a ser inspecionado.
DOC2	Requisitos do ambiente sSEE	Documento de Requisitos referente à arquitetura a ser inspecionada.
TEC	ArqCheck estendida	Extensões na abordagem ArqCheck para a avaliação de Reusabilidade.
EXP	A: acima da média B: na média C: abaixo da média --- 0: não especialista 1: especialista	Caracterização dos participantes em relação às competências necessárias para a aplicação de TEC e em relação ao conhecimento do domínio do problema.

Tabela 6.14: Variáveis dependentes do terceiro estudo de viabilidade.

Variável	Valores	Descrição
TEMP	Inteiro	O tempo gasto por cada participante durante a detecção de defeitos. O tempo deve ser contabilizado em minutos.
DEF	Inteiro	A quantidade de defeitos encontrados por cada participante para cada item do *checklist*.
FAL	Inteiro	A quantidade de falso-positivos encontrados por cada participante para cada item do *checklist*.

Validade dos Resultados

As seguintes ameaças se apresentam em relação à validade do estudo:

Seleção de participantes: não é possível a realização de blocagem, visto que não há uma quantidade suficiente de inspetores com diferentes perfis (i.e. conhecimento no domínio do problema, conhecimento de arquitetura de software, experiência em inspeção de documentos de software e experiência em reutilização) para participação no estudo. Dessa forma, não é possível traçar conclusões definitivas por perfil de participante.

Fatores utilizados no estudo: o estudo experimental utiliza um único documento arquitetural como objeto de inspeção. Portanto, o estudo pode não ser representativo da teoria sob a qual se sustenta, uma vez que pode não ficar claro se a causa dos resultados do estudo é decorrente da aplicação das extensões de ArqCheck ou da "facilidade" ou "dificuldade" de detecção de defeitos no documento utilizado. Assim, é necessária a realização de mais estudos sobre as extensões de ArqCheck em outros documentos arquiteturais.

Desempenho dos participantes: como forma de evitar possíveis influências no desempenho dos participantes, o objetivo do estudo não foi apresentado aos participantes. Além disso, foi solicitado aos participantes que a comunicação entre eles fosse evitada durante a inspeção.

Perfil dos participantes: os participantes são alunos de pós-graduação que, em sua maioria, possuem pouca experiência de desenvolvimento de software na indústria. Além disso, eles recebem cursos de engenharia de software, arquitetura, inspeção e reutilização de software na pós-graduação, conhecimentos estes não detidos pela maioria dos profissionais da indústria de software. Pesquisas têm demonstrado que muitos dos conceitos de engenharia de software ainda não estão difundidos

adequadamente na indústria de software brasileira (VILLELA, 2004). Portanto, a fim de que os resultados da aplicação das extensões de ArqCheck possam ser generalizados a nível industrial, é necessário que um novo estudo em ambiente industrial seja realizado.

Interação entre ambiente e tratamento: o fato do documento arquitetural ter sido desenvolvido no mesmo ambiente universitário que as extensões do *checklist* da abordagem ArqCheck pode representar uma ameaça à generalização dos resultados do estudo. Entretanto, o *checklist* foi estendido por um grupo de pesquisa, i.e. o grupo de reutilização da COPPE/UFRJ, e o documento arquitetural criado por um outro grupo, i.e. o grupo de engenharia de software experimental da COPPE/UFRJ.

6.4.2 – Execução do Estudo Experimental

Preparação

A fim de permitir a avaliação da arquitetura para o requisito não-funcional Reusabilidade, o *checklist* foi instanciado para o ambiente eSEE, tendo sido preenchidos o requisito não-funcional de Reusabilidade e o cenário de qualidade de Reusabilidade (Anexo C/C.3).

Operação

Os artefatos necessários para que a inspeção fosse realizada foram distribuídos aos participantes (i.e. inspetores). Cada inspetor recebeu um pacote de documentos, contendo: o formulário de consentimento (Anexo C/C.1), o questionário de caracterização (Anexo C/C.2) – adaptado dos estudos realizados em (BARCELOS, 2006), o *checklist* de avaliação de ArqCheck instanciado contendo apenas as questões referentes à avaliação do atributo de qualidade Reusabilidade (Anexo C/C.3), um documento descrevendo a abordagem de documentação arquitetural utlizada pelo eSEE, o documento arquitetural do ambiente de experimentação eSEE modificado pelo pesquisador, o documento de requisitos do ambiente de experimentação eSEE, o relatório de discrepâncias de ArqCheck (Anexo C/C.4), o questionário de avaliação pós-experimento (Anexo C/C.5) - utilizado nos estudos de BARCELOS (2006) - e um roteiro contendo instruções para a realização da inspeção (Anexo C/C.6).

A inspeção foi conduzida de forma remota, ou seja, os participantes não realizaram a inspeção ao mesmo tempo e nem no mesmo local. Contudo, eles foram informados que comentários entre eles deveriam ser evitados a fim de não prejudicar a validade do estudo. Foi também solicitado que eles realizassem a inspeção em uma única sessão e marcassem o tempo utilizado. Os 3 primeiros participantes não

receberam treinamento na utilização de ArqCheck, apenas o pacote de documentação mencionado. Entretanto, como eles tiveram dificuldades em realizar a inspeção e precisaram tirar dúvidas com o pesquisador, os 2 participantes seguintes receberam treinamento presencial aplicado pelo pesquisador.

Após todos os inspetores terem retornado os resultados da inspeção, o pesquisador, atuando no papel de moderador da inspeção, realizou a consistência dos defeitos identificados, gerando um relatório de discrepâncias geral. Neste relatório, defeitos identificados por mais de um inspetor foram adicionados apenas uma vez, apontando-se as questões do checklist que levaram à identificação do defeito. Em seguida, uma reunião de discriminação dos defeitos foi realizada entre o pesquisador (moderador), inspetores e especialistas da aplicação. O objetivo desta reunião era classificar as discrepâncias identificadas pelos inspetores em defeitos ou falso-positivos.

A reunião de discriminação gerou os seguintes resultados: uma lista de defeitos de Reusabilidade consolidada, melhorias que poderiam ser realizadas na arquitetura do eSEE com base nos defeitos identificados e melhorias que poderiam ser realizadas no checklist. Convém ressaltar que algumas das discrepâncias identificadas pelos inspetores não representavam defeitos de Reusabilidade, mas defeitos na representação da arquitetura. Embora correções tenham sido feitas no documento arquitetural, algumas inconsistências na representação da arquitetura permaneceram e, embora o objetivo das questões do checklist não fosse detectar esse tipo de inconsistência, elas acabaram sendo detectadas pelos inspetores em função da sua experiência em arquitetura de software ou em inspeção. Como o checklist será utilizado de forma completa em outros estudos, ou seja, as questões originais do checklist de ArqCheck, que identificam defeitos na consistência da representação arquitetural, serão acrescidas das questões referentes à Reusabilidade, acredita-se que esse tipo de problema não venha a ocorrer novamente.

6.4.3 – Análise dos Resultados Obtidos

O principal objetivo desse estudo é avaliar se as extensões realizadas no *checklist* original da abordagem ArqCheck, para a avaliação de Reusabilidade, de fato ajudam a identificar defeitos sob essa perspectiva em um documento arquitetural. Para tal, foi avaliado o apoio que esses itens de avaliação oferecem à detecção de defeitos de Reusabilidade. Seguem os resultados obtidos nessa avaliação.

Q1: Os itens de avaliação que compõem o *checklist* auxiliam os inspetores na identificação de defeitos de Reusabilidade?

A fim de verificar se os itens de avaliação de Reusabilidade do *checklist* de fato apoiaram os inspetores na detecção desse tipo de defeito arquitetural, foi analisado o relatório de discrepâncias de cada participante. Para cada participante, os defeitos identificados a partir de um determinado item do *checklist* foram classificados em defeitos ou falso-positivos. Esta classificação foi realizada comparando-se as discrepâncias identificadas pelo participante com as discrepâncias contidas no relatório de discrepâncias consolidado após a reunião de discriminação. O gráfico da Figura 6.2, a seguir, demonstra para cada item do *checklist* a quantidade de participantes que o utilizou para identificar defeitos de Reusabilidade na arquitetura.

Convém ressaltar que dos 5 inspetores que participaram do estudo, apenas as respostas de 4 inspetores foram consideradas na avaliação dos resultados do estudo. Um dos inspetores, o que já havia participado de um estudo anterior da abordagem ArqCheck, não identificou defeitos de Reusabilidade, mas somente defeitos de consistência do documento arquitetural. Dessa forma, percebe-se que o fato dele ter participado de um estudo anterior da ArqCheck causou um viés nesse estudo. Entretanto, não se pode generalizar que esse viés sempre vá ocorrer, visto que apenas 1 inspetor com este perfil participou do estudo.

Analisando o gráfico da Figura 6.2, percebe-se que os itens 1 e 6 do *checklist* foram utilizados por 3 participantes para identificar defeitos, ou seja 75% dos inspetores, e o item 4 foi utilizado por 2 participantes, ou seja, 50% dos inspetores, o que fornece indícios do apoio desses itens à detecção de defeitos de Reusabilidade. Os itens 3 e 5 foram utilizados por apenas 1 participante para identificar defeitos. Porém, isso se justifica visto que esses itens foram os que levaram à identificação de um percentual menor de defeitos de Reusabilidade na arquitetura do eSEE, ou seja, 6% dos defeitos apenas foram identificados através do item 3 e 12% de defeitos através do item 5. O item 2 não foi utilizado por nenhum participante para identificar defeitos, o que pode indicar uma falta de clareza neste item ou o fato dele não ser um identificador de defeitos para essa arquitetura.

Q2: O apoio dos itens do *checklist* na identificação de defeitos de Reusabilidade é maior do que na identificação de falso-positivos na arquitetura?

Dentre as discrepâncias identificadas pelos inspetores, 81% consistem de fato em defeitos arquiteturais e, dentre esses, 71% são defeitos de Reusabilidade.

Porém, após uma análise das discrepâncias realizada na reunião de discriminação, uma nova consolidação das mesmas foi realizada, tendo sido identificadas discrepâncias que, embora com descrições distintas e caracterizadas por itens distintos do *checklist*, representavam o mesmo defeito. Isso se deve ao fato de alguns itens do *checklist* apresentarem redundâncias que devem ser avaliadas. Dessa forma, o índice de acerto do *checklist* na identificação de defeitos, após a reunião de discriminação, caiu para 76%, com um percentual de detecção de defeitos de Reusabilidade de 65%. O percentual geral de falso-positivos foi de 24%.

Considerando o desempenho individual de cada inspetor, das discrepâncias identificadas por cada um deles, em média 81% representam defeitos na arquitetura, sendo 68% defeitos de Reusabilidade. O gráfico da Figura 6.3 apresenta a margem de acerto na detecção de defeitos por Inspetor.

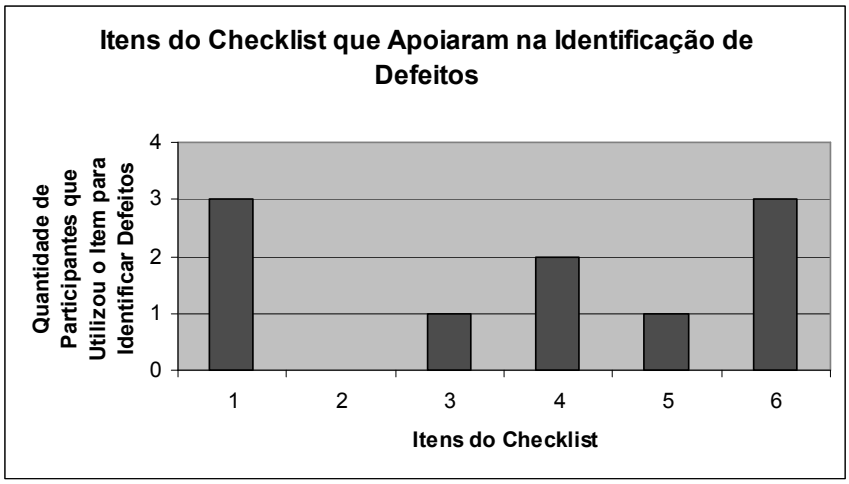

Figura 6.2: Itens do *Checklist* que apoiaram a identificação de defeitos de Reusabilidade.

Todos os inspetores identificaram mais defeitos de Reusabilidade do que outros tipos de defeitos ou falso-positivos, conforme pode ser visto no gráfico da Figura 6.3. Após a consolidação dos dados coletados por inspetor, foi possível realizar uma consolidação de defeitos por item do *checklist*. O gráfico da Figura 6.4, a seguir, apresenta para cada questão do *checklist* os seguintes itens: a quantidade total de discrepâncias identificadas, a quantidade de discrepâncias identificadas que representam

defeitos, a quantidade de defeitos de Reusabilidade e a quantidade de discrepâncias identificadas que representam falso-positivos.

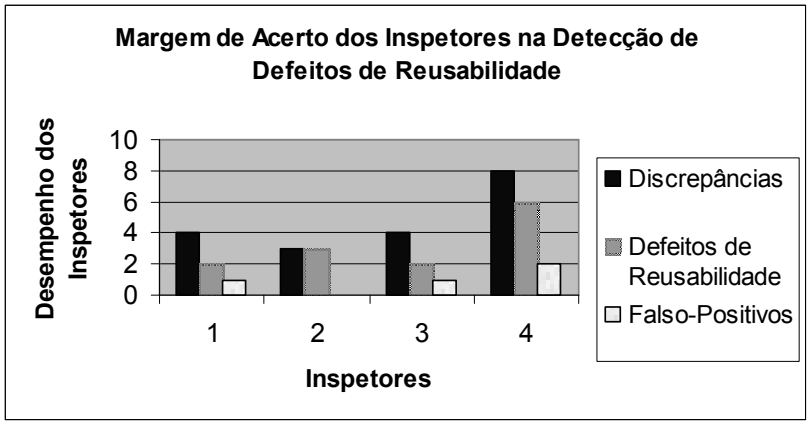

Figura 6.3: Margem de acerto dos inspetores

Em relação ao gráfico da Figura 6.4, convém ressaltar que a diferença entre os "Defeitos Gerais" e os "Defeitos de Reusabilidade" se encontra nos outros tipos de defeitos detectados na arquitetura. A partir de uma análise do gráfico da Figura 6.4, é possível confirmar que o item 2 não levou à identificação de defeitos para a arquitetura do eSEE, o que pode indicar que ele não era um identificador de defeitos para essa arquitetura ou que ele não estava claro. O item 1 levou apenas à detecção de defeitos de Reusabilidade e não de outro tipo de defeito, mas também levou à detecção de um grande percentual de falso-positivos, necessitando ser revisto. Os itens 4, 5 e 6 apresentaram um bom desempenho na detecção de defeitos de Reusabilidade. Já o item 3 precisa ser revisto, pois leva à identificação da mesma quantidade de defeitos de Reusabilidade que de falso-positivos, além de levar à detecção de um percentual maior de outros tipos de defeitos do que defeitos de Reusabilidade.

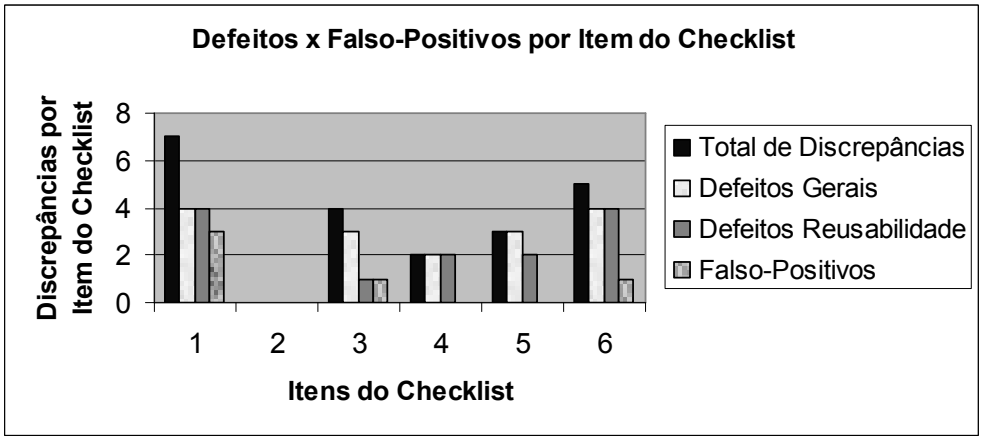

Figura 6.4: Defeitos x falso-positivos por item do *checklist* .

Considerações em Relação à Hipótese Nula

A fim de refutar a hipótese nula, dois tipos de dados coletados no estudo são considerados: dados quantitativos e dados qualitativos. Dados quantitativos, conforme apresentado na seção anterior de Análise dos Resultados, são obtidos a partir da análise dos relatórios de discrepâncias dos inspetores. Os dados qualitativos, por sua vez, são obtidos com base na análise dos questionários de pós-experimento (Anexo C/C.5), preenchidos pelos inspetores, e com base no retorno obtido de especialistas da aplicação e dos inspetores durante a reunião de discriminação.

Em relação à análise quantitativa, através de uma análise do gráfico da Figura 6.2, verifica-se que a maior parte dos itens do *checklist* apoiou a detecção de defeitos de Reusabilidade no documento arquitetural. Analisando o gráfico da Figura 6.3, é possível verificar que os participantes identificaram mais defeitos de Reusabilidade do que outros tipos de defeitos ou falso-positivos na arquitetura. Finalmente, a partir de uma análise do gráfico da Figura 6.4, é possível identificar que a maior parte dos itens do *checklist* leva à detecção de um percentual maior de defeitos de Reusabilidade na arquitetura do que de falso-positivos.

Em relação à análise subjetiva, como resultado da reunião de discriminação, os especialistas consideraram que o *checklist* ajuda a identificar defeitos de Reusabilidade na arquitetura, embora sejam necessárias revisões em algumas das questões. Analisando os questionários de avaliação pós-experimento, percebe-se que todos os inspetores concordam que o *checklist* os apoiou na detecção de defeitos na arquitetura.

Assim, com base nas análises realizadas, não é possível refutar totalmente a hipótese nula, i.e. de que as extensões realizadas no *checklist* de ArqCheck não ajudam a identificar defeitos de Reusabilidade na arquitetura, principalmente em função de 3 aspectos: o pequeno número de participantes no estudo que não permite obter dados estatísticos mais significativos; o fato de ter sido utilizado somente um documento arquitetural, que dificulta a generalização dos resultados; e o fato do número total de defeitos de Reusabilidade na arquitetura não ser conhecido, o que dificulta a avaliação da eficácia da detecção. As análises realizadas oferecem alguns indícios de que as extensões do *checklist* podem ser úteis para apoiar a detecção de defeitos de Reusabilidade em documentos arquiteturais, porém também indicam revisões necessárias no *checklist* e a necessidade da realização de novos estudos para se confirmar esses indícios.

6.4.4 – Lições Aprendidas

Embora não tenha sido possível realizar blocagem dos participantes em função do pequeno número de inspetores, verifica-se que dentre os inspetores, aquele que identificou o maior número de defeitos de Reusabilidade faz parte do grupo que recebeu treinamento para a utilização da técnica. Na realidade, todos os inspetores ressaltaram a importância da realização de um treinamento para a correta aplicação de ArqCheck. Esse treinamento poderia inclusive apontar que respostas no *checklist* indicam a existência de defeitos na arquitetura, pois os inspetores tiveram dificuldade em saber se a sua resposta ao item do *checklist* indicava a presença de defeitos ou não. Essa mesma dificuldade ocorreu em um estudo anterior de ArqCheck, tendo sido acrescentado no *checklist* original um campo "RE" (resultado esperado) (BARCELOS, 2006). Esse campo não foi usado nesse estudo por julgar-se que ele induziria o inspetor à determinada resposta ao item do *checklist.* Entretanto, com base na análise do comportamento dos inspetores durante o estudo, julgamos que algum indicativo a respeito das respostas aos itens do *checklist* seja necessária.

O mesmo inspetor que detectou um maior número de defeitos também se qualificou no questionário de caracterização (Anexo C/C.2) como apresentando maior conhecimento no domínio que os outros, o que confirma a importância desse tipo de conhecimento para o sucesso de uma inspeção (SHULL *et al.*, 2000). Finalmente, em relação ao perfil dos participantes, considerando a experiência sobre Reutilização de software, verifica-se que os inspetores com maior experiência tiveram um maior percentual de acerto na detecção de defeitos de Reusabilidade.

Em relação ao tempo, os participantes levaram em média 1h30m para a realização da inspeção, ou seja, metade do tempo médio de um dos estudos de ArqCheck relatados em (BARCELOS, 2006). Entretanto, o número de questões no estudo relatado em (BARCELOS, 2006) era 7 vezes maior do que nesse estudo, o que revela um elevado tempo na execução desse estudo. Isso se deve, principalmente, ao fato de que a avaliação de itens relacionados ao atendimento a atributos de qualidade como Modificabilidade e Reusabilidade requerem uma avaliação mais subjetiva e menos direcionada na arquitetura do que itens que avaliam a consistência da representação arquitetural, por exemplo.

Nesse contexto, os inspetores apontaram os seguintes itens do *checklist* como subjetivos ou redundantes:

- Vários inspetores consideraram que as questões 1 e 6 apresentam redundâncias, levando, em alguns casos, à identificação dos mesmos defeitos.
- A questão 1 precisa ser revista, pois está muito genérica.
- A questão 4 pode ser melhorada pois o conceito de "alto acoplamento" é subjetivo.
- Os conceitos de coesão e acoplamento podem ficar mais claros nas questões.
- A questão 2 não levou à identificação de defeitos nesse documento arquitetural do eSEE.

Além das questões do *checklist*, os inspetores identificaram possibilidades de melhoria em outros aspectos da abordagem, como na "Guia de Identificação de Contexto" e na taxonomia de defeitos do *checklist*. Em relação à "Guia de Identificação de Contexto", todos os inspetores consideraram que o seu apoio foi neutro para a identificação de defeitos de Reusabilidade. A dificuldade de interpretação da Guia pode ser melhorada através do treinamento na aplicação da abordagem. Em relação à taxonomia de defeitos, o objetivo inicial no estudo era que todos os defeitos de Reusabilidade fossem enquadrados na categoria "Ambigüidade" (ver taxonomia de defeitos no Anexo C/C.4), em função da seguinte definição presente nesta taxonomia:

" Quando a forma como os elementos arquiteturais ou suas responsabilidades foram definidos dificulta ou impossibilita o atendimento a um requisito de qualidade."

Entretanto, isso não foi intuitivo para os inspetores, dos quais somente um conseguiu chegar nesta categorização. Todos os demais não conseguiram enquadrar os defeitos identificados em nenhuma categoria e tiveram que recorrer ao pesquisador para tirar dúvidas. Além disso, todos os inspetores concordaram que a taxonomia deveria ser estendida para as questões de reutilização, pois ela acaba levando à identificação de outros tipos de defeitos que não de reutilização.

Na realidade, tanto a taxonomia de defeitos quanto a "Guia de Identificação de Contexto" foram elaboradas em função das questões originais do *checklist*, sendo necessário revisá-las também quando extensões são realizadas no *checklist* de ArqCheck. Embora a "Guia de Identificação de Contexto" vise ser genérica à sua aplicação para diferentes atributos de qualidade, bastando descrevê-la em função do

requisito, ela ainda não havia sido utilizada em um contexto de Reusabilidade, o que pode justificar alguma dificuldade na sua interpretação.

Por fim, alguns inspetores consideraram que o documento arquitetural do eSEE não apresentava um nível de detalhe suficiente na descrição dos elementos arquiteturais para a aplicação do *checklist*.

Com base nos resultados desse estudo, o *checklist* para a avaliação de Reusabilidade foi refinado. A questão 6 foi eliminada, em função das suas redundâncias com a questão 1. Na questão 1, foi introduzido o conceito de coesão funcional, visando torná-la mais clara. O conceito de alto acoplamento foi retirado da questão 4 por ser subjetivo. Assim, fica a cargo do especialista julgar se os relacionamentos entre o elemento arquitetural reutilizável e demais elementos da arquitetura se justificam.

O ideal seria rodar um novo estudo experimental com o *checklist* refinado para o mesmo documento arquitetural, a fim de verificar se os refinamentos levam à identificação de um número maior de defeitos. Esse estudo deveria contar, obviamente, com a participação de um novo grupo de inspetores para evitar o viés do conhecimento do documento arquitetural e da abordagem. Além disso, seria interessante também realizar estudos de viabilidade com o *checklist* de Reusabilidade em outros documentos arquiteturais mais detalhados, principalmente em ambiente industrial, a fim de generalizar os resultados do estudo. Contudo, como o objetivo principal dos estudos experimentais nesta tese é avaliar a viabilidade da abordagem de recuperação de arquitetura ArchMine, novos estudos sobre o *checklist* de Reusabilidade não foram priorizados. Assume-se, então, os riscos de utilizar o *checklist* estendido e refinado no último estudo experimental de ArchMine, sendo contudo solicitado um *feedback* aos participantes sobre a sua aplicação.

6.5 – Estudo de Viabilidade da Etapa de Avaliação de ArchMine

O objetivo deste último estudo de viabilidade é verificar se a utilização da abordagem de avaliação de arquitetura, i.e. ArqCheck (BARCELOS, 2006) estendida e adaptada, proporciona a melhoria da etapa de avaliação em ArchMine. Essa melhoria é avaliada sob dois aspectos: redução do esforço de avaliação e melhoria da qualidade da arquitetura para reutilização.

Convém ressaltar que ArqCheck foi estendida nesta tese para a avaliação de

arquiteturas quanto ao atendimento ao atributo de qualidade Reusabilidade. Além desse aspecto, ArqCheck avalia também a consistência das representações no documento arquitetural, o atendimento a requisitos funcionais e o atendimento a outros requisitos não-funcionais de interesse. Neste estudo, o *checklist* de ArqCheck foi configurado para a avaliação da arquitetura recuperada através da categorização das questões do *checklist* original como aplicáveis ou não. As questões do *checklist* também foram adaptadas ao documento arquitetural recuperado com ArchMine. Além dessas questões, as questões para a avaliação de Reusabilidade foram acrescentadas.

O estudo foi realizado para uma aplicação de uso industrial, o que possibilita a verificação da eficácia de ArchMine em um contexto diferente daquele no qual foi desenvolvida.

6.5.1 – Planejamento

Diante do cenário exposto, o objetivo deste quarto estudo de caso é avaliar a viabilidade da incorporação de ArqCheck, estendida e adaptada, na etapa de avaliação arquitetural de ArchMine, para a redução do esforço de avaliação e melhoria da qualidade da arquitetura recuperada para a reutilização dos seus elementos arquiteturais.

Objetivo do Estudo

Analisar a etapa de avaliação arquitetural de ArchMine realizada através da abordagem ArqCheck estendida e adaptada

Com o propósito de caracterizar

Com respeito à redução do esforço de avaliação e melhoria da qualidade da arquitetura recuperada para reutilização

Do ponto de vista de engenheiros de software

No contexto da recuperação da arquitetura de um *framework* OO escrito em Java e utilizado em contexto industrial

Questões

Q1: **O esforço de avaliação da arquitetura foi reduzido em relação aos estudos anteriores com a aplicação do *checklist* estendido da abordagem ArqCheck?**

M1.1: tempo gasto na avaliação da arquitetura pelos inspetores comparada com o tempo gasto na avaliação pelos especialistas nos estudos de viabilidade 1 e 2.

M1.2: número de inspetores que respondeu que o grau de dificuldade de aplicação do *checklist* é fácil ou mediano.

Q2: **O *checklist* apóia a avaliação da arquitetura recuperada no sentido de melhorar a sua qualidade para reutilização?**

M2: número de inspetores que respondeu que a arquitetura reestruturada em função das discrepâncias identificadas conduz a elementos arquiteturais que podem ser utilizados para compor uma arquitetura de referência para o domínio.

Seleção do Contexto

A recuperação da arquitetura é realizada para o *framework* CSBase (LIMA *et al.*, 2006), desenvolvido em parceria entre a academia e a indústria. O CSBase é um *framework* para o gerenciamento de recursos e execução de algoritmos em um ambiente computacional distribuído e foi desenvolvido pelo TecGraf, o Grupo de Tecnologia em Computação Gráfica da universidade PUC-Rio (TECGRAF, 2007), em parceria com a Petrobras - Petróleo Brasileiro S/A (PETROBRAS, 2007). Para a execução do *framework* e recuperação da sua arquitetura, a sua instância para grades computacionais, o CSGrid (LIMA *et al.*, 2005), foi utilizada. Essa instância é uma instância para testes do *framework*, não apresentando uso real. Além dessa instância, o CSBase é utilizado ainda nos seguintes projetos na Petrobras: WebSintesi - Sistema Integrador de Aplicação de Geofísica, VGE - Visualizador da Gestão da Exploração, SPROD – Sistema de Informações de Produção, InfoGrid - Sistema de Apoio à Execução de Métodos Científicos, GeoRisco - Sistema de Controle e Análise de Dados de Geotecnia e MARLIM - Sistema de Simulação de Reservatórios.

A seleção dessa aplicação para o estudo se deu em função da sua utilização em ambiente industrial e do fato de ter sido desenvolvida pensando em reutilização, o que propicia a verificação da eficácia da aplicação do *checklist* estendido de ArqCheck nesse contexto.

O *framework* CSBase possui 1214 classes e foi executado através da sua instância CSGrid, que não está em uso em ambiente industrial, tendo sido desenvolvida apenas para testes do *framework*. A recuperação da arquitetura se concentra no *framework* CSBase, que se encontra em uso em ambiente industrial. O CSGrid acrescenta apenas uma classe ao *framework*, uma vez que este se encontra quase em estado executável.

O CSBase é uma aplicação cliente-servidor, escrita em Java, que utiliza RMI (*Remote Method Invocation*) (DEITEL e DEITEL, 2002) para a comunicação remota.

Seleção dos Participantes

O próprio pesquisador aplicou a sua abordagem de recuperação de arquitetura sobre o *framework* CSBase. Especialistas da aplicação, selecionados pela TecGraf, atuaram como inspetores na avaliação da arquitetura recuperada. Os especialistas também apoiaram a etapa de preparação dos dados, envolvendo a definição e o monitoramento dos cenários de casos de uso.

Os especialistas têm pelo menos 7 anos de experiência em desenvolvimento de software na prática e boa experiência também em projeto arquitetural. Em relação à reutilização de software, um dos participantes selecionados tem bastante experiência e o outro uma experiência média. Ambos têm um bom conhecimento do domínio do CSBase.

Definição das Hipóteses

Hipótese em relação ao esforço de avaliação:

Hipótese Nula ($H0_1$): A incorporação do *checklist* estendido de ArqCheck para a avaliação de arquiteturas recuperadas com ArchMine não reduz o esforço da avaliação da arquitetura recuperada.

Hipótese em relação à qualidade da arquitetura para reutilização:

Hipótese Nula ($H0_2$): A incorporação do *checklist* estendido de ArqCheck para a avaliação de arquiteturas recuperadas com ArchMine não apóia a melhoria da qualidade da arquitetura recuperada para a sua reutilização.

Variáveis

As variáveis independentes, que impactam os resultados do estudo, são apresentadas na Tabela 6.15. A Tabela 6.16 apresenta as variáveis dependentes, que apresentam o efeito da aplicação de ArchMine sobre o CSBase.

Validade do Estudo

As seguintes ameaças à validade do estudo foram identificadas:

Fatores utilizados no estudo: o estudo experimental utilizou um único documento arquitetural para avaliação. Portanto, o estudo pode não ser representativo da teoria sob a qual se sustenta, uma vez que pode não ficar claro se a causa dos resultados do estudo é decorrente da aplicação de ArqCheck, estendida e adaptada, ou

da "facilidade" ou "dificuldade" de detecção de defeitos na arquitetura recuperada. O ideal seria utilizar mais de um sistema no estudo, mas em função do esforço para a realização do mesmo, essa opção não é viável em um contexto industrial.

Tabela 6.15: Variáveis independentes do quarto estudo de viabilidade.

Variável	Valores	Descrição
APL	CSBase	Aplicação sobre a qual ArchMine é aplicada.
CEN	Cenários	Cenários especificados para o CSBase.
CNF	Real	Valor de confiança mínima aplicado na mineração dos rastros de execução.
SUP	Real	Valor de suporte mínimo aplicado na mineração dos rastros de execução.
ANT	Antecedentes	Classes selecionadas como antecedentes para a mineração.
ARM	ArchMine	Abordagem de recuperação de arquitetura utilizada.
EXP	A: acima da média B: na média C: abaixo da média --- 0: não especialista no domínio 1: especialista no domínio	Caracterização dos participantes em relação às competências necessárias para apoio no estudo.

Tabela 6.16: Variáveis dependentes do quarto estudo de viabilidade.

Variável	Valores	Descrição
ARQ	Arquitetura Recuperada	Arquitetura recuperada para a aplicação CSBase aplicando ArchMine refinada.
TMP	Real	Tempo médio de avaliação da arquitetura recuperada pelos inspetores aplicando ArqCheck estendida e adaptada.
NUM1	Inteiro	Número de inspetores que respondeu que o grau de dificuldade de aplicação do *checklist* é fácil ou mediano.
NUM2	Inteiro	Número de inspetores que respondeu que a arquitetura reestruturada a partir dos defeitos identificados com o *checklist* poderia ser reutilizada para compor uma arquitetura de referência para o domínio.

Visões arquiteturais recuperadas: como o monitoramento dos cenários e coleta dos rastros de execução não pôde ser realizado com o apoio da ferramenta Tracer, diagramas de seqüência a nível arquitetural não puderam ser extraídos. O monitoramento foi realizado com a ferramenta JProfiler (EJ-TECHNOLOGIES, 2007). A ferramenta Tracer apresentou limitações para o monitoramento de uma aplicação cliente-servidor, utilizando RMI para a comunicação remota. Dessa forma, o uso do *checklist* de ArqCheck ficou restrito, pois algumas questões de avaliação da consistência entre diferentes visões arquiteturais não puderam ser aplicadas.

Viés das avaliações realizadas: a fim de verificar se a utilização de ArqCheck estendida e adaptada de fato ajuda a reduzir o esforço de avaliação de arquitetura em ArchMine, seria necessário colocar duas equipes de avaliação com o mesmo perfil

avaliando a mesma arquitetura com a ArqCheck e com os formulários utilizados nos estudos anteriores e medir o tempo de avaliação. Entretanto, como é objetivo deste estudo também corrigir a arquitetura a fim de verificar o quanto as correções, com base nos defeitos identificados com o *checklist,* conduzem a uma arquitetura reutilizável, os dois inspetores disponíveis para o estudo utilizaram o mesmo *checklist* para a avaliação. Dessa forma, a avaliação do esforço comparada com o esforço de avaliação dos estudos anteriores oferece algum indicativo da melhora ou piora nesse sentido, mas não representa uma conclusão definitiva a esse respeito dadas as variações das características das aplicações e do perfil dos inspetores.

6.5.2 – Execução do Estudo Experimental

Preparação

Nesta etapa, os participantes do estudo (i.e. inspetores) assinaram o formulário de consentimento para apoio no estudo (Anexo D/D.1). Eles apoiaram tanto a fase de preparação dos dados para a recuperação dos elementos arquiteturais quanto a avaliação da arquitetura recuperada. A preparação dos dados envolveu a definição dos cenários de caso de uso da aplicação e a sua execução monitorada para a coleta dos rastros de execução.

Como o CSBase é um sistema distribuído, que utiliza RMI, não foi possível utilizar a ferramenta Tracer para o seu monitoramento, tendo sido utilizada uma outra ferramenta que realiza o monitoramento de aplicações Java em tempo de execução, a JProfiler (EJ-TECHNOLOGIES, 2007). A limitação da utilização do Tracer se deu em função do seu mecanismo de carga dinâmica de recursos, que apresentou incompatibilidade com o RMI.

Foram definidos e monitorados 105 cenários no cliente e 24 no servidor, sendo alguns desses redundantes. Embora JProfiler esteja preparada para monitorar aplicações distribuídas, ela não coleta chamadas de métodos e eventos disparados no cliente e no servidor ao mesmo tempo, uma vez que esses rodam em máquinas virtuais distintas. O JProfiler até monitora mais de uma máquina virtual ao mesmo tempo, mas não agrega os resultados. Isso gerou a necessidade de replicar alguns cenários no monitoramento do cliente e do servidor e apresentou algum impacto nos resultados, conforme discutido nas Lições Aprendidas.

Além dessa limitação, a JProfiler não permite informar o cenário de caso de uso sendo monitorado, como a Tracer, uma vez que seu objetivo é avaliar o desempenho e

uso de recursos da aplicação. Dessa forma, um esforço maior foi exigido no monitoramento e coleta dos dados dos cenários. A JProfiler também não exporta, junto com os rastros de execução, todas as informações necessárias para a extração dos diagramas de seqüência pela Phoenix, descrita na seção 5.3.5, como a hierarquia de chamadas de métodos, o valor da instância e a thread. Dessa forma, os diagramas de seqüência não puderam ser extraídos para o CSBase, o que limitou o uso do checklist de ArqCheck neste estudo.

Além da definição dos cenários de caso de uso e coleta dos rastros de execução, a preparação dos dados também envolveu a ER estática do diagrama de classes da aplicação no ambiente OdysseyLight. Essa recuperação do diagrama de classes foi realizada com a ferramenta de ER do ambiente, a Ares.

Finalmente, o *checklist* de ArqCheck foi configurado para o CSBase, através dos seguintes passos:

- alteração dos termos utilizados para termos compatíveis com o modelo arquitetural recuperado por ArchMine;
- eliminação das questões não aplicáveis a uma documentação arquitetural recuperada com ArchMine;
- acréscimo das questões referentes à avaliação de Reusabilidade, refinadas com base nos resultados do estudo de viabilidade número 3;
- instanciação do requisito e cenário de Reusabilidade.

A versão do checklist utilizada neste estudo pode ser obtida no Anexo D/D.4. Convém ressaltar que, embora questões não aplicáveis à documentação arquitetural recuperada com ArchMine tenham sido eliminadas do checklist, algumas questões ainda foram classificadas como "não aplicáveis – NA". Isso se deve ao fato dos diagramas de seqüência (modelo dinâmico) não terem sido recuperados e ao fato do CSBase não apresentar uma documentação de requisitos anterior à recuperada com ArchMine. As colunas marcadas com "Def." se referem à resposta ao item do checklist que indica a presença de um defeito na arquitetura.

Operação

Uma vez preparados os artefatos de entrada, o pesquisador realizou a recuperação da arquitetura do CSBase, através da abordagem ArchMine. O modelo

arquitetural recuperado com ArchMine foi exportado para o OdysseyLight para avaliação pelos especialistas.

Os inspetores receberam o material do estudo, contendo: o *checklist* de avaliação da arquitetura (Anexo D/D.4), o relatório de discrepâncias (Anexo C/C.4), o questionário de caracterização (Anexo D/D.2), os formulários de avaliação pós-experimento 1 e 2 (anexos D/D.5 e D/D.6), além da abordagem de documentação arquitetural utilizada (Anexo D/D.3). Foram elaborados dois formulários de avaliação pós-experimento, cada um preenchido em um determinado momento, a saber: o primeiro quando da finalização do preenchimento do *checklist* e relatório de discrepâncias, e o segundo após a correção da arquitetura com base nos defeitos identificados na inspeção. Além do material, um treinamento presencial foi dado aos especialistas a respeito da inspeção do modelo arquitetural aplicando o *checklist* de ArqCheck estendido e adaptado. Explicações foram dadas sobre a forma de avaliar os itens do *checklist*, destacando-se os itens referentes ao atendimento do atributo de qualidade Reusabilidade, onde os inspetores receberam explicações a respeito da interpretação do cenário de reusabilidade e da "Guia de Identificação de Contexto".

Os inspetores avaliaram a arquitetura recuperada, preencheram o material e passaram para o pesquisador. Algumas dúvidas surgiram durante a inspeção e os inspetores contactaram o pesquisador por e-mail ou telefone para tirar dúvidas. Convém ressaltar que os inspetores foram instruídos a classificar os defeitos de Reusabilidade como "Ambigüidade". Após finalizada a avaliação, o material foi passado para o pesquisador que consolidou as discrepâncias identificadas e realizou correções na arquitetura com base nas mesmas. Uma reunião de fechamento da inspeção foi agendada para que o pesquisador pudesse tirar dúvidas a respeito de algumas discrepâncias e correções na arquitetura, sendo gerada a arquitetura final após essa reunião.

6.5.3 – Análise dos Resultados Obtidos

A análise dos resultados foi realizada com base nos questionamentos estabelecidos no estudo, conforme se segue.

Q1: **O esforço de avaliação da arquitetura foi reduzido em relação aos estudos anteriores com a aplicação do *checklist* estendido da abordagem ArqCheck?**

No segundo estudo de viabilidade, o especialista da OdysseyLight, que possui 595 classes, gastou cerca de 5 horas na avaliação. Na arquitetura recuperada do OdysseyLight constavam 466 classes. Neste estudo, a arquitetura recuperada constava de 721 classes, apesar das 1214 do CSBase. Isso se deve aos seguintes fatores: existem funcionalidades do *framework* que não são utilizadas pela instância CSGrid; algumas classes são de apoio a testes; algumas superclasses e interfaces não foram alocadas na arquitetura.

Um dos inspetores gastou 7h e 40m na avaliação da arquitetura recuperada para o CSGrid, em diversas sessões de inspeção, e o outro gastou 2h e 45m na avaliação. Entretanto, o primeiro inspetor, que possui maior experiência em inspeção, identificou 80% de defeitos a mais que o segundo. Dessa forma, comparando-se com o esforço de avaliação do estudo anterior, percebe-se que a arquitetura do CSGrid tem um tamanho aproximadamente 35% maior que a do OdysseyLight, e o tempo de inspeção do inspetor que identificou mais defeitos foi aproximadamente 35% maior também. Dessa forma, com base nesse estudo, não há indícios para refutar a hipótese nula ($H0_1$), ou seja, de que a aplicação do *checklist* estendido e adaptado de ArqCheck reduz o esforço de avaliação em ArchMine.

Analisando o esforço com base no grau de dificuldade de utilização do *checklist*, um dos inspetores, o que tem menos experiência em inspeção e identificou menos defeitos, considerou a utilização do *checklist* difícil, e o outro, que tem mais experiência em inspeção e identificou um percentual bem maior de defeitos, considerou o grau de dificuldade de utilização do *checklist* mediano.

Entretanto, os inspetores consideraram que o grau de dificuldade da avaliação se deve também ao fato do ambiente Odyssey não ordenar alfabeticamente as classes, apresentar recursos avançados de busca ou manipulação de modelos. Dessa forma, esses indícios em relação à hipótese nula ($H0_1$) precisam ser melhor investigados, através do planejamento e projeto de novos estudos.

Q2: **O *checklist* apóia a avaliação da arquitetura recuperada no sentido de melhorar a sua qualidade para reutilização?**

Ambos os inspetores responderam que a arquitetura reestruturada em função das discrepâncias identificadas conduz a elementos arquiteturais que podem ser reutilizados

em uma arquitetura de referência de domínio. Assim, do ponto de vista qualitativo, a incorporação de ArqCheck estendida e adaptada à ArchMine traz benefícios, havendo indícios de que é possível refutar a hipótese nula ($H0_2$).

6.5.4 – Lições Aprendidas

As seguintes conclusões foram traçadas ao final do estudo com base nos formulários de avaliação pós-experimento e entrevistas com os especialistas:

- O esforço na avaliação da arquitetura se manteve com a aplicação do *checklist* estendido de ArqCheck. Entretanto, os resultados finais obtidos na arquitetura corrigida representam uma contribuição para a reutilização de software, pois a avaliação pôde ser mais bem direcionada a esse objetivo e à eliminação de inconsistências na arquitetura. Além disso, é necessária a realização de novos estudos, utilizando o mesmo sistema e duas abordagens de avaliação (i.e. ArqCheck estendida e uma abordagem *ad-hoc*), além de inspetores com o mesmo perfil aplicando os 2 tratamentos, para confirmar esse resultado.
- A análise de um sistema cliente-servidor como o CSBase fica um pouco prejudicada em função da ferramenta não ser capaz de fazer a análise conjunta. De fato, foram feitas análises separadas, uma monitorando o cliente e outra monitorando o servidor, sendo que os casos de uso foram repetidos em cada uma. Esse procedimento é trabalhoso, propício a erros (em função da repetição manual do caso de uso) e leva ao surgimento de elementos distintos que, em uma análise conjunta, seriam um só.
- Como toda a comunicação entre os processos se dá através de RMI, o que implica em interfaces remotas explícitas, é possível que a coleta de informações seja executada, tanto no cliente quanto no servidor, simultaneamente, e, depois, os rastros de execução sejam agregados. Essa agregação poderia ser feita por meio da identificação da chamada remota, usando a interface do serviço exportado via RMI.
- Os nomes dos componentes ajudaram bastante na identificação dos componentes. A abordagem de geração dos nomes dos elementos arquiteturais foi qualificada como boa pelos especialistas.
- Um dos inspetores considerou o nível de granularidade dos elementos arquiteturais recuperados pequeno. Entretanto, conforme mencionado na seção 2.2.1, não existe um consenso na comunidade de Engenharia de Software sobre

o nível de granularidade adequado para se descrever arquiteturas de software. Além disso, o objetivo de ArchMine é recuperar elementos arquiteturais que sejam candidatos a componentes de software do domínio e isso justifica o baixo nível de granularidade obtido.

- O esforço de monitoramento dos cenários para a coleta de rastros de execução foi bem maior na ferramenta JProfiler do que teria sido na ferramenta Tracer. Além disso, a ferramenta JProfiler não recupera a hierarquia de chamadas de métodos, com suas instâncias, como a Tracer, impossibilitando a reconstrução dos diagramas de seqüência pela Phoenix.
- A utilização de JProfiler no lugar da Tracer demonstrou que a mineração e reconstrução dos elementos arquiteturais em TraceMining é independente de ferramenta de monitoramento, bastando que informações sobre cenários de casos de uso e classes relacionadas sejam coletadas.
- ArchMine se mostrou escalável, partindo da recuperação da arquitetura de uma aplicação com 51 classes (a TIM (DANTAS *et al.*, 2006)), no primeiro estudo de viabilidade, até a recuperação da arquitetura do CSBase, envolvendo 721 classes. O tempo de mineração foi de 2 horas, bem mais eficiente do que o tempo gasto no primeiro estudo experimental com a mineração dos rastros de uma aplicação de 596 classes (a Odyssey-PSW (CORREA e WERNER, 2006)), que foi de 30 horas. Percebe-se que os refinamentos em ArchMine, ao longo dos estudos, melhoraram a sua escalabilidade.
- As diretrizes para a definição de cenários de casos de uso e a análise de cobertura de classes em ArchMine permitem obter uma boa cobertura das classes do sistema na arquitetura recuperada. Neste estudo, observou-se que as classes não monitoradas, em sua maioria, não faziam parte da instância do *framework* utilizado. Outras eram superclasses e interfaces que não são monitoradas por não receberem mensagens diretamente. Embora utilizando a Odyssey-Metrics para apoiar a alocação dessas entidades na arquitetura, observa-se que uma maior automação dessa etapa do processo de ArchMine ainda se faz necessária.

6.6 – Considerações Finais

Neste capítulo, foram apresentados os estudos experimentais realizados no

contexto desta tese. Os estudos apontaram indícios da viabilidade da abordagem de recuperação de arquitetura ArchMine, tendo sido fundamentais para o seu refinamento ao longo deste processo de pesquisa. Foi possível perceber, ao longo da realização desses estudos, a importância em se avaliar uma tecnologia, método ou processo na Engenharia de Software antes que o mesmo seja transferido para um contexto industrial. Podemos dizer, após essa experiência, que o processo de avaliação é "vital" para o amadurecimento de uma abordagem ou confirmação de uma teoria em qualquer área tecnológica ou científica.

Conforme apresentado no capítulo 4, o processo proposto nesta tese engloba duas sub-abordagens, a saber: recuperação de modelos de sistemas legados (ER), i.e. ArchMine, e detecção de variabilidades entre os modelos estruturais recuperados, a fim de apoiar a criação de uma arquitetura de referência de domínio, i.e. ArchToDSSA. Dessa forma, estudos experimentais que confirmem a viabilidade da segunda etapa do processo proposto nesta tese ainda se fazem necessários. Esses serão conduzidos no contexto do trabalho de mestrado de KÜMMEL (2007).

Além desse estudo sobre a detecção de variabilidades, alguns outros estudos ainda poderiam ser conduzidos, a fim de propiciar novas melhorias na abordagem ArchMine e nos seus sub-produtos, sendo listados alguns que seriam de grande valia, como se segue:

- Estudos de Observação[12], com outros engenheiros de software aplicando a abordagem ArchMine, a fim de verificar a sua usabilidade.
- Variação dos valores de variáveis independentes, como a confiança mínima, e avaliação do seu impacto na qualidade da arquitetura recuperada.
- Avaliação do esforço de uma tarefa de manutenção, através do suporte oferecido pela TraceMining (visão estática) e Phoenix (visão dinâmica), comparado com o esforço da mesma tarefa de manutenção realizada através de depuração de código ou com diagramas extraídos por outras abordagens, ambos os cenários sendo realizados por equipes de mantenedores com o mesmo perfil.

[12] Estudo em ambiente onde os indivíduos desempenham tarefas enquanto são observados pelos pesquisadores (BARCELOS, 2006).

- Verificação da viabilidade de LegaToDSSA como um todo, envolvendo as etapas de recuperação e comparação de arquiteturas, para a criação de arquiteturas de referência de domínio.

Convém ressaltar que realizamos os estudos que julgamos prioritários para sustentar a teoria defendida nesta tese acerca da combinação da ER com a mineração de dados para a extração de conhecimento útil sobre sistemas legados, que possa apoiar empresas na migração para abordagens de reutilização. Nesse sentido, foi possível verificar a viabilidade de ArchMine e a viabilidade da reconstrução de elementos arquiteturais que podem ser utilizados para compor uma arquitetura de referência de domínio, através da combinação de ArchMine com ArqCheck estendida.

Vale ressaltar a importância de terem sido realizados estudos envolvendo, inicialmente, sistemas desenvolvidos no mesmo contexto de pesquisa desta tese, i.e. sistemas do grupo de reutilização da COPPE/Sistemas, e em seguida, com a abordagem mais amadurecida, estudos com sistema de utilização em ambiente industrial. A transferência de uma tecnologia imatura para a indústria pode levar a investimento de esforço e recursos desnecessários, acarretando em prejuízos. Os estudos produziram resultados que nos incentivam a continuar investindo nessa pesquisa, sendo ressaltada a necessidade de repetição de alguns estudos, principalmente o último, em que é avaliada a incorporação de ArqCheck estendida para a avaliação das arquiteturas recuperadas com ArchMine, em outros contextos, com outros sistemas e envolvendo participantes com diferentes perfis.

Os estudos experimentais realizados nos permitem verificar que nesta tese contribuições podem ser dadas tanto na área de ER quanto na área de reutilização de software.

Capítulo 7 – Conclusão

7.1 – Epílogo

Esta tese apresentou uma abordagem de apoio à criação de arquiteturas de referência de domínio, que pode ser aplicada nos contextos de ED e LP, denominada LegaToDSSA. LegaToDSSA é composta de 2 etapas, a saber: recuperação de arquitetura de sistemas legados em um domínio de aplicação e comparação das arquiteturas para a detecção de opcionalidades e variabilidades e criação de arquiteturas de referência.

Conforme apresentado, os métodos de ED e LP costumam oferecer um maior apoio à especificação de arquiteturas de referência através de abordagens de engenharia progressiva. Entretanto, como os sistemas legados disponíveis representam uma das mais ricas fontes de informações no domínio, além de representarem grande investimento de recursos das empresas, a análise desses para a criação de arquiteturas de referência é de grande valia para as organizações que desenvolvem sistemas de software similares. Abordagens de ED e LP que focam na análise de sistemas legados, em geral, não oferecem um apoio sistemático a todas às etapas do processo, desde a extração de informações dos sistemas até a geração das arquiteturas de referência, ou focam em algum aspecto específico, como a extração de componentes de software de código legado.

Uma vez que os sistemas legados costumam ser complexos, mal projetados e não possuem documentação de apoio atualizada, o maior esforço de pesquisa e desenvolvimento nesta tese se concentrou na área de ER, com foco em recuperação de arquiteturas, tendo sido desenvolvida uma abordagem de recuperação de arquitetura voltada à reutilização, denominada ArchMine. As arquiteturas recuperadas são avaliadas quanto à consistência da sua representação e Reusabilidade dos elementos arquiteturais através de ArqCheck adaptada e estendida. Finalmente, ArchToDSSA, que representa a última fase da abordagem proposta, envolve a comparação das arquiteturas recuperadas para a detecção das suas opcionalidades e variabilidades, permitindo a criação de arquiteturas de referência parcialmente especificadas. Essas arquiteturas podem ser exportadas para o ambiente de reutilização Odyssey, onde existe a

possibilidade do mapeamento dos elementos arquiteturais para características do domínio, a partir das quais novas aplicações no domínio são instanciadas.

7.2 – Contribuições

Esta tese apresenta contribuições nas áreas de Reutilização de Software, Engenharia Reversa e Arquitetura de Software. As contribuições são descritas de acordo com essas áreas, conforme se segue:

Reutilização de Software

As seguintes contribuições se apresentam na área de Reutilização de Software:

1) *Uma abordagem sistemática de análise de sistemas legados para a criação de arquiteturas de referência de domínio*: a abordagem proposta, LegaToDSSA, envolve etapas bem definidas, executadas através de processos com critérios, métodos e técnicas estabelecidos para apoiar a criação de arquiteturas de referência a partir da análise de sistemas legados, cobrindo desde uma abordagem de ER voltada à recuperação de arquitetura, combinada a uma abordagem de avaliação de arquiteturas, até uma abordagem de comparação de modelos estruturais.

2) *Recuperação de elementos arquiteturais que representam conceitos do domínio*: ArchMine recupera elementos arquiteturais que correspondem a conceitos do domínio, conforme foi verificado no último estudo experimental. O agrupamento de classes, através da análise dos cenários que elas implementam, e a estratégia de derivação de nomes, permite alcançar esse objetivo.

3) *Apoio ferramental integrado a um ambiente de reutilização de software*: foi desenvolvido apoio ferramental à abordagem LegaToDSSA, cobrindo as etapas de recuperação de arquitetura, comparação das arquiteturas e geração de arquiteturas de referência de domínio no Odyssey. O ambiente Odyssey apresenta uma infra-estrutura de apoio à reutilização, permitindo que os modelos gerados por LegaToDSSA sejam reutilizados em desenvolvimentos futuros.

4) *Comparação de arquiteturas de diferentes sistemas para a detecção de variabilidades e opcionalidades*: a abordagem ArchToDSSA contribui com a reutilização de software, através da detecção de elementos mandatórios e opcionais em um conjunto de arquiteturas de domínio, além de pontos de variação com as suas variantes. A abordagem permite a comparação de modelos arquiteturais de diferentes aplicações. As abordagens de cálculo de diferenças de

modelos pesquisadas permitem a comparação de diferentes versões de um mesmo modelo arquitetural. Dessa forma, essas abordagens não levam em conta questões tratadas em ArchToDSSA, como a incorporação de um dicionário de sinônimos e comparação por *substrings*, para lidar com a diferença de nomenclatura.

Engenharia Reversa e Manutenção de Software

As seguintes contribuições se apresentam nas áreas de ER e Manutenção de Software:

1) *Apoio à compreensão de software, através da recuperação de diferentes visões arquiteturais*: ArchMine, além de apoiar a reutilização de software, apóia também a manutenção de sistemas legados, uma vez que recupera modelos estáticos e dinâmicos atualizados dos sistemas, com rastros para os requisitos funcionais, permitindo a sua compreensão. O modelo estático permite a compreensão dos conceitos que o sistema compreende. Por outro lado, se uma funcionalidade do software precisa ser adaptada ou evoluída, o rastro dos elementos arquiteturais e classes para os requisitos funcionais, visualizada através de diagramas de seqüência, permite a localização dos elementos que podem ser impactados pela manutenção.
2) *Recuperação de diagramas de seqüência de forma iterativa e incremental*: a fim de oferecer um maior potencial de apoio à compreensão e manutenção de software, os diagramas de seqüência podem ser reconstruídos de forma iterativa e incremental, em um processo de busca gradativa de conhecimento sobre os sistemas. Os diagramas podem ser reconstruídos a nível arquitetural, i.e. demonstrando trocas de mensagens entre elementos arquiteturais, e detalhados através de diagramas em nível de classe, onde trocas de mensagens entre classes de um mesmo elemento arquitetural podem ser visualizadas. Além disso, os diagramas podem ser reconstruídos até um determinado nível de profundidade de mensagens, onde uma chamada de método em um nível "n" de profundidade pode ser detalhada em um diagrama a parte.
3) *Critérios genéricos para a reconstrução dos elementos arquiteturais*: ArchMine é uma abordagem de recuperação de arquitetura que apresenta critérios para a reconstrução de elementos arquiteturais independentes de

linguagem de programação e de domínio. Dessa forma, o objetivo de propor critérios genéricos para a reconstrução de elementos arquiteturais é atendido. Entretanto, ArchMine requer algum conhecimento da aplicação para a definição dos cenários de casos de uso e está voltada ao apoio da recuperação de arquitetura de sistemas OO.

4) *Apoio ferramental e integração a um ambiente de desenvolvimento*: foi desenvolvido um apoio ferramental às atividades de ArchMine integrado a um ambiente de desenvolvimento de software baseado em reutilização, o que permite a visualização e manipulação dos modelos recuperados, além da possibilidade da sua reutilização em novos esforços de desenvolvimento.
5) *Incorporação de uma abordagem de avaliação arquitetural*: ArchMine incorpora a abordagem de avaliação, ArqCheck, permitindo que a avaliação das arquiteturas recuperadas não seja realizada de forma aleatória, como em outras abordagens de ER, mas que seja realizada de forma direcionada aos objetivos da recuperação.
6) *Estudos experimentais que sugerem indícios da viabilidade de ArchMine:* estudos experimentais foram realizados no contexto desta tese, sugerindo indícios da viabilidade da abordagem de recuperação de arquitetura ArchMine no que tange à recuperação de um modelo arquitetural compreensível para sistemas OO escritos em Java.
7) *Sugestão de um pacote de experimentação:* o processo de experimentação seguido para avaliar a abordagem de recuperação de arquitetura, ArchMine, sugere um protocolo de pesquisa que poderia guiar a avaliação de novas tecnologias dessa natureza, iniciando pela avaliação da viabilidade da abordagem em ambiente acadêmico, passando por um estudo comparativo da abordagem e chegando a um estudo de avaliação da sua viabilidade em ambiente industrial.

Arquitetura de Software

As seguintes contribuições se apresentam na área de Arquitetura de Software:

1) *Extensão de uma abordagem de avaliação de arquiteturas*: nesta tese, a abordagem de avaliação de arquitetura ArqCheck foi estendida para a avaliação do requisito não-funcional Reusabilidade, além de ter sido adaptada para a representação arquitetural recuperada com ArchMine. Dessa forma, verificou-se a viabilidade de extensão e adaptação de ArqCheck.

2) *Estudos experimentais para avaliar a viabilidade das questões de avaliação de Reusabilidade*: estudos experimentais foram realizados sobre ArqCheck estendida, oferecendo indícios da possibilidade de avaliação de Reusabilidade através das extensões realizadas no *checklist*.

7.3 – Limitações

Esta seção apresenta as limitações em relação à abordagem proposta, a LegaToDSSA.

As seguintes limitações foram identificadas durante essa pesquisa:

1. *A coleta de rastros de execução exige grande esforço*. A coleta dos cenários de casos de uso é realizada de forma mecânica com o apoio da ferramenta Tracer. É necessário habilitar a coleta de rastros antes do início da execução de um cenário de caso de uso e desabilitá-la ao final. Dessa forma, um grande esforço é exigido nessa atividade. Esse esforço se justifica quando funcionalidades pontuais devem ser localizadas no código para fins de manutenção, ou quando uma empresa deseja migrar para uma abordagem de ED e LP e os modelos extraídos podem ser reutilizados em novos esforços de desenvolvimento.
2. *Dificuldade da recuperação da arquitetura de sistemas distribuídos*: a análise de um sistema de processos distribuídos, como um sistema cliente-servidor, fica prejudicada em função da ferramenta Tracer não ser capaz de fazer o monitoramento conjunto desses processos. Nesses casos, é necessário realizar análises separadas, monitorando o cliente e o servidor, sendo que os casos de uso devem ser repetidos em cada uma. Esse procedimento é trabalhoso, propício a erros (em função da repetição manual do caso de uso) e leva ao surgimento de elementos arquiteturais distintos que, em uma análise conjunta, seriam um só. Além disso, nesse caso, os diagramas de seqüência reconstruídos não demonstram adequadamente aspectos de concorrência e sincronização entre os processos.
3. *Dificuldade na representação de concorrência utilizando diagramas de seqüência da UML 1.4*: embora ArchMine recupere diagramas dinâmicos, i.e. de seqüência, multi-thread, o comportamento concorrente não é representado de forma clara com a sintaxe da UML 1.4. A adoção

da UML 2.0, com seus recursos para representação de comportamento concorrente em diagramas de seqüência, poderia ajudar a solucionar esse problema.

4. *Impacto dos cenários definidos, antecedentes selecionados e valor de confiança e suporte mínimos na qualidade da arquitetura recuperada*: os cenários definidos para a coleta de rastros de execução, os antecedentes selecionados e os valores de suporte mínimo e confiança mínima adotados na mineração apresentam grande impacto na qualidade da arquitetura recuperada com ArchMine. Entretanto, diretrizes e heurísticas foram definidas para apoiar o processo ao longo dos estudos realizados, visando minimizar esses impactos. Além disso, o conhecimento humano no processo é importante para minimizar esses impactos. Convém ressaltar que outras abordagens de recuperação de arquitetura, como as descritas em (SARTIPI, 2003; MITCHELL e MANCORIDIS, 2006), também sofrem impacto em relação aos dados de entrada e aleatoriedade do algoritmo, podendo gerar diferentes modelos arquiteturais em diferentes execuções do algoritmo.
5. *Dificuldade de recuperação da arquitetura quando existe conhecimento embutido em regras de negócio armazenadas no banco de dados*: a recuperação da arquitetura com ArchMine pode ficar prejudicada quando o sistema possui conhecimento sobre o negócio embutido em regras de negócio armazenadas no SGDB (Sistema Gerenciador de Banco de Dados) na forma de *triggers* e *storage procedures*. Esse conhecimento não pode ser recuperado no monitoramento da execução da aplicação com a ferramenta Tracer.
6. *Esforço na avaliação das arquiteturas recuperadas*: a avaliação através de ArqCheck leva a checagens na arquitetura que podem exigir grande esforço quando o modelo arquitetural é extenso. Em (BARCELOS, 2006), é relatado nos resultados de um dos estudos experimentais que os inspetores julgaram como mediana a utilização da abordagem ArqCheck, justamente pela dificuldade em se utilizar essa abordagem na avaliação de documentos arquiteturais grandes. Isso se torna mais grave no caso das arquiteturas recuperadas com ArchMine em função do seu nível de granularidade. Além disso, algumas questões incluídas para a avaliação

de Reusabilidade, como as que avaliam acoplamento, apresentam alguma subjetividade, aumentando o esforço. Esse fato ocorre também com algumas questões de avaliação de requisitos não-funcionais no *checklist* original. Embora, nesse último caso, as questões sejam em sua maioria mais concretas.

7. *Versão da UML e XMI*: em 2002, quando esse trabalho de pesquisa foi iniciado, a versão corrente da especificação da UML era a 1.4. A partir desse momento, outras versões surgiram, estando atualmente a linguagem na versão 2.0. O XMI se encontra na versão 2.1. Entretanto, a abordagem LegaToDSSA ainda trabalha com a versão 1.4 da UML e 1.2 do XMI, que são as versões utilizadas pela versão atual do ambiente Odyssey. Dessa forma, o XMI gerado para as arquiteturas recuperadas ou modelos de domínio, somente podem ser lidos por ambientes de desenvolvimento ou ferramentas que trabalhem com essas mesmas versões de UML/XMI. Isso se deve principalmente ao fato do repositório MOF utilizado, o MDR, ainda não ter evoluído para a versão atual do XMI e da UML.
8. *Generalidade de ArchMine*: embora generalidade tenha sido um requisito buscado nesta tese para uma abordagem de recuperação de arquitetura de software, observa-se que, assim como outras abordagens da literatura, ArchMine atinge uma generalidade parcial. A abordagem é voltada à análise de sistemas OO e o seu ferramental apóia a análise de sistemas escritos em Java.

7.4 – Trabalhos Futuros

Algumas oportunidades de melhorias e possibilidades de extensão desta pesquisa foram identificadas ao longo do trabalho, podendo ser tratadas através de trabalhos futuros, como se segue:

1. *Definição automatizada de cenários de casos de uso*: seria interessante encontrar uma forma de definição dos cenários e coleta de rastros de execução menos mecânica do que a que é realizada atualmente pela Tracer. Uma possibilidade seria sugerir a definição de um novo cenário de caso de uso de forma automática a cada evento disparado na interface gráfica,

permitindo alguma forma de filtro desses eventos por parte do desenvolvedor.

2. *Extensão da abordagem para sistemas distribuídos e sistemas Web*: uma possibilidade de pesquisa interessante é estender ArchMine para sistemas distribuídos e sistemas Web, verificando a adequação ou necessidade de extensão das heurísticas nesses cenários e estendendo e/ou desenvolvendo ferramental de apoio.
3. *Adoção da UML 2.0 para a representação dos modelos recuperados*: a UML 2.0 apresenta evoluções significativas particularmente nos modelos de componentes e de seqüência. Com a adoção da UML 2.0, seria possível representar de forma mais clara o comportamento concorrente, decisões e laços nos diagramas de seqüência. Porém, seria necessário adotar outra tecnologia, caso o MDR não evolua para essa versão.
4. *Exploração de outras técnicas de mineração de dados*: uma possibilidade de trabalho futuro seria a exploração de outras técnicas de mineração de dados, como algoritmos de *clustering* (MIRKIN, 2005), comparando a eficácia dos resultados obtidos com os resultados obtidos com o Apriori.
5. *Reengenharia de OO para componentes de software*: uma vez que, nesta tese, são recuperados elementos arquiteturais funcionalmente coesos e que representam conceitos do domínio, implementando um conjunto comum de funcionalidades ou serviços, observa-se uma oportunidade da reengenharia desses elementos para componentes de software do domínio. A fim de que os elementos arquiteturais recuperados com ArchMine possam se tornar componentes de software de fato, um trabalho de reengenharia, evolvendo possivelmente um conjunto de refatorações sobre a arquitetura recuperada seria necessário. Refatorações envolvendo, por exemplo, a transformação de herança entre classes de diferentes elementos arquiteturais em delegação. Além disso, as interfaces dos componentes de software precisariam ser derivadas. Esse trabalho de reengenharia para componentes de software sobre o modelo arquitetural recuperado com ArchMine está sendo desenvolvido por um trabalho de mestrado em fase inicial no grupo de reutilização da COPPE/UFRJ.
6. *Evolução da arquitetura recuperada*: à medida que modificações são realizadas no código ou na arquitetura recuperada para um sistema legado, é

necessário que o modelo recuperado com ArchMine se mantenha atualizado. Algumas possibilidades se apresentam para a manutenção dessa atualização, como: adotar uma ferramenta como ArchTrace (MURTA *et al.*, 2006), sendo possível manter ligações de rastreabilidade entre a arquitetura e o código fonte à medida que modificações sejam realizadas; aplicar a heurística **H6** de ArchMine, que envolve o agrupamento de classes não agrupadas, para o agrupamento de classes que porventura venham a ser criadas ou modificadas após a recuperação da arquitetura; definir novas heurísticas para manter a arquitetura atualizada em relação ao código quando classes são criadas, atualizadas ou removidas.

7. *Possibilidade de extensão da ER a outros modelos dinâmicos*: com base nos rastros de execução coletados, outros modelos dinâmicos da UML, como diagramas de estado, poderiam ser recuperados.
8. *Possibilidade de desenvolvimento de outras estratégias de recuperação de arquitetura*: com os insumos gerados pelo ferramental desenvolvido nesta tese, como os rastros de execução, novas estratégias de recuperação de arquitetura podem ser desenvolvidas. É possível, por exemplo, desenvolver uma abordagem de detecção de estilos arquiteturais, como camadas, extraindo métricas de mensagens enviadas diretamente por cada classe. Dessa forma, é possível encontrar um grupo de classes que juntas enviam mensagens para um outro grupo específico. Colocando classes destino de envio de mensagens em um espaço vetorial de características para classes que enviam mensagens, talvez fosse possível descobrir similaridades entre grupos de classes, caracterizando a camada a qual pertencem na arquitetura.
9. *Comparação de outros modelos para a detecção de diff, além do modelo arquitetural estático*: ArchToDSSA poderia calcular **diff** também sobre outros modelos recuperados com ArchMine, como modelos de seqüência e de casos de uso, facilitando a geração desses modelos para o nível do domínio.
10. *Elaboração de um modelo de ontologia para padronizar os termos do domínio*: uma possibilidade alternativa ao uso do dicionário de sinônimos para a comparação das arquiteturas em ArchToDSSA seria a elaboração de um modelo de ontologia de domínio, onde os conceitos das arquiteturas recuperadas fossem padronizados antes da realização da sua comparação.

Esta tese apresentou a abordagem LegaToDSSA, que envolve o apoio à criação de arquiteturas de referência de domínio, através da recuperação de arquiteturas de sistemas legados em um domínio, sua comparação e derivação de arquiteturas de referência parcialmente especificadas. LegaToDSSA representa um avanço para abordagens de reutilização de ED e LP, permitindo que a sua adoção leve as organizações a utilizar mais efetivamente a sua base de software instalada, que representa recurso investido e conhecimento do negócio que não pode ser perdido.

LegaToDSSA propõe um processo sistemático de análise de sistemas legados para a criação de DSSA, que, além de contribuir com a Reutilização de software, contribui também para o avanço das áreas de Engenharia Reversa, Manutenção e Arquitetura de Software, como apresentado na seção 7.2.

Finalmente, a combinação de técnicas e princípios dessas diferentes áreas, i.e. Engenharia Reversa, Manutenção, Reutilização e Arquitetura de Software, é que dão à LegaToDSSA a capacidade de atingir seus objetivos e prover um apoio à especificação de arquiteturas de referência, ainda não encontrado nas abordagens de ED e LP pesquisadas até o momento.

Referências Bibliográficas

ABI-ANTOUN, M., ALDRICH, J., NAHAS, N.*, et al.*, 2006, "Differencing and Merging of Architectural Views". In: *21st IEEE/ACM International Conference on Automated Software Engineering 2006, ASE'06*, pp. 47-58, Tokyo, Japan, September.

AGRAWAL, R., LIN, K., SAWHNEY, H.S.*, et al.*, 1995, "Fast Similarity Search in the Presence of Noise, Scaling, and Translation in Time-Series Database". In: *21st Very Large Databases Conference*, pp. 490-501, Zurich, Switzerland, September.

AGRAWAL, R., SRIKANT, R., 1994, "Fast Algorithms for Mining Association Rules". In: *20th Very Large Databases Conference*, pp. 487-499, Santiago, Chile, September.

AGRAWAL, R., SRIKANT, R., 1995, "Mining Sequential Patterns". In: *International Conference on Data Engineering (ICDE'95)*, pp. 3-14, Taipei, Taiwan, March.

ALANEN, M., PORRES, I., 2003, "Difference and Union of Models". In: *6th International Conference on The Unified Modeling Language, Modeling Languages, and Applications (UML 2003)*, pp. 2-17, San Francisco, California, USA, April.

ALDRICH, J., CHAMBERS, C., NOTKIN, D., 2002, "ArchJava: Connecting Software Architecture to Implementation". In: *24th International Conference on Software Engineering (ICSE 2002)*, pp. 187-197, Orlando, Florida, USA, May.

ALVES, V., MATOS JR, P., COLE, L.*, et al.*, 2007, "Extracting and Evolving Code in Product Lines with Aspect-Oriented Programming", *Transactions on Aspect-Oriented Software Development (TAOSD): Special Issue on Software Evolution, To Appear*.

ALVES, V., MATOS JR., P., COLE, L.*, et al.*, 2005, "Extracting and Evolving Mobile Games Product Lines". In: *9th International Software Product Line Conference (SPLC'05)*, v. 3714 of Lecture Notes in Computer Science, pp. 70-81, Rennes, France, September.

AMARAL, E.A.G., 2003, *Empacotamento de Experimentos em Engenharia de Software*, Dissertação de M.Sc., Programa de Engenharia de Sistemas e Computação - COPPE, UFRJ, Rio de Janeiro, Brasil.

ANQUETIL, N., 2000, *Curso de Engenharia Reversa, notas de aula*, COPPE/UFRJ.

ANQUETIL, N., FOURRIER, C., LETHBRIDGE, T.C., 1999, "Experiments with Hierarchical Clustering Algorithms as Software Remodularization Methods". In: *Working Conference on Reverse Engineering*, pp. 235-255, Pittsburgh, PA, USA, October.

ANQUETIL, N., LETHBRIDGE, T., 1999, "Recovering Software Architecture from the Names of Source Files", *Journal of Software Maintenance: Research and Practice, John Wiley & Sons Ltd.*, v. 11, pp. 201-221.

ANQUETIL, N., LETHBRIDGE, T.C., 2003, "Comparative Study of Clustering Algortihms and Abstract Representations for Software Remodularization", *IEE Software*, v. 150, n. 3 (June), pp. 185-201.

ANTONIOL, G., GUEHENEUC, Y.G., 2005, "Feature identification: A Novel Approach and a Case Study". In: *21st International Conference on Software Maintenance*, pp. 357-366, Budapest, Hungary, September.

ASPECTJ, 2007, "Eclipse Project, AspectJ 1.5.3". In: http://eclipse.org/aspectj/, accessed in February 10.

ATKINSON, C., BAYER, J., BUNSE, C., *et al.*, 2002, *Component-based Product Line Engineering with UML*, Boston, Addison-Wesley Longman Publishing Co., Inc.

BAEZA-YATES, R., RIBEIRO-NETO, B., 1999, *Modern Information Retrieval*, ACM Press.

BARCELOS, R., TRAVASSOS, G.H., 2006, "ArqCheck: Uma Abordagem para Inspeção de Documentos Arquiteturais baseada em Checklist". In: *V Simpósio Brasileiro de Qualidade de Software (SBQS)*, pp. 174-188, Vila Velha, ES, Brasil, Maio-Junho.

BARCELOS, R.F., 2006, *Uma Abordagem para Inspeção de Documentos Arquiteturais Baseada em Checklist*, Dissertação de M.Sc., Programa de Engenharia de Sistemas e Computação - COPPE, UFRJ, Rio de Janeiro, Brasil.

BASILI, V., CALDIEIRA, G., ROMBACH, H., 1994, *Goal Question Metrics Paradigm, Encyclopedia of Software Engineering*, John Wiley & Sons.

BASS, L., CLEMENTS, P., KAZMAN, R., 2003, *Software Architecture in Practice*, 2nd ed., Addison-Wesley.

BERRY, M.J.A., LINOFF, G., 1997, *Data Mining Techniques*, John Wiley & Sons, Inc.

BIGGERSTAFF, T.J., MITBANDER, B.G., WEBSTER, D.E., 1994, "Program Understanding and the Concept Assignment Problem", *Communications of the ACM*, v. 37, n. 5, pp. 72-82.

BLOIS, A.P.B., 2006, *Uma Abordagem de Projeto Arquitetural Baseado em Componentes no Contexto de Engenharia de Domínio*, Tese de D.Sc., Programa de Engenharia de Sistemas e Computação - COPPE, UFRJ, Rio de Janeiro, Brasil.

BOJIC, D., VELASEVIC, D., 2000, "A Use-Case Driven Method of Architecture Recovery for Program Understanding and Reuse Reengineering". In: *4th European Software Maintenance and Reengineering Conference*, pp. 23-31, Zuriq, Swiss, February/March.

BOOCH, G., RUMBAUGH, J., JACOBSON, I., 1998, *The Unified Modeling Language User Guide*, 1st ed., Addison-Wesley.

BOSCH, J., MOLIN, P., 1999, "Software Architecture Design: Evaluation and Transformation". In: *IEEE Engineering of Computer Based Systems Symposium (ECBS'99)*, pp. 4-10, Nashville, TN, USA, March.

BRAGA, R., 2000, *Busca e Recuperação de Componentes em Ambientes de Reutilização de Software*, Tese de D.Sc., COPPE, UFRJ, Rio de Janeiro, Brasil.

BRIAND, L.C., LABICHE, Y., MIAO, Y., 2003, "Towards the Reverse Engineering of UML Sequence Diagrams". In: *10th IEEE Working Conference on Reverse Engineering*, pp. 57-66, Victoria, Canada, November.

BURMEISTER, P., 1998, *Formal Concept Analysis with ConImp: Introduction to the Basic Features*, 7, 64289 ed., Darmstadt, Germany, Arbeitsgruppe Allgemeine Algebra und Diskrete Mathematik, Technische Hochschule Darmstadt, Schlobgartenstr.

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

CALDIERA, G., BASILI, V.R., 1991, "Identifying and Qualifying Reusable Software Components", *IEEE Computer* (February), pp. 61-70.

CEDERQVIST, P., 2003, *Version Management with CVS*, Free Software Foundation.

CEPEDA, R.S.V., VASCONCELOS, A.P.V., WERNER, C.M.L., 2006, "Tracer e Phoenix: Ferramentas Integradas para a Engenharia Reversa de Modelos Dinâmicos de Java para UML". In: *XX Simpósio Brasileiro de Engenharia de*

Software, XIII Sessão de Ferramentas, pp. 25-30, Florianópolis, SC, Brasil, Outubro.

CHEN, K., RAJLICH, V., 2000, "Case Study of Feature Location Using Dependence Graph". In: *8th International Workshop on Program Comprehension*, pp. 241-249, Limerick, Ireland, June.

CHEN, M.S., HAN, J., YU, P.S., 1996, "Data Mining: an Overview from Database Perspective", *IEEE Transactions on Knowledge and Data Engineering*, v. 8, n. 6, pp. 866-883.

CHEN, P., CRITCHLOW, M., CARG, A.*, et al.*, 2003, "Differencing and Merging within an Evolving Product Line Architecture". In: *International Workshop on Software Product-Family Engineering (PFE-5)*, pp. 269-281, Siena, Italy, November.

CHIDAMBER, S.R., KEMERER, C.F., 1994, "A Metrics Suite for Object Oriented Design", *IEEE Transactions on Software Engineering*, v. 20, n. 6, pp. 476-493.

CHIKOFSKY, E.J., CROSS II, J.H., 1990, "Reverse Engineering and Design Recovery: a Taxonomy", *IEEE Software*, v. 7, n. 1 (January), pp. 13-17.

CLEMENTS, P., BACHMANN, F., BASS, L.*, et al.*, 2004, *Documenting Software Architectures*, Addison-Wesley.

COLLINS-SUSSMAN, B., FITZPATRICK, B.W., PILATO, C.M., 2004, *Version Control with Subversion*, O'Reilly.

CORNELISSEN, B., GRAAF, B., MOONEN, L., 2005, "Identification of Variation Points Using Dynamic Analysis". In: *12th Working Conference on Reverse Engineering, 1st International Workshop on Reengineering Towards Product Lines*, pp. 9-13, Pittsburgh, PA, USA, November.

CORREA, A., WERNER, C.M.L., 2006, "Odyssey-PSW: Uma Ferramenta de Apoio à Verificação e Validação de Especificações e Restrições OCL". In: *XX Simpósio Brasileiro de Engenharia de Software, XIII Sessão de Ferramentas*, pp. 61-66, Florianópolis, Brasil, Outubro.

D'SOUZA, D.F., WILLS, A.C., 1999, *Objects, components, and frameworks with UML: the catalysis approach*, Addison-Wesley Longman Publishing Co., Inc.

DANTAS, A.R., 2001, *Um Sistema de Críticas para a UML*, Projeto Final de Graduação, IM, UFRJ, Rio de Janeiro, RJ, Brasil.

DANTAS, A.R., CORREA, A.L., WERNER, C.M.L., 2001, "Oráculo: Um Sistema de Críticas para a UML". In: *XV Simposio Brasileiro de Engenharia de Software - SBES, Caderno de Ferramentas*, pp. 398-403, Rio de Janeiro, RJ , Brasil.

DANTAS, A.R., VERONESE, G.O., BARROS, M.O.*, et al.*, 2006, "Model Driven Game Development - Experience and Model Enhacements in Software Project Management Education", *Software Process Improvement and Practice*, v. 11, n. 4, pp. 411-421.

DANTAS, C.R., 2005, *Odyssey-WI: Uma Abordagem para a Mineração de Rastros de Modificação de Modelos em Repositórios Versionados*, Dissertação de M.Sc., COPPE, UFRJ, Rio de Janeiro, RJ, Brasil.

DASHOFY, E., HOEK, A., TAYLOR, R., 2002, "An Infrastructure for the Rapid Development of XML-based Architecture Description Languages". In: *24th International Conference on Software Engineering*, pp. 266-276., Orlando, USA, May.

DE PAUW, W., LORENZ, D., VLISSIDES, J.*, et al.*, 1998, "Execution Patterns in Object-Oriented Visualization". In: *4th USENIX Conference on Object-Oriented Technologies and Systems (COOTS)*, pp. 219-234, Santa Fe, New Mexico, April.

DEITEL, H.M., DEITEL, P.J., 2002, *Java: Como Programar*, 4 ed., Bookman.

DEURSEN, A.V., HOFMEISTER, C., KOSCHKE, R.*, et al.*, 2004, "Symphony: View-Driven Software Architecture Reconstruction". In: *Fourth Working IEEE/IFIP Conference on Software Architecture (WICSA 2004)*, pp. 122-132, Oslo, Norway, June.

DIAS, M.S., VIEIRA, M.E.R., 2000, "Software Architecture Analysis based on Statechart Semantics". In: *International Workshop on Software Specification and Design*, pp. 133-137, Shelter Island, San Diego, California, USA, November.

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

DOVAL, D., MANCORIDIS, S., MITCHEL, B.S., 1999, "Automatic Clustering of Software Systems Using a Genetic Algorithm". In: *International Conference on Software Tools and Engineering Practice (STEP'99)*, pp. 73-81, Pittsburgh, PA, August.

DUENAS, J.C., DE OLIVEIRA, W.L., DE LA PUENTE, J.A., 1998, "A Software Architecture Evaluation Model". In: *2nd International ESPRIT ARES Workshop*, pp. 148-157, Las Palmas de Gran Canaria, Spain, February.

EISENBARTH, T., KOSCHKE, R., SIMON, D., 2003, "Locating Features in Source Code", *IEEE Transactions on Software Engineering*, v. 29, n. 3 (March), pp. 210-224.

EISENBERG, A.D., VOLDER, K.D., 2005, "Dynamic Feature Traces: Finding Features in Unfamiliar Code". In: *21st IEEE International Conference on Software Maintenance (ICSM'05)*, pp. 337-346, Budapest, Hungary, September.

EJ-TECHNOLOGIES, 2007, "JProfiler, v. 4.3.1". In: http://www.ej-technologies.com/products/jprofiler, accessed in February, 2007.

ERICKSON, R.L., GRIFFETH, N.D., LAI, M.Y.*, et al.*, 1993, "Software Architecture Review for Telecommunications Software Improvement". In: *IEEE International Conference on Communications*, v. 2, pp. 616-620, Geneva, Switzerland, May.

ESE, 2007, "Grupo de Engenharia de Software Experimental da COPPE/UFRJ". In: http://ese.cos.ufrj.br/ese/, accessed in 15/01/2007.

FAYYAD, U.M., 1996, *Advances in Knowledge Discovery and Data Mining*, Menlo Park, California, USA, MIT Press.

FRAKES, W., KANG, K., 2005, "Software Reuse Research: Status and Future", *IEEE Transactions on Software Engineering*, v. 31, n. 7 (July), pp. 529-536.

FREEMAN, M., 1980, *Reusable Software Engineering: A Statement of Long-Range Research Objectives*, Technical Report TR-159, Universidade da California.

FREEMAN, P., 1987, "Reusable Software Engineering Concepts and Research Directions", *Software Reusability (tutorial), IEEE Computer Society Press*, pp. 129-137.

GALL, H., JAZAYERI, M., KLÖSCH, R., 1996, "Architecture Recovery in ARES". In: *International Software Architecture Workshop (ISAW96)*, pp. 111-115, San Francisco, CA, USA, October.

GALL, H., PINZGER, M., 2002, "Pattern-Supported Architecture Recovery". In: *10th International Workshop on Program Comprehension*, pp. 53-61, Paris, France, June.

GAMMA, E., HELM, R., JOHNSON, R., *et al.*, 1995, *Padrões de Projeto - Soluções Reutilizáveis de Software Orientado a Objetos* Ed. Bookman.

GANESAN, D., KNODEL, J., 2005, "Identifying Domain-Specific Reusable Components from Existing OO Systems to Support Product Line Migration". In: *First International Workshop on Reenegineering Towards Product Lines (R2PL), co-located with the 12th WCRE and the 5th WICSA*, pp. 16-20, Pittsburgh, PA, USA, November.

GIMENES, I.M.S., TRAVASSOS, G.H., 2002, "O Enfoque de Linha de Produto para Desenvolvimento de Software". In: *XXI Jornada de Atualização em Informática (JAI) – Evento Integrante do XXII Congresso da SBC*, pp. 1-31, Florianópolis, Brasil, Julho.

GOMAA, H., 2004, *Designing Software Product Lines with UML: from Use Cases to Pattern-Based Software Architectures*, Addison-Wesley Professional.

GORTON, I., 2006, *Essential Software Architecture*, 1st ed., New York, Springer.

GRISS, M.L., FAVARO, J., D'ALESSANDRO, M., 1998, "Integrating feature modelling with the RSEB". In: *Proceedings of Fifth International Conference on Softwre Reuse - ICSR5*, pp. 76-85 Victoria, British Columbia, Canada, June.

HAMOU-LHADJ, A., LETHBRIDGE, T., 2004, "A Survey of Trace Exploration Tools and Techniques". In: *IBM Centre for Advanced Studies, 2004 Conference of the Centre for Advanced Studies on Collaborative Research*, pp. 42-55, Markham, Ontario, Canada, October.

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

HARRIS, D.R., YEH, A., CHASE, M.P., 1997a, "Manipulating Recovered Software Architecture Views". In: *19th International Conference on Software Engineering*, pp. 184-194, Massachusets, USA, May.

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

HARRIS, D.R., YEH, A., REUBENSTEIN, H.B., *et al.*, 1995, "Reverse Engineering to the Architectural Level". In: *17th International Conference on Software Engineering*, pp. 186-195, Scattle, Washington, USA, April.

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

HOFMEISTER, C., NORD, R.L., D., S., 1999, "Describing Software Architecture with UML". In: *1st Working IFIP Conference on Software Architecture*, pp. 145-160, San Antonio, Texas, USA, February.

HOPKINS, J., 2000, "Component Primer", *Commun, ACM*, v. 43, n. 10, pp. 27-30.

IEEE, 2000, "IEEE Recommended Practice for Architectural Description of Software Intensive Systems - IEEE Standard 1471-2000", *Institute of Electrical and Electronic Engineers*.

IVKOVIC, I., GODFREY, M.W., 2003, "Enhancing Domain-Specific Software Architecture Recovery". In: *International Conference/Workshop on Program Comprehension*, pp. 266-273, Portland, Oregon, USA, May.

JAIN, A.K., 1988, *Algorithms for Clustering Data*, Englewood, Cliffs, N.J., Prentice Hall.

JERDING, D., RUGABER, S., 1997, "Using Visualization for Architecture Localization e Extraction". In: *4th Working Conference on Reverse Engineering*, pp. 56-65, Amsterdam, Holland, October.

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

JSA, 2007, "Journal of Systems Architecture - Elsevier". In: http://www.elsevier.com/wps/find/journaldescription.cws_home/505616/description#description, accessed in January 25.

KALINOWSKI, M., TRAVASSOS, G.H., 2004, "A Computational Framework for Supporting Software Inspections". In: *19th ACM/IEEE International Conference on Automated Software Engineering (Main Conference)*, pp. 46-55, Linz, Austria.

KANG, K.C., COHEN, S.G., HESS, J.A., *et al.*, 1990, *Feature-Oriented Domain Analysis (FODA): Feasibility Study*, Software Engineering Institute (SEI), CMU/SEI-90-TR-21, ESD-90-TR-222.

KANG, K.C., LEE, J., DONOHOE, P., 2002, "Feature-Oriented Product Line Engineering", *IEEE Software*, v. 9, n. 4 (Jul./Aug 2002), pp. 58-65.

KAZMAN, R., BASS, L., G., A., *et al.*, 1994, "SAAM: a Method for Analyzing the Properties of Software Architectures". In: *International Conference on Software Engineering (ICSE)*, pp. 81-90, Sorrento, Italy, May.

KAZMAN, R., CARRIÈRE, S.J., 1997, *Playing Detective: Reconstructing Software Architecture from Available Evidence*, Technical Report CMU/SEI-97-TR-010, CMU/SEI.

KAZMAN, R., KLEIN, M., CLEMENTS, P., 2000, *ATAM: Method for Architecture Evaluation*, CMU/SEI, Technical Report, CMU/SEI-2000-TR-004.

KEIENBURG, F., RAUSCH, A., 2001, "Using XML/XMI for Tool Supported Evolution of UML Models". In: *34th Hawaii International Conference on System Sciences (HICSS)*, pp. 9064-9073, Island of Maui, Hawaii, USA, January.

KELTER, U., WEHREN, J., NIERE, J., 2005, "A Generic Difference Algorithm for UML Models". In: *Software Engineering (SE) 2005*, pp. 105-116, Essen, Germany, March.

KICZALES, G., LAMPING, J., MENDHEKAR, A., *et al.*, 1997, "Aspect-Oriented Programming". In: *European Conference on Object-Oriented Programming (ECOOP)*, pp. 220-242, Jyväskylä, Finland, June.

KITCHENHAM, B., 2004, *Procedures for Performing Systematic Reviews* Technical Report, Keele University.

KLEINBERG, J.M., 1999, "Authoritative Sources in a Hyperlinked Environment", *Journal of the ACM*, v. 46, n. 5, pp. 604-632.

KLEMETTINEN, M., MANNILA, H., TOIVONEN, H., 1997, "A Data Mining Methodology and Its Application to Semi-Automatic Knowledge Acquisition". In: *8th International Workshop on Database and Expert Systems Applications*, pp. 670-677, Tolouse, France, September.

KNODEL, J., MUTHIG, D., 2005, "Analyzing the Product Line Adequacy of Existing Components". In: *First Workshop on Reengineering towards Product Line (R2PL), co-located with the 12th WCRE and the 5th WICSA*, pp. 21-25, Pittsburgh, PA, USA, November.

KNOR, R., TRAUSMUTH, G., WEIDL, J., *et al.*, 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, Spain, February.

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

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

KRUEGER, C.W., 1992, "Software Reuse", *ACM Computing Surveys*, v. 24, n. 2 (June), pp. 131-183.

KÜMMEL, G., 2007, *Abordagem para a Criação de Arquiteturas de Referência de Domínio a partir da Comparação de Modelos Arquiteturais de Aplicações*, Dissertação de M.Sc., COPPE, UFRJ, Rio de Janeiro, Brasil, em fase de conclusão.

KÜMMEL, G.Z., WERNER, C.M.L., 2006, "Comparando Modelos Arquiteturais de Software para Apoiar a Criação de uma Arquitetura de Referência em Domínios de Aplicação Específicos". In: *XX Simpósio Brasileiro de Engenharia de Software, XI Workshop de Teses e Dissertações em Engenharia de Software (WTES)*, pp. 55-60, Florianópolis, SC, Brasil, Outubro.

LEE, K., KANG, K.C., LEE, J., 2002, "Concepts and Guidelines of Feature Modeling for Product Line Software Engineering". In: *Software Reuse: Methods, Techniques, and Tools : 7th International Conference, ICSR-7, Proceedings* pp. 62 - 77, Austin, TX, USA, April.

LICHTNER, K., ALENCAR, P., COWAN, D., 2000, "A Framework for Software Architecture Verification". In: *Australian Software Engineering Conference*, pp. 149-157, Canberra, ACT, Australia, April.

LIMA, M.J.D., MELCOP, T., CERQUEIRA, R.*, et al.*, 2005, "CSGrid: um Sistema para Integração de Aplicações em Grades Computacionais". In: *Salão de Ferramentas do 23o. Simpósio Brasileiro de Redes de Computadores*, Fortaleza, Brasil.

LIMA, M.J.D., URURAHY, C., MOURA, A.*, et al.*, 2006, "CSBase: A Framework for Building Customized Grid Environments". In: *Third International Workshop on Emerging Technologies for Next-generation GRID (ETNGRID 2006)*, Manchester, Reino Unido.

LOPES, M.A.M., 2005, *Um Mecanismo para Apoio à Percepção Aplicado a Modelos de Software Compartilhados*, Dissertação de M.Sc., COPPE, UFRJ, Rio de Janeiro, RJ, Brasil.

MAGICDRAW, 2007, "MagicDraw UML 12.1". In: http://www.magicdraw.com/, accessed in March.

MAIA, N.E.N., 2006, *Uma Abordagem para a Transformação de Modelos*, Dissertação de M.Sc., COPPE, UFRJ, Rio de Janeiro, RJ, Brasil.

MAIA, N.E.N., BLOIS, A.P.B., WERNER, C.M., 2005, "Odyssey-MDA: Uma Ferramenta para Transformação de Modelos UML". In: *XIX Simpósio Brasileiro de Engenharia de Software, Sessão de Ferramentas*, pp. 51-56, Uberlândia, MG, Brasil, Outubro.

MANCORIDIS, S., MITCHELL, B.S., CHEN, Y.*, et al.*, 1999, "Bunch: a Clustering Tool for the Recovery and Maintenance of Software System Structures". In: *International Conference on Software Maintenance (ICSM'99)*, pp. 50-59, Oxford University, England, August.

MANGAN, M.A.S., 2006, *Uma Abordagem para o Desenvolvimento de Apoio à Percepção em Ambientes Colaborativos de Desenvolvimento de Software*, Tese de D.Sc., COPPE, UFRJ, Rio de Janeiro, Brasil.

MATULA, M., 2007, "NetBeans MetaData Repository". In: http://mdr.netbeans.org/, accessed in March 21.

MEDVIDOVIC, N., TAYLOR, R., 2000, "A Classification and Comparison Framework for Software Architecture Description Languages", *IEEE Transactions on Software Engineering*, v. 26, n. 1, pp. 70-93.

MEHRA, A., GRUNDY, J., HOSKING, J., 2005, "A Generic Approach to Supporting Diagram Differencing and Merging for Collaborative Design". In: *20th IEEE/ACM International Conference on Automated Software Engineering (ASE'05)*, pp. 204-213, Long Beach, CA, USA, November.

MELO JR, C.R.S., 2005, *Metrictool: Uma Ferramenta Parametrizável para Extração de Métricas de Projetos Orientados a Objetos*, Projeto Final de Graduação, IM, UFRJ, Rio de Janeiro, Brasil.

MENDES, A., 2002, *Arquitetura de Software: Desenvolvimento Orientado para Arquitetura*, Rio de Janeiro, Brasil, Campus.

MENDONÇA, N.C., 1999, *Software Architecture Recovery for Distributed Systems*, Department of Computing, Imperial College of Science, Technology and Medicine, London, UK.

MENDONÇA, N.C., KRAMER, J., 2001, "An Approach for Recovering Distributed System Architectures", *International Journal of Automated Software Engineering, Kluwer Academic Publishers*, v. 8, n. 3/4, pp. 311-355.

METTALA, E., GRAHAM, M.H., 1992, *The Domain-Specific Software Architecture*, CMU/SEI, SEI Technical Report CMU/SEI-92-SR-009.

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

MIRKIN, B., 2005, *Clustering for Data Mining: A Data Recovery Approach*, Chapman & Hall/CRC.

MITCHELL, B.S., 2002, *A Heuristic Search Approach to Solving the Software Clustering Problem*, PhD, Drexel University.

MITCHELL, B.S., MANCORIDIS, S., 2006, "On the Automatic Modularization of Software Systems Using the Bunch Tool", *IEEE Transactions on Software Engineering*, v. 32, n. 3 (March), pp. 193-208.

MOHAMMAD, E., STROULIA, E., SORENSON, P., 2002, "From Run-Time Behavior to Usage Scenarios: an Interaction-Pattern Mining Approach". In: *8th ACM International Conference on Knowledge Discovery and Data Mining*, pp. 315-324, Alberta, Canada, July.

MONROE, R., KOMPANEK, A., MELTON, R*., et al.*, 1997, "Architectural Styles, Design Patterns and Objects", *IEEE Software*, v. 14, n. 1 (January/February), pp. 43-52.

MONTES DE OCA, C.M., CARVER, D.L., 1998, "A Visual Representation Model for Software Subsystems Decomposition". In: *Working Conference on Reverse Engineering*, pp. 231-240, Honolulu, Hawaii, USA, October.

MORISIO, M., TRAVASSOS, G.H., STARK, M.E., 2000, "Extending UML to Support Domain Analysis". In: *Proceedings of the The Fifteenth IEEE International Conference on Automated Software Engineering (ASE'00)*, pp. 321-324, Grenoble, France, September.

MURTA, L.G.P., 1999, *FRAMEDOC: Um Framework para a Documentação de Componentes Reutilizáveis*, Projeto Final de Graduação, IM, UFRJ, Rio de Janeiro, RJ, Brasil.

MURTA, L.G.P., 2002, *Charon: Uma Máquina de Processos Extensível Baseada em Agentes Inteligentes*, Dissertação de M.Sc., COPPE, UFRJ, Rio de Janeiro, RJ, Brasil.

MURTA, L.G.P., 2006, *Gerência de Configuração no Desenvolvimento Baseado em Componentes*, Tese de D.Sc., COPPE, UFRJ, Rio de Janeiro, RJ, Brasil.

MURTA, L.G.P., BARROS, M.O., WERNER, C.M.L., 2002, "Charon: Uma Ferramenta para a Modelagem, Simulação, Execução e Acompanhamento de Processos de Software". In: *XVI Simpósio Brasileiro de Engenharia de Software*, pp. 366-371, Gramado, RS, Outubro.

MURTA, L.G.P., VAN DER HOEK, A., WERNER, C.M.L., 2006, "ArchTrace: Policy-Based Support for Managing Evolving Architecture-to-Implementation Traceability Links". In: *International Conference on Automated Software Engineering (ASE)*, pp. 135-144, Tokyo, Japan, September.

MURTA, L.G.P., VASCONCELOS, A.P.V., BLOIS, A.P.B*., et al.*, 2004, "Run-Time Variability through Component Dynamic Loading". In: *Sessão de Ferramentas, XVIII Simpósio Brasileiro de Engenharia de Software*, pp. 67-72, Brasília, Brasil, Outubro.

NETO, A.C.D., BARCELOS, R.F., CHAPETTA, W.A*., et al.*, 2004, "Infrastructure for Software Engineering Experiments Definition and Planning". In: *Experimental Software Engineering Latin American Workshop*, Brasília, Brasil, October.

NIERE, J., 2004, "Visualizing Differences of UML Diagrams with Fujaba". In: *2nd Fujaba Days*, Paderborn, Germany September.

NORTHROP, L., 2002, "SEI's Software Product Line Tenets", *IEEE Software*, v. 19, n. 4 (July/August, 2002), pp. 32-40.

O'BRIEN, L., STOERMER, C., 2003, *Architecture Reconstruction Case Study*, Software Engineering Institute, Technical Note CMU/SEI-2003-TN-008.

ODYSSEY, 2007, "Odyssey: Infra-Estrutura de Reutilização baseada em Modelos de Domínio". In: http://reuse.cos.ufrj.br/odyssey, accessed in 03/05/2007.

OHST, D., WELLE, M., KELTER, U., 2003, "Differences Between Versions of UML Diagrams". In: *9th European Software Engineering Conference in conjunction with 11th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, pp. 227-236, Helsinki, Finland, September.

OLIVEIRA, H.L.R., 2006a, *Odyssey-VCS: Uma Abordagem de Controle de Versões para Elementos da UML*, Dissertação de M.Sc., COPPE, UFRJ, Rio de Janeiro, RJ, Brasil.

OLIVEIRA, H.L.R., MURTA, L.G.P., WERNER, C.M.L., 2005a, "Odyssey-VCS: a Flexible Version Control System for UML Model Elements". In: *International Workshop on Software Configuration Management (SCM)*, pp. 1-16, Lisbon, Portugal, September.

OLIVEIRA, R., 2006b, *Formalização e Verificação de Consistência na Representação de Variabilidades*, Dissertação de M.Sc., COPPE, UFRJ, Rio de Janeiro, Brasil.

OLIVEIRA, R., BLOIS, A.P., VASCONCELOS, A*., et al.*, 2005b, "Representação de Variabilidades em Componentes de Negócio no Contexto da Engenharia de Domínio". In: *V Workshop de Desenvolvimento Baseado em Componentes (WDBC)*, pp. 73-80, Juiz de Fora, MG, Brasil, Novembro, 2005.

OLIVEIRA, R., BLOIS, A.P.B., VASCONCELOS, A.P.V*., et al.*, 2005c, *Metamodelo de Características da Notação Odyssey-FEX: Descrição de Classes*, Publicações Técnicas 2/2005, COPPE/UFRJ.

OMG, 2001, *Unified Modeling Language (UML) Specification, version 1.4*, formal/01-09-67, Object Management Group.

OMG, 2002a, *Meta Object Facility (MOF) Specification, version 1.4*, formal/02-04-03, Object Management Group.

OMG, 2002b, *XML Metadata Interchange (XMI) Specification, version 1.2*, formal/02-01-01, Object Management Group.

OMG, 2007a, "Object Management Group". In: http://www.omg.org, accessed in March 16.

OMG, 2007b, "Unified Modeling Language". In: http://www.uml.org/, accessed in January 27.

PACIONE, M.J., ROPER, M., WOOD, M., 2003, "A Comparative Evaluation of Dynamic Visualization Tools". In: *10th Working Conference on Reverse Engineering (WCRE'03)*, pp. 187-196, British Columbia, Canada, November.

PASHOV, I., RIEBISCH, M., 2003, "Using Feature Modeling for Program Comprehension and Software Architecture Recovery". In: *10th IEEE Symposium and Workshops on Engineering of Computer-Based Systems*, pp. 297-304, Huntsville, Alabama, USA, April.

PENEDO, M., RIDDLE, W., 1993, "Process-Sensitive SEE Architecture (PSEEA)", *Software Engineering Notes, ACM SIGSOFT*, v. 18, n. 3 (July), pp. A78-A94.

PETROBRAS, 2007, "Petróleo Brasileiro S/A". In: http://www2.petrobras.com.br/, accessed in 21/07/2007.

PRESSMAN, R.S., 2001, *Software Engineering, a Practitioner's Approach*, 5th ed., McGraw-Hill.

PRIETO-DIAZ, R., ARANGO, G., 1991, "Domain Analysis Concepts and Research Directions", *PRIETO-DIAZ, R., ARANGO, G. (eds), Domain Analysis and Software Systems Modeling, IEEE Computer Society Press*, pp. 9-33.

QOSA, 2007, "International Conference on Quality of Software-Architectures". In: http://qosa.ipd.uka.de/, accessed in January 25.

REUSE, 2007, "Grupo de Reutilização – COPPE/UFRJ". In: http://reuse.cos.ufrj.br, accessed in 15/01/2007.

RICHNER, T., DUCASSE, S., 1999, "Recovering High-Level Views of Object-Oriented Applications from Static and Dynamic Information". In: *International Conference on Software Maintenance (ICSM'99)*, pp. 13-22, Oxford, UK, September.

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

ROUNTEV, A., VOLGIN, O., REDDOCH, M., 2004, *Control Flow Analysis for Reverse Engineering of Sequence Diagrams*, Department of Computer Science and Engineering, Ohio State University, Technical Report OSU-CISRC-3/04-TR12.

SAMETINGER, J., 1997, *Software Engineering with Reusable Components*, Springer-Verlag New York, Inc.

SARTIPI, K., 2003, *Software Architecture Recovery based on Pattern Matching*, PhD Thesis, School of Computer Science, University of Waterloo.

SARTIPI, K., KONTOGIANNIS, K., 2003, "Software Architecture Recovery based on Pattern Matching". In: *International Conference on Software maintenance*, pp. 293-296, Amsterdam, the Netherlands, September.

SCHMERL, B., ALDRICH, J., GARLAN, D*., et al.*, 2006, "Discovering Architectures from Running Systems", *IEEE Transactions on Software Engineering*, v. 32, n. 7 (July), pp. 454-466.

SEEMANN, J., GUDENBERG, J., 1998, "Pattern-Based Design Recovery of Java Software". In: *6th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, pp. 10-16, Lake Buena Vista, Florida, USA, November.

SEI/CMU, 2005, "A Framework for Software Product Line Practice - Version 4.2". In: http://www.sei.cmu.edu/productlines/framework.html, accessed in April 4.

SHAW, M., 1996, "Some Patterns for Software Architectures", *Pattern Languages of Program Design 2*, Boston, MA, USA, Addison-Wesley Longman Publishing Co., Inc.

SHAW, M., GARLAN, D., 1996, *Software Architecture: Perspectives on an Emerging Discipline*, New Jersey, Prentice-Hall.

SHULL, F., CARVER, J., TRAVASSOS, G.H., 2001, "An Empirical Methodology for Introducing Software Processes". In: *European Software Engineering Conference*, pp. 288-296, Vienna, Austria, September.

SHULL, F., RUS, I., BASILI, V., 2000, "How Perspective-Based Reading can Improve Requirements Inspections", *IEEE Computer*, v. 33, n. 7, pp. 73-79.

SHULL, F.J., 1998, *Developing Techniques for Using Software Documents: a Series of Empirical Studies*, PhD Thesis, University of Mariland.

SILVA, I.A., 2005, *GAW: Um Mecanismo Visual de Percepção de Grupo Aplicado ao Desenvolvimento Distribuído de Software*, Projeto Final de Graduação, IM, UFRJ, Rio de Janeiro, RJ, Brasil.

SIMOS, M., ANTHONY, J., 1998, "Weaving the Model Web: A Multi-Modeling Approach to Concepts and Features in Domain Engineering". In: *5th International Conference of Software Reuse (ICSR-5)*, pp. 94-102, Victoria, British Columbia, Canada, June.

STOERMER, C., O'BRIEN, L., 2001, "MAP: Mining Architectures for Product Line Evaluations". In: *3rd Working IFIP Conference on Software Architecture (WICSA)*, pp. 35-44, Amsterdam, Holland, August.

SUGUMARAN, V., PARK, S., KANG, K.C., 2006, "Software Product Line Engineering", *Communications of the ACM*, v. 49, n. 12.

SUN, 2007a, "Java Technology". In: http://java.sun.com/, accessed in March 17.

SUN, 2007b, "JavaCC: Parser/Scanner Generator for Java". In: https://javacc.dev.java.net/, accessed in March 17.

SYSTÄ, T., 2000, "Understanding the Behaviour of Java Programs". In: *7th Working Conference on Reverse Engineering (WCRE)*, pp. 214-223, Brisbane, QL, Australia, November.

TANIGUCHI, K., ISHIO, T., KAMIYA, T.*, et al.*, 2005, "Extracting Sequence Diagram from Execution Trace of Java Programs". In: *International Workshop on Principles of Software Evolution (IWPSE'05)*, pp. 148-151, Lisboa, Portugal, Setembro.

TECGRAF, 2007, "Grupo de Tecnologia em Computação Gráfica da PUC-Rio". In: http://www.tecgraf.puc-rio.br/, accessed in 21/01/2007.

TEIXEIRA, H.V., 2003, *Geração de Componentes de Negócio a partir de Modelos de Análise*, Dissertação de MSC., COPPE, UFRJ, Rio de Janeiro, Brasil.

TICHY, W., 1985, "RCS: a system for version control", v. 15, n. 7 (July), pp. 637-654.

TJORTJIS, C., SINOS, L., LAYZELL, P., 2003, "Facilitating Program Comprehension by Mining Association Rules from Source Code". In: *11th IEEE International Workshop on Program Comprehension* pp. 125-132, Portland, Oregon, USA, May.

TOGETHER, 2007, "Together 2006 Release 2 for Eclipse". In: http://www.borland.com/downloads/download_together.html, accessed in April 4.

TONELLA, P., POTRICH, A., 2003, "Reverse Engineering of the Interaction Diagrams from C++ Code". In: *International Conference on Software Maintenance*, pp. 159-168, Amsterdam, The Netherlands, September.

TRACZ, W., 1987, "Software Reuse: Motivations and Inhibitors". In: *COMPCON'87*, pp. 358-363, Santa Clara, USA, February.

TRACZ, W., HAYES, R., 1994, *DSSA Tool Requirements for Key Process Functions*, Loral Federal Systems, Owego, Technical Report, ADAGE-IBM-93-13B.

TRAVASSOS, G.H., GUROV, D., AMARAL, E.A.G.G., 2002, *Introdução à Engenharia de Software Experimental*, Programa de Engenharia de Sistemas e Computação, COPPE/UFRJ, Relatório Técnico ES-590/02-Abril.

TVEDT, R.T., COSTA, P., LIMDVALL, M., 2002, "Does the Code Match the Design? A Process for Architecture Evaluation". In: *International Conference on Software Maintenance*, pp. 393-401, Montreal, Canada, October.

TZERPOS, V., 2001, *Comprehension-Driven Software Clustering*, PhD, Department of Computer Science, University of Toronto.

VASCONCELOS, A.P.V., CEPÊDA, R.S.V., WERNER, C.M.L., 2005, "An Approach to Program Comprehension through Reverse Engineering of Complementary Software Views". In: *1st International Workshop on Program Comprehension through Dynamic Analysis*, pp. 58-62, Pittsburgh, PA, USA, November.

VASCONCELOS, A.P.V., WERNER, C.M.L., 2004a, "An Approach to Software Architecture Recovery Aiming at its Reuse in the Context of Domain Engineering". In: *XVIII Simpósio Brasileiro de Engenharia de Software, IX Workshop de Teses e Dissertações em Engenharia de Software*, pp. 09-16, Brasília, Brasil, Outubro.

VASCONCELOS, A.P.V., WERNER, C.M.L., 2004b, "Software Architecture Recovery based on Dynamic Analysis". In: *XVIII Simpósio Brasileiro de Engenharia de Software, Workshop de Manutenção de Software Moderna*, Brasília, Brasil, Outubro.

VASCONCELOS, A.P.V., WERNER, C.M.L., 2005a, "Towards a set of Application Independent Clustering Criteria within an Architecture Recovery Approach". In: *5th IEEE/IFIP Working Conference on Software Architecture - Software Architecture Evaluation and Analysis Working Session*, pp. 235-236, Pittsburgh, PA, USA, November.

VASCONCELOS, A.P.V., WERNER, C.M.L., 2005b, "Um Conjunto de Heurísticas de Agrupamento de Classes para Apoiar a Recuperação da Arquitetura de Software". In: *II Workshop de Manutenção de Software Moderna*, pp. 34-49, Manaus, AM, Brasil, Novembro.

VASCONCELOS, A.P.V., WERNER, C.M.L., 2007a, "Architectural Elements Recovery and Quality Evaluation to Assist in Reference Architectures Specification". In: *19th International Conference on Software Engineering and Knowledge Engineering, To Appear*, Boston, USA, July.

VASCONCELOS, A.P.V., WERNER, C.M.L., 2007b, "Architecture Recovery and Evaluation Aiming at Program Understanding and Reuse". In: *QoSA'07: Third International Conference on the Quality of Software-Architectures, To Appear*, Boston, Massachusetts, USA, July.

VERONESE, G.O., NETTO, F.J., 2001, *ARES: Uma Ferramenta de Auxílio à Recuperação de Modelos UML de Projeto a partir de Código Java*, Projeto Final de Graduação, IM, UFRJ, Rio de Janeiro, Brasil.

VICI, A.D., ARGENTIERI, N., 1998, "FODAcom: An Experience with Domain Analysis in the Italian Telecom Industry". In: *Fifth International Conference on Software Reuse (ICSR5)* pp. 166-175, Victoria, British Columbia, Canada, April.

VILLELA, K., 2004, *Definição e Construção de Ambientes de Software Orientados à Organização*, Tese de D.Sc., Programa de Engenharia de Sistemas e Computação - COPPE, UFRJ, Rio de Janeiro, Brasil.

VITHARANA, P., JAIN, H., ZAHEDI, F.M., 2004, "Strategy-Based Design of Reusable Business Components", *IEEE Transactions on Systems, Man, and Cybernetics – Part C: Applications and Reviews*, v. 34, n. 4, pp. 460-474.

W3C, 2007, "Document Object Model (DOM)". In: http://www.w3.org/DOM/, accessed in March 20.

WALKER, R.J., MURPHY, G.C., FREEMAN-BENSON, B.*, et al.*, 1998, "Visualizing Dynamic Software System Information through High-Level Models". In: *ACM Conference on Object-Oriented Programming, Systems, Languages, and Applications*, pp. 271-283, British Columbia, Canada, October.

WALLNAU, K.C., HISSAM, S.A., SEACORD, R.C., 2002, *Building Systems from Commercial Components*, 1st ed., Boston, USA, Addison-Wesley.

WEN, Z., TZERPOS, V., 2004, "An Effectiveness Measure for Software Clustering Algorithms". In: *12th IEEE International Workshop on Program Comprehension (IWPC'04)*, pp. 194-203, Bari, June.

WENTZEL, K., 1994, "Software Reuse, Facts and Myths". In: *16th Annual International Conference on Software Engineering*, pp. 267-273, Sorrento, Italy, May.

WERNER, C.M.L., 2005, "Reuse: Técnicas e Ferramentas de Apoio à Reutilização de Software", *Projeto Integrado de Pesquisa do CNPQ*.

WERNER, C.M.L., BRAGA, R.M.M., 2005, "A Engenharia de Domínio e o Desenvolvimento Baseado em Componentes". In: GIMENES, I.M.S., HUZITA, E.H.M. (eds), *Desenvolvimento Baseado em Componentes: Conceitos e Técnicas*, Rio de Janeiro, Ciência Moderna.

WERNER, C.M.L., MATTOSO, M., BRAGA, R.*, et al.*, 1999, "Odyssey: Infra-Estrutura de Reutilização Baseada em Modelos de Domínio". In: *Sessão de Ferramentas do XIII Simpósio Brasileiro de Engenharia de Software*, pp. 17-20, Florianópolis, SC, Brasil, Outubro.

WICSA, 2007, "Sixth Working IEEE/IFIP Conference on Software Architecture". In: http://www.gv.psu.edu/WICSA2007/, accessed in January 25.

WIKIPÉDIA, 2007, "Wikipédia: A Enciclopédia Livre". In: http://pt.wikipedia.org/wiki, accessed in 04/04/2007.

WILLIAMS, L.G., SMITH, C.U., 1998, "Performance Evaluation of Software Architectures". In: *1st International Workshop on Software and Performance (WOSP)*, pp. 164-177, Santa Fe, New Mexico, USA, October.

WILLIAMS, T., 1998, "Reusable Components for Evolving Systems". In: *5th International Conference on Software Reuse*, pp. 12-16, Victoria, British Columbia, Canada, June.

WOHLIN, C., RUNESON, P., HÖST, M.*, et al.*, 2000, *Experimentation in Software Engineering: an Introduction*, USA, Kluwer Academic Publishers.

WONG, K., TILLEY, S., MÜLLER, H.*, et al.*, 2004, "Programmable Reverse Engineering", *International Journal of Software Engineering and Knowledge Engineering* v. 4, n. 4 (December), pp. 501-620.

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

ZAIDMAN, A., CALDERS, T., DEMEYER, S.*, et al.*, 2005, "Applying WebMining Techniques to Execution Traces to Support the Program Comprehension Process". In: *European Conference on Software Maintenance and Reengineering*, pp. 134-142, Manchester, UK, March.

Anexo A: Instrumentação do Estudo Experimental de Viabilidade de ArchMine

A.1: Formulário de Consentimento para Apoio no Estudo de Viabilidade da Abordagem ArchMine

Apoio no Estudo de Viabilidade da Abordagem de Recuperação de Arquitetura ArchMine

Eu declaro ter mais de 18 anos de idade e que concordo em participar de um estudo conduzido por Aline Pires Vieira de Vasconcelos, como parte das atividades de doutoramento no Programa de Engenharia de Sistemas e Computação da COPPE/UFRJ. Este estudo visa verificar a viabilidade em se utilizar a abordagem de recuperação de arquitetura baseada na mineração de rastros de execução, ArchMine, para apoiar a recuperação da arquitetura de software.

PROCEDIMENTO

Este estudo está dividido em três etapas: primeiramente, o especialista, participante do estudo, apóia o pesquisador na definição dos cenários de caso de uso da aplicação; em seguida, o pesquisador aplica a abordagem ArchMine para recuperar a arquitetura da aplicação; por fim, o especialista avalia a arquitetura recuperada, preenchendo os formulários de avaliação. Com base na análise dos formulários de avaliação, o pesquisador será capaz de verificar se a abordagem é viável e, em caso positivo, verificar pontos da abordagem que necessitam de melhoria para refiná-la.

BENEFÍCIOS, LIBERDADE DE DESISTÊNCIA

Eu entendo que sou livre para realizar perguntas a qualquer momento ou solicitar que qualquer informação relacionada a minha pessoa não seja incluída no estudo. Eu entendo que participo de livre e espontânea vontade com o único intuito de contribuir para o avanço e desenvolvimento de técnicas e processos para a Engenharia de Software.

EQUIPE: **PESQUISADOR RESPONSÁVEL**
MSC. Aline Pires Vieira de Vasconcelos
Programa de Engenharia de Sistemas e Computação - COPPE/UFRJ

PROFESSOR RESPONSÁVEL
Profa. Cláudia M.L. Werner
Programa de Engenharia de Sistemas e Computação - COPPE/UFRJ

Nome (em letra de forma):_____

Assinatura: _____ Data:_____

A.2: Formulários de Avaliação da Arquitetura Recuperada

Formulário1 ***Avaliação da Arquitetura Recuperada*** **Identificação:**
Aplicação: _____ **Especialista: _____**
***Questões*:**
Q1. Quantos elementos arquiteturais foram recuperados? _____
Q2. Dentre os elementos arquiteturais recuperados, quantos elementos você julga que correspondem a elementos arquiteturais da aplicação? _____
Q3. Indique, nas linhas abaixo, os elementos arquiteturais recuperados que você julga que correspondem a elementos arquiteturais da aplicação. _____ _____ _____ _____ _____ _____
Q4. Quantos elementos arquiteturais estão faltando? _____
Q5. Indique, nas linhas abaixo, os elementos arquiteturais da aplicação que estão faltando. _____ _____ _____ _____

Q6. Preencha o quadro a seguir levando em conta a estrutura interna dos elementos arquiteturais que correspondem a elementos da aplicação.

Elemento Arquitetural	Classes Alocadas Incorretamente	Classes Faltantes

Q7. Qual o percentual de classes coberto pela arquitetura recuperada?

_____

Q8. Qual o Estilo Arquitetural da aplicação?

_____

Formulário 2

Avaliação da Arquitetura Recuperada

Este é um formulário de perguntas subjetivas, as quais devem ser respondidas com um dos valores do conjunto {"Sim", "Não","Parcialmente"}, seguido de uma justificativa.

Q9. Os elementos arquiteturais recuperados representam uma possível representação da arquitetura da aplicação?

_____

_____

_____

_____

_____

Q10. Os elementos arquiteturais recuperados apóiam a compreensão da aplicação para fins de manutenção?

_____

_____

_____

_____

_____

Q11. Os elementos arquiteturais recuperados poderiam ser utilizados como base para uma Reengenharia da aplicação?

_____

_____

_____

_____

_____

Q12. Os elementos arquiteturais recuperados poderiam ser utilizados para uma Reengenharia da aplicação do paradigma de Orientação a Objetos para Componentes que implementam funcionalidades/serviços bem definidos?

_____

_____

_____

_____

_____

Q13. Os elementos arquiteturais recuperados podem ser reutilizados no projeto de novas aplicações no mesmo domínio?

_____

_____

_____

_____

_____

_____

Q14. A arquitetura da aplicação corresponde à organização dos pacotes do código fonte?

_____

_____

_____

_____

_____

Se a resposta à pergunta anterior foi ”Parcialmente”, a seguinte pergunta deve ser respondida.

Q15. A arquitetura da aplicação está mais próxima da organização de pacotes do código fonte ou dos elementos arquiteturais recuperados? (esta pergunta deve ser respondida com uma das duas opções: “código fonte” ou “elementos arquiteturais recuperados” e uma justificativa)

_____

_____

_____

_____

A.3: Formulário de Avaliação dos Formulários de Avaliação de Arquitetura

Avaliação dos Formulários

O objetivo deste formulário é avaliar a clareza e grau de dificuldade nas respostas em relação aos dois formulários anteriores.

Q1. Os formulários de avaliação da arquitetura são claros? (Responda com "Sim", "Não" ou "Parcialmente", seguido de uma justificativa).

_____
_____
_____
_____
_____
_____
_____

Q2. Se a resposta à pergunta anterior foi "Não" ou "Parcialmente", cite e justifique as questões dos formulários que geram dúvida em sua interpretação.

_____
_____
_____
_____
_____

Q3. Você indicaria alguma questão a ser incluída ou retirada dos formulários de avaliação da arquitetura? Por quê?

_____
_____
_____
_____
_____
_____
_____

A.4: Instruções para o preenchimento dos Formulários de Avaliação da Arquitetura Recuperada

I. A seção de "Identificação" deve ser preenchida com as seguintes informações:

1. No campo "Aplicação" deve ser preenchido o nome da aplicação para a qual a arquitetura foi recuperada.
2. No campo "Especialista" deve ser preenchido o nome completo do especialista da aplicação.

II. A seção de "Questões" deve ser preenchida da seguinte forma:

1. **Q1**: a arquitetura da aplicação é representada através de um diagrama de pacotes da UML com suas dependências. Assim, a quantidade de elementos arquiteturais recuperados é obtida contabilizando-se os pacotes de mais alto nível da arquitetura.
2. **Q2**: dentre os pacotes recuperados, deve-se julgar quais que correspondem de fato a possíveis elementos arquiteturais da aplicação. Estes pacotes devem ser contabilizados.
3. **Q3:** esta pergunta representa um detalhamento da pergunta Q2. Nesta resposta, os nomes dos elementos contabilizados na pergunta Q2 como corretos devem ser informados.
4. **Q4**: nesta questão devem ser informados os pacotes que deveriam estar presentes no modelo arquitetural e que estão faltando.
5. **Q5**: esta pergunta representa um detalhamento da pergunta Q4. Nesta resposta, os nomes dos elementos contabilizados na pergunta Q4 devem ser informados.
6. **Q6**: a fim de preencher o quadro da questão Q6, a estrutura interna dos pacotes indicados na questão Q3 deve ser verificada. Classes presentes em um pacote (i.e. elemento arquitetural) e que deveriam estar em outro, devem ser indicadas como "Classes Alocadas Incorretamente". Por outro lado, classes que deveriam estar presentes no elemento arquitetural que o pacote representa e que estão faltando devem ser informadas na coluna "Classes Faltantes".

7. **Q7**: do número total de classes da aplicação, quantas foram alocadas na arquitetura recuperada? Este número deve ser convertido para um valor percentual.
8. **Q8**: deve ser informado o estilo arquitetural ou estilos arquiteturais adotados pela aplicação, como MVC, Tubos e Filtros, Camadas etc.

As questões **Q9** a **Q15** têm um caráter mais subjetivo

9. **Q9**: O especialista deve julgar se a arquitetura recuperada corresponde de fato a uma possível representação da arquitetura da aplicação (i.e. a um possível modelo da arquitetura da aplicação) que poderia ser utilizada para apoiar tarefas de manutenção.
10. **Q10**: a mesma lógica anterior vale para esta questão.
11. A questão **Q11** somente deve ser levada em conta caso o especialista tenha identificado que a arquitetura recuperada corrige problemas na arquitetura original da aplicação.
12. A questão **Q12** visa identificar se o agrupamento de classes em elementos arquiteturais conseguiu alocar num mesmo elemento classes coesas, i.e. que implementam funcionalidades relacionadas.
13. A questão **Q13** está relacionada com a **Q12.** Ou seja, para os elementos arquiteturais que implementam funcionalidades relacionadas, i.e. que apresentam coesão, o especialista deve avaliar se no seu ponto de vista seria possível reutilizá-los em novas aplicações do mesmo domínio.
14. As questões **Q14** e **Q15** visam identificar se a arquitetura original da aplicação se aproxima mais da organização de pacotes do código fonte ou da arquitetura recuperada pela abordagem ArchMine.

Anexo B: Instrumentação do Estudo Experimental Comparativo de ArchMine

B.1: Formulário de Consentimento para Apoio no Estudo Comparativo da Abordagem ArchMine

Apoio na Avaliação da Arquitetura de Software Recuperada

Eu declaro ter mais de 18 anos de idade e que concordo em participar de um estudo conduzido por Aline Pires Vieira de Vasconcelos, como parte das atividades de doutoramento no Programa de Engenharia de Sistemas e Computação da COPPE/UFRJ. Este estudo visa verificar a viabilidade em se utilizar a abordagem de recuperação de arquitetura baseada na mineração de rastros de execução, ArchMine, para apoiar a recuperação da arquitetura de software.

PROCEDIMENTO

Este estudo está dividido em duas etapas: primeiramente, o pesquisador aplica as abordagens ArchMine e Bunch para recuperar dois modelos da arquitetura da aplicação. Em seguida, o especialista, participante do estudo, avalia as arquiteturas recuperadas, preenchendo os formulários de avaliação. Com base na análise dos formulários de avaliação, o pesquisador será capaz de verificar qual das abordagens apresenta melhor eficácia na recuperação de arquitetura, além de identificar pontos de melhoria em ArchMine.

BENEFÍCIOS, LIBERDADE DE DESISTÊNCIA

Eu entendo que sou livre para realizar perguntas a qualquer momento ou solicitar que qualquer informação relacionada a minha pessoa não seja incluída no estudo. Eu entendo que participo de livre e espontânea vontade com o único intuito de contribuir para o avanço e desenvolvimento de técnicas e processos para a Engenharia de Software.

EQUIPE: **PESQUISADOR RESPONSÁVEL**
MSC. Aline Pires Vieira de Vasconcelos
Programa de Engenharia de Sistemas e Computação - COPPE/UFRJ

PROFESSOR RESPONSÁVEL
Profa. Cláudia M.L. Werner
Programa de Engenharia de Sistemas e Computação - COPPE/UFRJ

Nome (em letra de forma):_____

Assinatura: _____ Data:_____

B.2: Formulários de Avaliação da Arquitetura Recuperada

Formulário 1: *Avaliação da Arquitetura Recuperada* **Identificação:**
Aplicação: **Especialista:**

***Questões*:**

Q1. Quantos elementos arquiteturais foram recuperados?

_____

Q2. Para cada elemento arquitetural recuperado (i.e. pacote na Structural View subordinado ao pacote "arquiteturax"), preencha no quadro abaixo o Conceito ou Funcionalidade de domínio que você julga que ele represente. Em função desta resposta, indique as classes que você julga que estariam alocadas incorretamente.

Obs.: cada Elemento Arquitetural deve ter correspondência para apenas um conceito ou funcionalidade do domínio. Se você encontrar mais de uma correspondência, indique o conceito ou funcionalidade para o qual a maior parte das classes do elemento arquitetural tem correspondência. Se você não conseguir encontrar correspondência para o elemento arquitetural no Domínio, indique que não existe esta correspondência.

Elemento Arquitetural	Correspondência no Domínio	Classes Alocadas Incorretamente

Q3. Para cada elemento arquitetural recuperado (i.e. pacote na Structural View subordinado ao pacote "arquiteturax"), levando em conta o conceito ou funcionalidade do domínio que ele representa, preencha no quadro abaixo as classes que estariam faltando. Indique casos onde você julga que o elemento indicado na primeira coluna poderia ser combinado com um ou mais elementos, indicando seus nomes na segunda coluna.

Elemento Arquitetural	Classes ou Elementos a serem Incluídos

Q4. Qual(is) o(s) Estilo(s) Arquitetural(is) da aplicação?

_____

Formulário 2:

Avaliação da Arquitetura Recuperada

Este é um formulário de perguntas subjetivas, as quais devem ser respondidas com um dos valores do conjunto {"Sim", "Não", "Parcialmente"}, seguido de uma justificativa.

Q5. O modelo arquitetural recuperado representa um possível modelo da arquitetura da aplicação, que apóie a sua manutenção?

_____

_____

_____

_____

_____

Q6. Os elementos arquiteturais recuperados poderiam ser utilizados para uma Reengenharia da aplicação do paradigma de Orientação a Objetos para Componentes que implementam funcionalidades/serviços bem definidos? Esta questão deve ser respondida olhando o conteúdo dos elementos individualmente.

_____

_____

_____

_____

Q7. A arquitetura recuperada poderia ser reutilizada no projeto de novas aplicações no mesmo domínio? Esta questão deve ser respondida olhando a estrutura de pacotes recuperada.

_____

_____

_____

_____

_____

_____

B.3: Formulário de Avaliação dos Formulários de Avaliação de Arquitetura

Avaliação dos Formulários

O objetivo deste formulário é avaliar a clareza e grau de dificuldade nas respostas em relação aos dois formulários anteriores.

Q1. Os formulários de avaliação da arquitetura são claros? (Responda com "Sim", "Não" ou "Parcialmente", indicando e justificando questões a serem incluídas ou retiradas).

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

Anexo C: Instrumentação do Estudo Experimental de Viabilidade das Extensões realizadas na Abordagem de Avaliação de Arquitetura

C.1: Formulário de Consentimento para Apoio no Estudo de Viabilidade das Extensões no *Checklist* de ArqCheck

Apoio na Avaliação de Arquiteturas através da Abordagem ArqCheck Estendida

Eu declaro ter mais de 18 anos de idade e que concordo em participar de um estudo conduzido por Aline Pires Vieira de Vasconcelos, como parte das atividades de doutoramento no Programa de Engenharia de Sistemas e Computação da COPPE/UFRJ. Este estudo visa avaliar se as extensões realizadas no *checklist* da abordagem ArqCheck para avaliação arquitetural de fato apóiam a detecção de defeitos em relação ao atributo de qualidade Reusabilidade.

PROCEDIMENTO

Os participantes deverão inspecionar o documento arquitetural, através de uma abordagem de inspeção remota e assíncrona, onde informações não devem ser trocadas entre si durante a realização da inspeção. A inspeção deve ser conduzida em apenas uma sessão e o tempo de inspeção deve ser registrado pelos participantes.

BENEFÍCIOS, LIBERDADE DE DESISTÊNCIA

Eu entendo que sou livre para realizar perguntas a qualquer momento ou solicitar que qualquer informação relacionada a minha pessoa não seja incluída no estudo. Eu entendo que participo de livre e espontânea vontade com o único intuito de contribuir para o avanço e desenvolvimento de técnicas e processos para a Engenharia de Software.

EQUIPE: PESQUISADOR RESPONSÁVEL

MSC. Aline Pires Vieira de Vasconcelos
Programa de Engenharia de Sistemas e Computação - COPPE/UFRJ

PROFESSOR RESPONSÁVEL
Profa. Cláudia M.L. Werner
Programa de Engenharia de Sistemas e Computação - COPPE/UFRJ

Nome (em letra de forma):_____

Assinatura: _____ **Data:**_____

C.2: Questionário de Caracterização dos Participantes – Adaptado de BARCELOS (2006)

Caracterização do Participante – Estudo sobre as extensões realizadas no *checklist* da abordagem de avaliação de arquitetura ArqCheck para a detecção de defeitos de Reusabilidade

Nome _____

Nível (Ms.c/D.Sc.): ____

Formação Geral

Qual é sua experiência anterior com desenvolvimento de software na prática ? (marque aqueles itens que melhor se aplicam)

__ nunca desenvolvi software.

__ tenho desenvolvido software para uso próprio.

__ tenho desenvolvido software como parte de uma equipe, relacionado a um curso.

__ tenho desenvolvido software como parte de uma equipe, na indústria.

Por favor, explique sua resposta. Inclua o número de semestres ou número de anos de experiência relevante em desenvolvimento (E.g. "Eu trabalhei por 10 anos como programador na indústria"):

_____

_____

Experiência em Desenvolvimento de Software

Por favor, indique o grau de sua experiência nesta seção seguindo a escala de 5 pontos abaixo:

1 = nenhum

2 = estudei em aula ou em livro

3 = pratiquei em 1 projeto em sala de aula

4 = usei em 1 projeto na indústria

5 = usei em vários projetos na indústria

Experiência com Requisitos

- Experiência escrevendo requisitos 1 2 3 4 5
- Experiência escrevendo casos de uso 1 2 3 4 5

- Experiência revisando requisitos 1 2 3 4 5
- Experiência revisando casos de uso 1 2 3 4 5
- Experiência modificando requisitos para manutenção 1 2 3 4 5

Experiência em Arquitetura de Software

- Experiência em projeto de arquitetura de software 1 2 3 4 5
- Experiência em leitura de documento arquitetural 1 2 3 4 5

Experiência em Projeto

- Experiência em projeto de sistemas 1 2 3 4 5
- Experiência em desenvolver projetos a partir de requisitos e casos de uso 1 2 3 4 5
- Experiência criando projetos Orientado a Objetos 1 2 3 4 5
- Experiência lendo projetos Orientado a Objetos 1 2 3 4 5
- Experiência com Unified Modeling Language (UML) 1 2 3 4 5
- Experiência alterando projeto para manutenção 1 2 3 4 5

Experiência em Reutilização de Software

- Experiência em desenvolver modelos para reutilização, como *frameworks* 1 2 3 4 5
- Experiência em reutilizar *frameworks* 1 2 3 4 5
- Experiência em desenvolver componentes 1 2 3 4 5
- Experiência em reutilizar componentes 1 2 3 4 5

Outras Experiências

- Experiência com inspeções de software? 1 2 3 4 5
- Experiência na utilização da abordagem ArqCheck? 1 2 3 4 5

Para a questão a seguir, considere a seguinte escala de experiência:

1 = nenhuma

2 = já li a respeito

3 = já estudei a respeito

4 = já utilizei aplicações

5 = já desenvolvi aplicações

- Experiência no domínio para o qual a arquitetura está sendo avaliada? 1 2 3 4 5

C.3: Extensões realizadas no *Checklist* da abordagem ArqCheck para a Avaliação do Atributo de Qualidade Reusabilidade

CheckList para Inspeção de Documento Arquitetural

=> **Instruções**:

- Avalie a documentação arquitetural através dos itens de avaliação abaixo.
- A opção NA (Não Aplicável) deve ser marcada se for considerado que não é possível aplicar o item para avaliar a documentação arquitetural em questão.
- A documentação arquitetural deve ser avaliada em relação aos requisitos de qualidade identificados nos Cenários e aos elementos identificados na Guia de Identificação de Contexto. Caso os requisitos não tenham sido identificados, o inspetor deve avaliar somente se o item é aplicável ou não.

Equipe/Revisor: _____

Itens de avaliação de atendimento aos requisitos de qualidade				
Nº	**Itens relacionados a Reusabilidade**	Sim	Não	NA
Requisito de Reusabilidade: a fim de realizar as principais seqüências de execução da infra-estrutura eSEE (i.e. fluxo de configuração, de mapeamento, de instanciação e de execução), três elementos arquiteturais principais foram definidos, a saber: o Meta-Configurador, o Ambiente para Instanciação e o Ambiente para Execução. Estes elementos, bem como seus sub-elementos arquiteturais, foram definidos com o menor acoplamento quanto possível, a fim de permitir a sua reutilização em novas aplicações do mesmo domínio. Embora Reusabilidade não seja um requisito não-funcional previsto na documentação de requisitos do eSEE, este é também um requisito não-funcional desejável.				

Cenário de Reusabilidade	
Fonte do estímulo	Desenvolvedor
Estímulo	Reutilização dos elementos arquiteturais que representam as principais funcionalidades da aplicação em outras aplicações do mesmo domínio.
Contexto do Sistema	Desenvolvimento de novas aplicações no mesmo domínio
Artefato	Meta-Configurador, Ambiente para Instanciação, Ambiente para Execução, DadosConfigurador, DadosExecutor e seus sub-elementos arquiteturais.
Resposta	Elementos arquiteturais extraídos da arquitetura original e reutilizados em outras aplicações.
Medida de Resposta	-------

Guia de identificação de contexto: de acordo com os requisitos de reusabilidade que se deseja avaliar, identificar os elementos arquiteturais que visam ser reutilizados em novas aplicações (*Elementos Reutilizáveis*), seus módulos internos, responsabilidades e relacionamentos com os demais elementos arquiteturais (i.e. elementos para os quais ele possua relacionamento direto, independente deste relacionamento ser para um *Elemento Reutilizável*). A identificação dos *Elementos Reutilizáveis* pode ser feita principalmente a partir da análise do estímulo e artefato descritos nos cenários de reusabilidade.

		Sim	Não	NA
1.	As responsabilidades dos módulos internos de um *Elemento Reutilizável* pertencem a um mesmo contexto, ou seja, visam atingir um mesmo propósito, manipulam um mesmo tipo de dado ou são utilizadas nas mesmas seqüências de execução?			
2.	Do ponto de vista dos serviços ou conceito que o *Elemento Reutilizável* representa, existem módulos que deveriam estar alocados nele e que estão alocados num outro elemento arquitetural?			

3.	É possível identificar grupos de *Elementos Reutilizáveis* com funcionalidades ou responsabilidades relacionadas que, para compor um componente de software, poderiam ser agrupados em um único elemento?			
4.	Acoplamento forte entre elementos arquiteturais reduz a sua reusabilidade por causa da necessidade de se utilizar estes elementos arquiteturais em conjunto. Nesse contexto, existem *Elementos Reutilizáveis* com alto acoplamento, para os quais alguns de seus relacionamentos com os demais elementos arquiteturais poderiam ser eliminados?			
5.	Ainda considerando o acoplamento, existem relacionamentos entre o *Elemento Reutilizável* e demais elementos arquiteturais que justificaria o seu agrupamento em um único *Elemento Reutilizável*?			
6.	Considerando que um grande componente tende a prover funcionalidades além daquelas necessárias para satisfazer um requisito em particular, o tamanho (i.e. granularidade) dos *Elementos Reutilizáveis* é adequado para permitir a sua reutilização?			

Glossário dos termos utilizados no *checklist*:

1 – **Elemento Arquitetural**: elemento da arquitetura utilizado para agrupar sub-elementos arquiteturais ou módulos que teoricamente pertencem a um mesmo contexto, ou seja, possuem responsabilidades similares. Implementa um conjunto de funcionalidades (ou serviços) relacionadas ou conceito da aplicação. A representação da sua estrutura interna é essencial para o entendimento da solução. Em UML, elementos arquiteturais podem ser representados através de pacotes, cujos elementos internos podem ser representados através de subpacotes ou classes.

2 – **Sub-Elemento Arquitetural**: elemento arquitetural que compõe um elemento arquitetural de mais alto nível de abstração.

3 – **Módulo (ou módulo interno)**: elemento pertencente ao mais baixo nível de abstração, o qual compõe os elementos arquiteturais. Este elemento realiza diversas responsabilidades e se relaciona com outros elementos visando atender a um conjunto de requisitos. Em UML, módulos podem ser representados através de classes.

4 - **Relacionamento entre elementos arquiteturais**: qualquer ligação (ex: dependência) entre dois elementos arquiteturais.

5 - **Relacionamento entre módulos internos**: relacionamento entre módulos de dois elementos arquiteturais.

6 – **Componentes de software**: componentes de software são artefatos autocontidos, claramente identificáveis, que descrevem ou realizam uma função específica e têm interfaces claras, documentação apropriada e um grau de reutilização definido (SAMETINGER, 1997).

7 – **Seqüência de execução**: conjunto de fluxos de mensagens entre os elementos arquiteturais ou entre módulos de elementos arquiteturais que determina o comportamento esperado do sistema quando se deseja realizar uma determinada tarefa. Em UML, esse comportamento pode ser representado através de diagramas de seqüência associados a casos de uso.

Raciocínio para as questões do *checklist* referentes à avaliação do atributo de qualidade Reusabilidade:

Convém ressaltar que, embora a idéia por trás do *checklist* é a de que ele possa ser usado para avaliar a reusabilidade de elementos arquiteturais genericamente, independente do paradigma de desenvolvimento adotado, as razões (*rationale*) para as questões do *checklist* foram retiradas da literatura de DBC (Desenvolvimento Baseado em Componentes). Isso se deve ao fato de DBC representar um paradigma que solidifica princípios e métodos para a estruturação de artefatos reutilizáveis bem definidos. Dessa forma, o conceito de componente nos itens a seguir pode se aplicar a componentes de software propriamente ou a elementos arquiteturais que pretendem ser reutilizados em outras aplicações. Segue então os itens que dão embasamento às questões apresentadas no *checklist.*

Acoplamento: acoplamento forte entre componentes reduz a sua reusabilidade por causa da necessidade de se utilizar estes componentes em conjunto (VITHARANA *et al.*, 2004). Acoplamento entre componentes ou entre elementos arquiteturais pode ser definido como: a extensão em que classes no componente se relacionam a outras classes que não estão neste componente (VITHARANA *et al.*, 2004). Expressando a importância do acoplamento, HOPKINS (2000) afirma que "comunicação inter-componentes é bastante cara em termos de tempo e recursos de plataforma".

Coesão: refere-se à força de associação dos elementos em um sistema (WALLNAU *et al.*, 2002). Coesão pode ser medida pela extensão em que as classes de um componente ou elemento arquitetural estão relacionadas (VITHARANA *et al.*, 2004). Segundo VITHARANA *et al.* (2004), coesão representa um importante critério de projeto na construção de componentes reutilizáveis. Além desse aspecto, VITHARANA *et al.* (2004) afirmam ainda que uma vez que os requisitos para uma aplicação particular tendem a estar relacionados, um componente ou elemento arquitetural mais coeso vai satisfazer a requisitos relacionados mais freqüentemente, aumentando, portanto, a sua reusabilidade.

Granularidade: o número de classes em um componente determina o seu tamanho (VITHARANA *et al.*, 2004). WILLIAMS (1998) argumenta que restringindo a funcionalidade de um componente pode aumentar a sua reusabilidade. Um grande componente ou elemento arquitetural tende a prover funcionalidades além daquelas necessárias para satisfazer um requisito em particular, o que reduz a sua reusabilidade.

Acoplamentos entre Componentes: em (BLOIS, 2006), é argumentado que quando se tem um grande número de componentes na arquitetura, estes componentes podem trocar um grande volume de mensagens e, muitas vezes, esta interação ocorre entre um grupo específico de componentes. Dessa forma, BLOIS (2006) sugere que componentes que trocam muitas mensagens entre si sejam agrupados em um único elemento arquitetural. Os critérios oferecidos em (BLOIS, 2006) para apoiar estes agrupamentos se baseiam, dentre outros, em: interfaces providas e requeridas; e componentes requeridos e requerendo outros. Este último critério avalia quantos e quais componentes são requeridos por um outro componente e pode ser aplicado a qualquer tipo de elemento arquitetural. Se um componente requer dois ou mais (i.e. um pequeno número) de componentes, é sugerido o agrupamento dos mesmos. Pode-se formar grupos, neste caso, onde 2 ou 3 componentes só possuam dependências entre si e um dos componentes do grupo apenas possui dependências com outros componentes.

C.4:

Relatório de Discrepâncias

Equipe/Revisor #: _____

Tempo de inspeção (em minutos): _____

- ⇨ **Instruções**:
  - Os elementos arquiteturais e/ou módulos onde se localizam os defeitos devem ser identificados pelo seu nome.
  - Os diagramas onde os defeitos foram localizados devem ser identificados pelo nome da visão e o seu número (Visão::Diagrama).
  - Os tipos de defeitos podem ser: omissão, ambigüidade, inconsistência, informação estranha ou fato incorreto.

Tipos de Defeitos	**Descrição**
Omissão	1. Quando um elemento arquitetural necessário para o atendimento a um requisito não foi definido.
Ambigüidade	2. Quando a forma como os elementos arquiteturais ou suas responsabilidades foram definidos dificulta ou impossibilita o atendimento a um requisito de qualidade. 3. Quando elementos descritos em visões distintas possuem o mesmo nome, mas responsabilidades diferentes (Homônimos).
Inconsistência	4. Quando elementos descritos em visões distintas possuem mesmas responsabilidades, mas nomes distintos (Sinônimos). 5. Quando um elemento arquitetural presente em diagramas das demais visões não foi definido no diagrama avaliado. 6. Quando a representação não condiz com a semântica estabelecida pela abordagem de documentação. 7. Quando um elemento arquitetural é definido com responsabilidades distintas em duas ou mais visões. 8. Quando um elemento é representado de maneiras diferentes em duas visões.
Fato Incorreto	9. Quando um elemento não foi descrito ou representado de forma correta. 10. Quando não é possível mapear um elemento arquitetural para algum elemento descrito em outra visão.
Informação Estranha	11. Quando não é possível determinar o papel de um elemento arquitetural ou de uma de suas responsabilidades no atendimento aos requisitos especificados.

Nº	Item do *checklist*	Elementos Arquiteturais Envolvidos	Diagramas	Descrição da Discrepância	Tipo do Defeito

C.5: Questionário de Avaliação Pós-Experimento (BARCELOS, 2006)

Questionário de Avaliação Pós-Experimento

Equipe/Revisor #: _____

Por favor, responda o questionário abaixo de forma a nos permitir o registro de sua opinião sobre a aplicação da abordagem para inspeção de documentos arquiteturais baseada em checklist (ArqCheck), além dos procedimentos envolvidos na condução do exercício. A sua opinião é muito importante para nós.

Checklist de avaliação arquitetural

1. **Como você classificou o grau de dificuldade da aplicação do checklist?**

___Muito Fácil ___Fácil ___Mediano ___Difícil ___Muito Difícil

2. **Na sua opinião, quais aspectos da técnica tornam sua aplicação fácil/difícil de usar?**

3. **Você identificou os defeitos utilizando somente os itens de avaliação presentes no checklist?**

___ Sim. Utilizei somente o conhecimento presente nos itens para avaliar o documento arquitetural fornecido.
___ Não. Eu utilizei conhecimento próprio

4. **Caso você tenha identificado defeitos com o auxílio de outro tipo de conhecimento, além do presente no checklist, por exemplo, conhecimento do domínio, identifique para que defeitos isso ocorreu e que tipo de conhecimento que você utilizou.**

5. **Como o checklist o auxiliou a identificar defeitos nos requisitos?**

___ Negativamente. O checklist atuou como um obstáculo. O meu desempenho teria sido melhor se eu não o tivesse utilizado.
___ Neutro. Acho que encontraria os mesmos defeitos caso não tivesse utilizado o checklist
___ Positivamente. O checklist me auxiliou na detecção de defeitos. Talvez não tivesse detectado alguns defeitos caso não o tivesse utilizado.

6. **Como as "Guias de identificação de contexto" auxiliaram na identificação de defeitos relacionados a requisitos de qualidade?**

___ Negativamente. As guias atuaram como um obstáculo. O meu desempenho teria sido melhor se eu não a tivesse utilizado.
___ Neutro. Acho que encontraria os mesmos defeitos caso não tivesse utilizado as guias
___ Positivamente. As guias me auxiliaram na detecção de defeitos. Talvez não tivesse detectado alguns defeitos caso não a tivesse utilizado.

7. **Existe algum item de avaliação que não foi compreendido? Se sim, identifique-o.**

8. **Você utilizaria a técnica em inspeções futuras?**

___Não. ___ Talvez. ___ Sim.

9. **Na sua opinião, como a técnica poderia ser melhorada?**

Itens de avaliação para identificar defeitos:

Taxonomia para categorização de defeitos:

10. **Por favor, registre quaisquer comentários que julgar pertinente.**

Muito obrigado por sua participação!

C.6: Roteiro de Instruções para a Inspeção

Roteiro para a realização do estudo de Inspeção de documento arquitetural:

1. Preencher o formulário de Caracterização de Participante.
2. Iniciar a marcação do tempo de estudo.
3. Ler o documento de Abordagem para Documentação Arquitetural (abordagem_documentacao.pdf).
4. Em seguida, ler o documento de requisitos do eSEE (Infra-Estrutura para a Definição e Execução de Ambientes de Engenharia de Software) – requisitos_eSEE.pdf.
5. Ler o documento arquitetural do eSEE – documentação_arquitetural_eSEE.pdf.
6. Finalizar a marcação do tempo de estudo.
7. Iniciar a marcação do tempo de inspeção.
8. Ler o checklist de apoio à Inspeção.
9. Para cada questão do checklist, marcar a coluna de Sim, Não ou Não Aplicável. Se você julgar que a sua resposta leva à identificação de defeitos na arquitetura, preencher então o relatório de discrepâncias, referenciando a questão do checklist que levou à identificação do defeito.
10. Ler a taxonomia de defeitos do relatório de discrepâncias e classificar o defeito identificado.
11. Indicar os elementos arquiteturais e, se for o caso, seus módulos internos onde o defeito foi identificado. Indicar também os diagramas que o apoiaram na identificação do defeito.
12. Indicar o item do checklist que levou à identificação do defeito e descrever o defeito ou discrepância.
13. Após preencher todos os itens do checklist e preencher o relatório de discrepâncias, finalizar a marcação do tempo de inspeção.
14. Preencher o questionário de avaliação pós-experimento.

Anexo D: Instrumentação do Estudo Experimental de Viabilidade da Etapa de Avaliação de ArchMine

D.1: Formulário de Consentimento para Apoio no Estudo de Viabilidade da Etapa de Avaliação de ArchMine

Apoio no Estudo de Viabilidade da Abordagem de Recuperação de Arquitetura ArchMine com o Checklist de Avaliação de Arquitetura

Eu declaro ter mais de 18 anos de idade e que concordo em participar de um estudo conduzido por Aline Pires Vieira de Vasconcelos, como parte das atividades de doutoramento no Programa de Engenharia de Sistemas e Computação da COPPE/UFRJ. Este estudo visa avaliar se a incorporação do checklist de ArqCheck estendido para a avaliação arquitetural reduz o esforço de avaliação de arquiteturas em ArchMine e possibilita a melhoria da qualidade da arquitetura para reutilização.

PROCEDIMENTO

Este estudo está dividido em cinco etapas: primeiramente, o participante apóia o pesquisador na definição e monitoramento dos cenários de caso de uso da aplicação; em seguida, o pesquisador aplica a abordagem ArchMine refinada para recuperar a arquitetura da aplicação; o participante então preenche o formulário de caracterização, avalia a arquitetura utilizando um *checklist* de avaliação e preenche o formulário número 1 de avaliação pós-experimento; os participantes e o pesquisador se reúnem a fim de consolidar os defeitos identificados na arquitetura e propor correções; por fim, os participantes preenchem o formulário número 2 de avaliação pós-experimento. Com base na análise dos resultados do estudo, o pesquisador será capaz de verificar se o esforço de avaliação das arquiteturas recuperadas com ArchMine foi reduzido aplicando o checklist estendido de ArqCheck e se a aplicação do checklist estendido de ArqCheck possibilita a melhoria da qualidade da arquitetura para reutilização.

BENEFÍCIOS, LIBERDADE DE DESISTÊNCIA

Eu entendo que sou livre para realizar perguntas a qualquer momento ou solicitar que qualquer informação relacionada a minha pessoa, ao departamento ou à empresa envolvidos não seja incluída no estudo. Eu entendo que participo de livre e espontânea vontade com o único intuito de contribuir para o avanço e desenvolvimento de técnicas e processos para a Engenharia de Software.

EQUIPE: **PESQUISADOR RESPONSÁVEL**
MSC. Aline Pires Vieira de Vasconcelos
Programa de Engenharia de Sistemas e Computação - COPPE/UFRJ
PROFESSOR RESPONSÁVEL
Profa. Cláudia M.L. Werner
Programa de Engenharia de Sistemas e Computação - COPPE/UFRJ

Nome (em letra de forma):_____

Assinatura: _____ **Data:**_____

D.2: Questionário de Caracterização dos Participantes

Nome _____

Nível (Ms.c/D.Sc.): ____

Experiência em Desenvolvimento de Software

Qual é a sua experiência com desenvolvimento de software na prática? Por favor, inclua na sua resposta o número de semestres ou número de anos de experiência relevante em desenvolvimento.

_____

Por favor, indique o grau de sua experiência nesta seção seguindo a escala de 5 pontos abaixo:

1 = nenhum

2 = estudei em aula ou em livro

3 = pratiquei em 1 projeto em sala de aula

4 = usei em 1 projeto na indústria

5 = usei em vários projetos na indústria

Experiência com Requisitos

- Experiência escrevendo requisitos 1 2 3 4 5
- Experiência escrevendo casos de uso 1 2 3 4 5
- Experiência revisando requisitos 1 2 3 4 5
- Experiência revisando casos de uso 1 2 3 4 5
- Experiência modificando requisitos para manutenção 1 2 3 4 5

Experiência em Arquitetura de Software

- Experiência em projeto de arquitetura de software 1 2 3 4 5
- Experiência em leitura de documento arquitetural 1 2 3 4 5

Experiência em Projeto

- Experiência em projeto de sistemas 1 2 3 4 5
- Experiência em desenvolver projetos a partir de requisitos e casos de uso 1 2 3 4 5
- Experiência criando projetos Orientado a Objetos 1 2 3 4 5

- Experiência lendo projetos Orientado a Objetos 1 2 3 4 5
- Experiência com Unified Modeling Language (UML) 1 2 3 4 5
- Experiência alterando projeto para manutenção 1 2 3 4 5

Experiência em Reutilização de Software

- Experiência em desenvolver modelos para reutilização, como *frameworks* 1 2 3 4 5
- Experiência em reutilizar *frameworks* 1 2 3 4 5
- Experiência em desenvolver componentes 1 2 3 4 5
- Experiência em reutilizar componentes 1 2 3 4 5

Outras Experiências

- Experiência com inspeções de software? 1 2 3 4 5
- Experiência na utilização da abordagem ArqCheck? 1 2 3 4 5

Para a questão a seguir, considere a seguinte escala de experiência:

1 = nenhuma

2 = já li a respeito

3 = já estudei a respeito

4 = já utilizei aplicações

5 = já desenvolvi aplicações

- Experiência no domínio para o qual a arquitetura está sendo avaliada? 1 2 3 4 5

D.3: Descrição da Abordagem de Documentação Arquitetural Utilizada

Os modelos recuperados pela ArchMine são representados no OdysseyLight. A arquitetura é representada na Visão Estrutural do ambiente (*Structural View*), através de um diagrama de pacotes da UML, conforme exemplo apresentado na Figura D.1. Os pacotes que compõem a arquitetura ficam subordinados ao pacote "architecture" na árvore semântica do OdysseyLight dentro da Structural View. A aplicação apresentada é do domínio escolar e os elementos arquiteturais Alunos, Turmas, Professores, Cursos e Biblioteca são representados através de pacotes da UML. As dependências entre os pacotes revelam que existem relacionamentos entre os seus módulos internos, i.e. entre as suas classes, e indicam que existe um acoplamento entre eles.

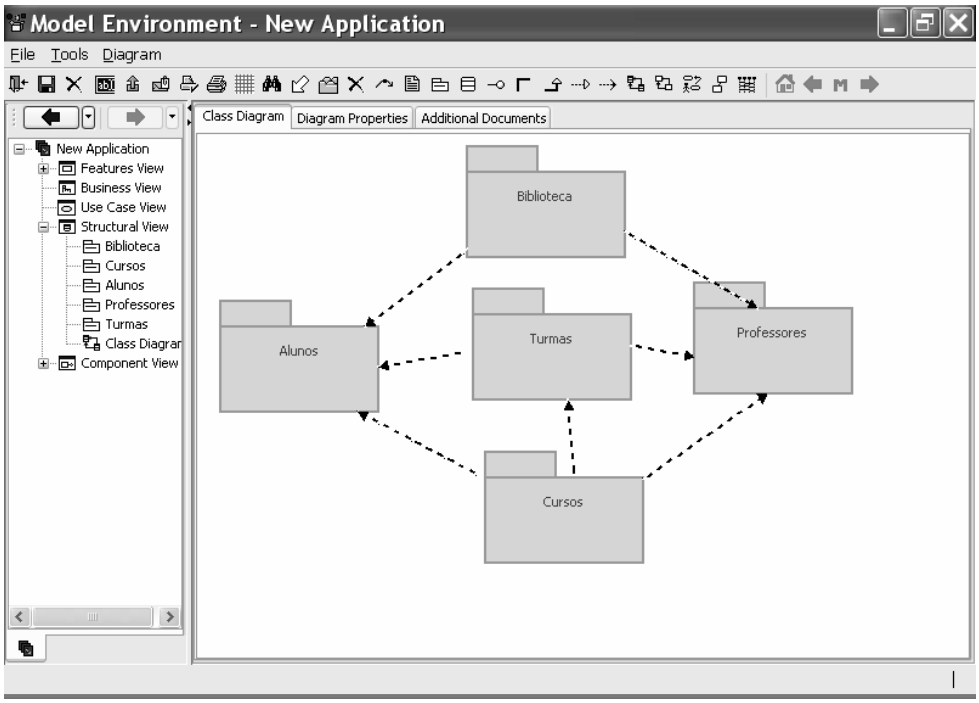

Figura D.1. Modelo arquitetural para uma aplicação de controle escolar.

Cada elemento arquitetural possui um conjunto de módulos internos representados através das suas classes, conforme mostra a Figura D.2 para o elemento arquitetural Cursos. Se o desenvolvedor desejar visualizar os relacionamentos entre as classes de um elemento arquitetural, ele deve arrastá-las para um diagrama de classes. A cada elemento arquitetural, é associado o conjunto de cenários de casos de uso que lhe deu origem. Esses cenários ficam descritos na seção de documentação do elemento

arquitetural ou pacote no OdysseyLight. As classes internas de um elemento arquitetural também têm associados os cenários de caso de uso que ela implementa.

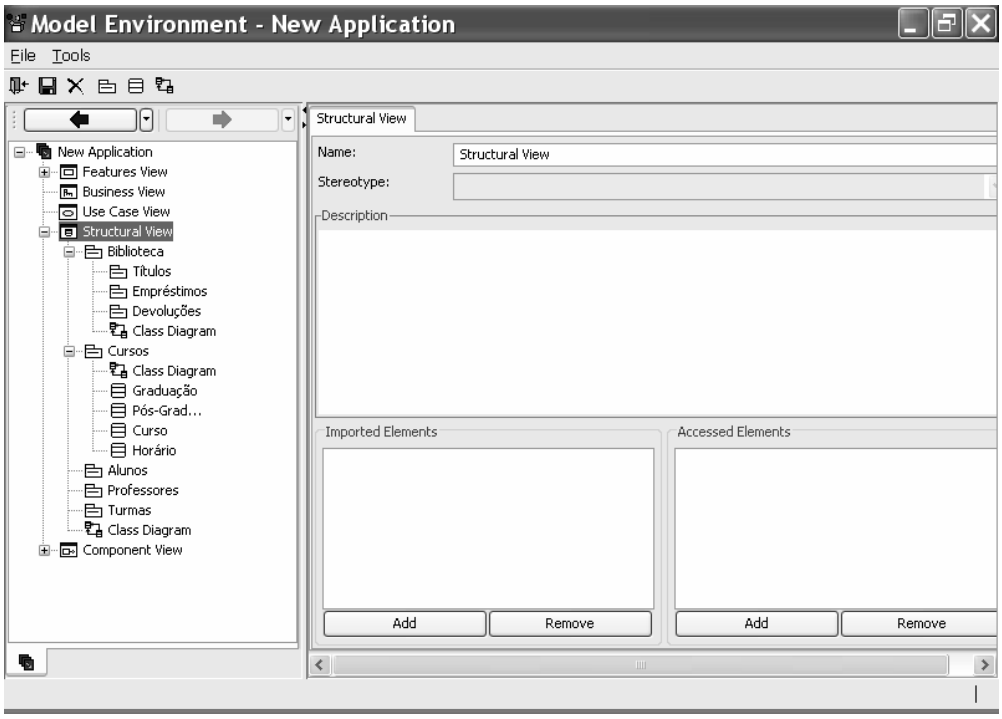

Figura D.2. Detalhamento dos elementos arquiteturais para uma aplicação de controle escolar.

D.4: *Checklist* Utilizado no Estudo de Viabilidade da Etapa de Avaliação de ArchMine

CheckList para Inspeção de Documento Arquitetural

=> **Instruções**:

- Avalie a documentação arquitetural através dos itens de avaliação abaixo.
- A opção NA (Não Aplicável) deve ser marcada se for considerado que não é possível aplicar o item para avaliar a documentação arquitetural em questão.
- Para os itens de avaliação referentes ao Requisito de Qualidade Reusabilidade, os elementos arquiteturais devem ser identificados através do cenário de Reusabilidade e da "Guia de identificação de contexto".

Inspetor: _____

Tempo de Inspeção: _____

Nº	**Itens de avaliação da consistência das representações entre os diagramas**	Sim	Não	NA
1	Nos diagramas, existe algum elemento arquitetural que não possui relacionamentos, ficando isolado dos demais?	Def.		
2	Todos os elementos arquiteturais, identificados através do seu nome, foram representados através da mesma abstração nos diferentes diagramas?		Def.	
Itens de avaliação da consistência das representações entre os diagramas (específicos à abordagem de documentação arquitetural utilizada)				
Nº	**Visão Estrutural**	Sim	Não	NA
3	Os módulos internos de cada elemento arquitetural foram descritos na Visão Estrutural?		Def.	
4	Todo relacionamento definido com um elemento arquitetural foi devidamente mapeado para um de seus módulos internos?		Def.	
Nº	**Visão Dinâmica**	Sim	Não	NA
5	Todo módulo/elemento arquitetural representado na visão Dinâmica foi descrito na Visão Estrutural?		Def.	x
6	Todo fluxo entre dois elementos arquiteturais pode ser mapeado para algum relacionamento da Visão Estrutural?		Def.	x
Nº	**Itens de avaliação do atendimento aos requisitos**	Sim	Não	NA
7	Todo elemento arquitetural tem a sua presença na arquitetura justificada por um conjunto de requisitos?		Def.	
8	Todo requisito funcional ou de qualidade ou adicional, criado pelas decisões arquiteturais, foi atendido por algum elemento arquitetural?		Def.	x
Itens de avaliação de atendimento aos requisitos de qualidade				
Itens relacionados a Reusabilidade **Requisito de Reusabilidade**: os elementos arquiteturais recuperados aplicando a abordagem				

ArchMine devem ser utilizados para apoiar a geração de uma Arquitetura de Referência de domínio, reutilizada na instanciação de aplicações no domínio.

Cenário de Reusabilidade	
Fonte do estímulo	Engenheiro de Domínio
Estímulo	Reutilização dos elementos arquiteturais na geração de uma Arquitetura de Referência do domínio.
Contexto do Sistema	Processo de Engenharia de Domínio
Artefato	Todos os elementos arquiteturais recuperados aplicando a abordagem ArchMine e representados através de pacotes na *Structural View* do ambiente Odyssey.
Resposta	Geração de uma Arquitetura de Referência do domínio.
Medida de Resposta	-------

Guia de identificação de contexto: de acordo com o requisito de reusabilidade que se deseja avaliar, identificar os elementos arquiteturais que visam ser reutilizados em novas aplicações (*Elementos Reutilizáveis*), seus módulos internos e relacionamentos com demais elementos arquiteturais. A identificação dos *Elementos Reutilizáveis* pode ser feita principalmente a partir da análise do estímulo e artefato descritos nos cenários de reusabilidade.

		Sim	Não	NA
9	Coesão pode ser medida pela extensão em que os módulos de um componente ou elemento arquitetural estão relacionados. Neste sentido, as responsabilidades dos módulos internos de um *Elemento Reutilizável* pertencem a um mesmo contexto, ou seja, visam atingir um mesmo propósito ou são utilizadas nos mesmos cenários de casos de uso?		Def.	
10	Ainda considerando a Coesão dos elementos arquiteturais, do ponto de vista dos serviços ou conceito que o *Elemento Reutilizável* representa, existem módulos que deveriam estar alocados nele, em função das similaridades das suas responsabilidades ou cenários de casos de uso com os seus módulos internos, mas que estão alocados num outro elemento arquitetural?	Def.		
11	Ainda considerando Coesão, é possível identificar grupos de *Elementos Reutilizáveis* com funcionalidades ou responsabilidades similares que, para compor um componente de software, poderiam ser agrupados em um único elemento?	Def.		
12	Acoplamento forte entre elementos arquiteturais reduz a sua reusabilidade por causa da necessidade de se utilizar estes elementos arquiteturais em conjunto. Nesse contexto, existem relacionamentos entre o *Elemento Reutilizável* e demais elementos arquiteturais que prejudica a sua reutilização?	Def.		

13	Ainda considerando o acoplamento, existem relacionamentos entre o *Elemento Reutilizável* e demais elementos arquiteturais que justificaria o seu agrupamento em um único *Elemento Reutilizável*?	Def.		

Glossário dos termos utilizados no *checklist*:

1 – **Elemento Arquitetural**: elemento da arquitetura utilizado para agrupar módulos que devem pertencer a um mesmo contexto, ou seja, que devem possuir responsabilidades similares. Implementa um conjunto de funcionalidades (ou serviços) relacionadas ou conceito do domínio. A representação da sua estrutura interna é essencial para o entendimento da solução. Em UML, elementos arquiteturais podem ser representados através de pacotes, cujos módulos internos são representados através de classes.

2 – **Módulo (ou módulo interno)**: elemento pertencente ao mais baixo nível de abstração, o qual compõe os elementos arquiteturais. Este elemento realiza diversas responsabilidades e se relaciona com outros elementos visando atender a um conjunto de requisitos. Em UML, módulos podem ser representados através de classes.

3 - **Relacionamento entre elementos arquiteturais**: qualquer ligação (ex: dependência) entre dois elementos arquiteturais.

4 - **Relacionamento entre módulos internos**: relacionamento entre módulos de elementos arquiteturais.

5 – **Componentes de software**: componentes de software são artefatos auto-contidos, claramente identificáveis, que descrevem ou realizam uma função específica e têm interfaces claras, documentação apropriada e um grau de reutilização definido (SAMETINGER, 1997).

6 – **Cenários de Casos de Uso**: funcionalidades da aplicação que o elemento arquitetural implementa através da colaboração com outros elementos arquiteturais.

***Rationale* para as questões do *checklist* de avaliação de Reusabilidade:**

Convém ressaltar que, embora a idéia por trás do *checklist* é a de que ele possa ser usado para avaliar a reusabilidade de elementos arquiteturais genericamente, independente do paradigma de desenvolvimento adotado, o *rationale* para as questões do checklist foi retirado da literatura de DBC (Desenvolvimento Baseado em Componentes). Isso se deve ao fato de DBC representar um paradigma que solidifica princípios e métodos para a estruturação de artefatos reutilizáveis bem definidos. Dessa forma, o conceito de componente nos itens a seguir pode se aplicar a componentes de

software propriamente ou a elementos arquiteturais que pretendem ser reutilizados em outras aplicações. Segue então os itens que dão embasamento às questões apresentadas no *checklist.*

Acoplamento: acoplamento forte entre componentes reduz a sua reusabilidade por causa da necessidade de se utilizar estes componentes em conjunto (VITHARANA *et al.*, 2004). Acoplamento entre componentes ou entre elementos arquiteturais pode ser definido como: a extensão em que classes no componente se relacionam a outras classes que não estão neste componente (VITHARANA *et al.*, 2004). Expressando a importância do acoplamento, HOPKINS (2000) afirma que "comunicação inter-componentes é bastante cara em termos de tempo e recursos de plataforma".

Coesão: refere-se à força de associação dos elementos em um sistema (WALLNAU *et al.*, 2002). Coesão pode ser medida pela extensão em que as classes de um componente ou elemento arquitetural estão relacionadas (VITHARANA *et al.*, 2004). Segundo VITHARANA *et al.* (2004), coesão representa um importante critério de projeto na construção de componentes reutilizáveis. Além desse aspecto, VITHARANA *et al.* (2004) afirmam ainda que uma vez que os requisitos para uma aplicação particular tendem a estar relacionados, um componente ou elemento arquitetural mais coeso vai satisfazer a requisitos relacionados mais freqüentemente, aumentando, portanto, a sua reusabilidade.

Acoplamentos entre Componentes: em (BLOIS, 2006), é argumentado que quando se tem um grande número de componentes na arquitetura, estes componentes podem trocar um grande volume de mensagens e, muitas vezes, esta interação ocorre entre um grupo específico de componentes. Dessa forma, BLOIS (2006) sugere que componentes que trocam muitas mensagens entre si sejam agrupados em um único elemento arquitetural. Os critérios oferecidos em (BLOIS, 2006) para apoiar estes agrupamentos se baseiam, dentre outros, em: interfaces providas e requeridas; e componentes requeridos e requerendo outros. Este último critério avalia quantos e quais componentes são requeridos por um outro componente e pode ser aplicado a qualquer tipo de elemento arquitetural. Se um componente requer dois ou mais (i.e. um pequeno número) de componentes, é sugerido o agrupamento dos mesmos. Pode-se formar grupos, neste caso, onde 2 ou 3 componentes só possuam dependências entre si e um dos componentes do grupo apenas possui dependências com outros componentes.

D.5: Questionário de Avaliação Pós-Experimento 1 do Estudo de Viabilidade da Etapa de Avaliação de ArchMine com ArqCheck Estendida e Adaptada

Questionário de Avaliação Pós-Experimento

Inspetor : _____

Avaliação da Arquitetura Recuperada e do Checklist de Avaliação

---

Este é um formulário de perguntas subjetivas, as quais devem ser respondidas com um dos valores do conjunto {"Sim", "Não", "Parcialmente"}, seguido de uma justificativa.

Q1. O modelo arquitetural recuperado representa um possível modelo da arquitetura da aplicação?

_____
_____
_____
_____
_____

Q2. Na sua opinião, os defeitos identificados com o checklist se encontram na arquitetura original ou nos agrupamentos recuperados com ArchMine?

_____
_____
_____
_____
_____

Q3. Como você classificou o grau de dificuldade da aplicação do checklist?

___Muito Fácil ___Fácil ___Mediano ___Difícil ___Muito Difícil

Q4. Na sua opinião, quais aspectos da técnica tornam sua aplicação fácil/difícil de usar?

_____

_____

_____

_____

_____

Q5. Como o checklist o auxiliou a identificar defeitos na arquitetura?

___ Negativamente. O checklist atuou como um obstáculo. O meu desempenho teria sido melhor se eu não o tivesse utilizado.

___ Neutro. Acho que encontraria os mesmos defeitos caso não tivesse utilizado o checklist

___ Positivamente. O checklist me auxiliou na detecção de defeitos. Talvez não tivesse detectado alguns defeitos caso não o tivesse utilizado.

Q6. Existe algum item de avaliação que não foi compreendido?

Se sim, identifique-o.

_____

_____

_____

_____

_____

Q7. Por favor, registre quaisquer comentários que julgar pertinente.

_____

_____

_____

_____

Muito obrigado por sua participação!

D.6: Questionário de Avaliação Pós-Experimento 2 do Estudo de Viabilidade da Etapa de Avaliação de ArchMine com ArqCheck Estendida e Adaptada

Questionário de Avaliação Pós-Experimento 2

Inspetor:

_____

Q1. A arquitetura reestruturada em função das discrepâncias identificadas conduz a elementos arquiteturais que podem ser utilizados para compor uma arquitetura de referência pro domínio?

_____
_____
_____
_____
_____

Q2. Por favor, registre quaisquer comentários que julgar pertinente em relação ao checklist ou à arquitetura recuperada com ArchMine.

_____
_____
_____
_____
_____

Muito obrigado por sua participação!

Anexo E: Resumo da Taxonomia de Características da Notação Odyssey-FEX (OLIVEIRA, 2006b)

Ícone	Tipo de Característica	
	**Características de Domínio** – Características intimamente ligadas à essência do domínio. Representam as funcionalidades e/ou os conceitos do modelo e correspondem a casos de uso e componentes estruturais concretos.	**Características de Análise**
	**Características de Entidade** – São os atores do modelo. Entidades do mundo real que atuam sobre o domínio. Podem, por exemplo, expor a necessidade de uma interface com o usuário ou de procedimentos de controle.	
	**Características de Ambiente Operacional -** Características que representam atributos de um ambiente que uma aplicação do domínio pode usar e operar. Ex: tipo de terminal, sistemas operacionais, bibliotecas etc.	**Características de Projeto (Tecnológicas)**
	**Características de Tecnologia de Domínio -** Características que representam tecnologias utilizadas para modelar ou implementar questões específicas de um domínio. Ex: métodos de navegação em um domínio de aviões.	
	**Características de Técnicas de Implementação** – Características que representam tecnologias utilizadas para implementar outras características, podendo ser compartilhadas por diversos domínios. Ex: técnicas de sincronização.	

As características podem estar ligadas via relacionamentos UML, i.e. associação, dependência, implementação e herança, via relacionamentos específicos do modelo de características propostos em (OLIVEIRA, 2006b), como implementação e alternativo, ou via regras de composição. Implementação (*implemented-by*) é um relacionamento entre características de domínio e características tecnológicas. O alternativo (*alternative*) é um relacionamento entre um ponto de variação e suas variantes. Regras de composição definem relacionamentos de dependência entre 2 ou mais características (i.e. regras de composição inclusivas) ou mútua exclusividade (i.e. regras de composição exclusivas).

Anexo F: Heurísticas de Mapeamento do Modelo de Classes para o Modelo de Características no ambiente Odyssey

Modelo de Classes	Modelo de Características
Classe	Característica Conceitual, Funcional ou Tecnológica
Atributo	Característica Conceitual ligada por Composição à Característica que representa à Classe que o contém
Método	Característica Funcional ligada por Composição à Característica que representa à Classe que o contém
Pacote	Característica Conceitual, Funcional ou Tecnológica
Dependência, Associação com Multiplicidade Mínima 1	Regra Composição Inclusiva
Classe, Método, Pacote (estereotipados com <<xor>>)	Regra Composição Exclusiva
Superclasse, Interface	Ponto de Variação
Subclasse, Classe que implementa uma Interface	Variante
Opcionalidade do Elemento (i.e. mandatório ou opcional)	Opcionalidade do Elemento
Variabilidade do Elemento (i.e. ponto de variação, variante ou invariante)	Variabilidade do Elemento
Valor da Propriedade "Não Definido" (i.e. verdadeiro ou falso)	Valor da Propriedade "Não Definido" (i.e. verdadeiro ou falso)
Associação com Multiplicidade Mínima 0, Composição ou Agregação	Associação com Multiplicidade Mínima 0, Composição ou Agregação
Relação de Hierarquia do Pacote para Classe	Agregação

Essas heurísticas estão baseadas na semântica estabelecida pelo metamodelo da notação Odyssey-FEX e pelo metamodelo do modelo de classes da UML, versão 1.4, tendo sido derivadas com base na análise de modelos em 3 diferentes domínios conduzida em (OLIVEIRA, 2006b).

Anexo G: Versão Simplificada do Checklist Original da Abordagem ArqCheck Utilizado em uma Avaliação Arquitetural

***CheckList* para inspeção de modelos arquiteturais utilizado em um dos estudos de viabilidade de ArqCheck conduzidos em (BARCELOS, 2006) (versão simplificada)**

=> **Instruções**:

- Avalie a documentação arquitetural através dos itens de avaliação abaixo.
- A opção NA (Não Aplicável) deve ser marcada se for considerado que não é possível aplicar o item para avaliar a documentação arquitetural em questão.
- Para os itens de avaliação dos requisitos de qualidade (21-42), a documentação arquitetural deve ser avaliada em relação aos requisitos de qualidade identificados na "**Guia de identificação de contexto**". Caso os requisitos não tenham sido identificados, o inspetor deve avaliar somente se o item é aplicável ou não.

Equipe/Revisor: _____

Nº	**Itens de avaliação da consistência das representações entre os diagramas**	Sim	Não	NA
1	Nos diagramas, existe algum módulo/cluster que não possui relacionamentos, ficando isolado dos demais?			
2	Todos os elementos arquiteturais, identificados através do seu nome, foram representados através da mesma abstração nos diferentes diagramas?			
	Itens de avaliação da consistência das representações entre os diagramas (específicos à abordagem de documentação arquitetural utilizada)			
Nº	**Visão Modular**	Sim	Não	NA
3	Os módulos internos de cada cluster foram descritos em algum diagrama da visão modular?			
4	Todo relacionamento definido com um cluster foi devidamente mapeado para um de seus módulos internos?			
Nº	**Visão Dinâmica**	Sim	Não	NA
5	Toda porta/interface possui um nome, é utilizada com um único propósito e de forma única?			
6	Os fluxos de execução, descritos na visão dinâmica, alocam todos os módulos definidos na visão modular?			
7	Todo módulo/cluster representado na visão dinâmica foi descrito na visão modular?			
8	Todo fluxo entre dois elementos arquiteturais pode ser mapeado para algum relacionamento da visão modular?			

9	Todo relacionamento descrito na visão modular pode ser mapeado para algum fluxo de comunicação, de dados ou de controle da visão dinâmica?			
Nº	**Visão de Alocação**	Sim	Não	NA
10	Todo módulo/cluster, representado na visão de alocação, foi descrito na visão modular?			
	...			
Nº	**Visão de Contexto Geral**	Sim	Não	NA
13	**...**			
Nº	**Itens de avaliação do atendimento aos requisitos**	Sim	Não	NA
15	Todo elemento arquitetural tem a sua presença na arquitetura justificada por um conjunto de requisitos?			
16	Todo requisito funcional ou de qualidade ou adicional, criado pelas decisões arquiteturais, foi atendido por algum elemento arquitetural?			
17	As responsabilidades atribuídas a um elemento arquitetural estão relacionadas aos requisitos que ele atende?			
18	Nos diagramas da visão dinâmica, todo fluxo de execução é justificado por um conjunto de requisitos?			
	...			
Itens de avaliação da abordagem de atendimento aos requisitos de qualidade				
Nº	**Itens Relacionados a Desempenho**	Sim	Não	NA
Requisito RNF1 **Cenário Desempenho** **....** **Guia de identificação de contexto**: de acordo com os requisitos de desempenho que se deseja avaliar, identificar os fluxos de execução do sistema (Fluxo) relacionados ao atendimento dos requisitos selecionados e os elementos que participam desses fluxos (Elementos Participantes). Essa identificação pode ser feita através da análise do estímulo, do artefato e da resposta descritos nos requisitos selecionados.				
21	Todos os *Elementos Participantes* foram alocados em um mesmo nó computacional visando aumentar o desempenho do fluxo?			
	...			
Nº	**Itens Relacionados a Disponibilidade**	Sim	Não	NA
Requisito RNF2 **Cenário de Disponibilidade** **....** **Guia de identificação de contexto**: de acordo com os requisitos de disponibilidade selecionados, identifique os elementos arquiteturais que devem possuir alta disponibilidade (Elemento Principal). A identificação pode ser feita a partir do artefato e da resposta descritos nos cenários de disponibilidade.				
26	Existe algum elemento arquitetural responsável por verificar periodicamente a disponibilidade dos Elementos Principais?			

	...			
Nº	**Itens Relacionados a Testabilidade**	Sim	Não	NA

RNF3
Cenário de Testabilidade
...
Guia de identificação de contexto: de acordo com os requisitos de testabilidade que se deseja avaliar, identificar os elementos que participam do atendimento às funcionalidades que se deseja permitir testabilidade (Elemento Testável). Essa identificação pode ser feita através de uma análise do estímulo e artefato dos requisitos selecionados.

29	Todo Elemento Testável disponibiliza interfaces de comunicação que permitem a sua substituição?			
	...			
Nº	**Itens Relacionados a Usabilidade**	Sim	Não	NA

Requisito RNF4
A infra-estrutura deve permitir que a execução de um processo ocorra de forma desacoplada das atividades de configuração e instanciação de modelos.

Cenário de Usabilidade	
Fonte do estímulo	Usuário Engenheiro
Estímulo	Modelar ou instanciar processos
Contexto do Sistema	Durante a execução normal da infra-estrutura
Artefato	Ambientes para instanciação e configuração
Resposta	Novos modelos ou ambientes instanciados
Medida de Resposta	Não afeta a execução de ambientes previamente instanciados

Guia de identificação de contexto: para atender a requisitos de usabilidade, funcionalidades ou elementos arquiteturais podem ser definidos ou adaptados. Portanto, de acordo com os requisitos de usabilidade que se deseja avaliar, caso seja necessário, essas funcionalidades (Funcionalidades Envolvidas) devem ser identificadas. Além disso, devem-se identificar quais elementos arquiteturais serão adaptados ou criados para atender diretamente ao Requisito ou às Funcionalidades Adicionais (Elementos Relacionados). A identificação das funcionalidades e dos elementos arquiteturais pode ser feita através da análise do estímulo, da resposta e da medida de resposta dscritos nos requisitos.

31	Toda Funcionalidade Envolvida, caso tenha sido definida, está associada a algum elemento arquitetural?			
	...			
Nº	**Itens Relacionados a Modificabilidade**	Sim	Não	NA

Requisito RNF5
A infra-estrutura deve ser construída de forma a permitir que os desenvolvedores modifiquem a abordagem de recuperação e persistência dos artefatos manuseados, sem que seja necessário realizar alterações em outras funcionalidades. Essa alteração pode ocorrer durante o desenvolvimento do sistema ou durante a sua manutenção.

Cenário de Modificabilidade	
Fonte do estímulo	Desenvolvedores
Estímulo	Modificação da abordagem de manipulação de dados
Contexto do Sistema	Desenvolvimento do sistema / manutenção do sistema
Artefato	Repositório de dados
Resposta	O sistema deve manipular os dados como antes da modificação
Medida de Resposta	----

Guia de identificação de contexto: de acordo com os requisitos de modificabilidade que se deseja avaliar, identificar os elementos arquiteturais que podem ser modificados (Elemento Modificável) e também os elementos que possuem relacionamentos com os Elementos Modificáveis de forma direta (Elemento Relacionado). A identificação dos Elementos Modificáveis pode ser feita principalmente através da análise do estímulo e contexto dos requisitos selcionados.

33	As responsabilidades de um Elemento Relacionado pertencem a um mesmo contexto, ou seja, visam atingir um mesmo propósito, manipulam um mesmo tipo de dado ou são utilizadas em um mesmo fluxo de execução? **...**			
Nº	**Itens Relacionados a Segurança**	Sim	Não	NA

RequisitoRNF 6
Cenário de Segurança
....
Guia de identificação de contexto: de acordo com os requisitos de segurança que se deseja avaliar, três tipos de elementos devem ser identificados: os elementos arquiteturais seguros (Elemento Seguro) que são acessados de forma restrita, os elementos que possuem relacionamentos com os elementos seguros (Elemento Relacionado) e os elementos que buscam garantir o acesso seguro ao Elemento Seguro (Elemento Gerenciador). Para que o Elemento Seguro seja identificado, o estímulo e o artefato dos requisitos selecionados devem ser analisados.

38	Todo Elemento Seguro implementa alguma funcionalidade ou possui algum relacionamento com um Elemento Gerenciador que autentique o acesso aos seus dados ou funcionalidades?			
	...			