UMA ABORDAGEM PARA A CRIAÇÃO DE
ARQUITETURAS DE REFERÊNCIA DE DOMÍNIO A PARTIR DA
COMPARAÇÃO DE MODELOS ARQUITETURAIS DE APLICAÇÕES

Guilherme Zanetti Kümmel

DISSERTAÇÃO 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 MESTRE EM CIÊNCIAS EM ENGENHARIA DE SISTEMAS E COMPUTAÇÃO.

Aprovada por:

Profa. Cláudia Maria Lima Werner, D.Sc.

Profa. Aline Pires Vieira de Vasconcelos, D.Sc.

Prof. Geraldo Bonorino Xexéo, D.Sc.

Profa. Rosângela Aparecida Dellosso Penteado, D.Sc.

RIO DE JANEIRO, RJ – BRASIL
JULHO DE 2007

KÜMMEL, GUILHERME ZANETTI

Uma Abordagem para a Criação de Arquiteturas de Referência de Domínio a partir da Comparação de Modelos Arquiteturais de Aplicações [Rio de Janeiro] 2007

XIII, 130 p., 29,7 cm (COPPE/UFRJ, M.Sc., Engenharia de Sistemas e Computação, 2007)

Dissertação – Universidade Federal do Rio de Janeiro, COPPE

1. Arquitetura de Referência
2. Domínio
3. Opcionalidade
4. Variabilidade

I. COPPE/UFRJ II. Título (série)

A todos que, de alguma forma,
me ajudaram a chegar até aqui,
e que certamente estarão comigo.

Agradecimentos

Agradeço sinceramente à professora Claudia Werner pela grande oportunidade de fazer o mestrado. Sua orientação, paciência, seriedade, competência e exigência foram fundamentais para a realização deste trabalho. Levarei sempre comigo esta gratidão.

À Aline Vasconcelos que me incentivou e orientou em todos os momentos, sempre solícita, pronta para ajudar, praticamente incansável. Sua participação foi, e continua sendo, muito importante.

Ao Leonardo Murta que, com sua competência, humildade e profundos conhecimentos sobre diversos assuntos, guiou meu aprendizado nas diversas tecnologias usadas.

Ao professor Guilherme Travassos, com quem muito aprendi sobre Engenharia de Software. Seus ensinamentos determinaram como deveriam ser feitos os estudos experimentais deste trabalho.

Aos colegas da COPPE que apoiaram o estudo experimental da abordagem: Marco Lopes, Eldanae Teixeira, Paula Fernandes e Anderson Marinho.

Aos colaboradores da Prolink que apoiaram o estudo experimental da ferramenta: Daniel Schmitz, Filipe Manganelli, André Pena e Marcelo Furtado.

Aos colegas da COPPE que de alguma forma colaboraram para a realização dos trabalhos: Ana Maria Moura, Ana Paula Blois, dentre outros.

A todos da área da produção da Prolink, em especial Adriano Mendes, Hélcio Maciel e Carlos Eduardo Dellagarza pelo apoio e compreensão.

Ao amigo Marco Antonio Araújo pela indicação durante meu processo de seleção na COPPE.

Aos amigos e companheiros de viagem: Wagner Arbex e José Fortuna.

Segundo o Dalai Lama, o valor que damos a algo, é proporcional a tudo aquilo que tivemos de abrir mão para conquistá-lo. Desta forma, posso afirmar, não somente através de sentimentos, mas também em função de tudo aquilo que tive que deixar de fazer durante este período, o grande valor que este mestrado representa para mim.

Portanto gostaria de fazer um agradecimento a todos aqueles que deixaram de estar comigo em vários momentos, por estar dedicado a este trabalho, em especial minha esposa Juliana, minha família e os amigos Marcelo Castanha e Carlos Cavalari.

Agradecimento especial também aos professores Geraldo Xexéo e Rosângela Dellosso Penteado por terem aceitado participar da minha banca de mestrado, e por dispensarem seu tempo e conhecimento dedicados a esta leitura.

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

UMA ABORDAGEM PARA A CRIAÇÃO DE ARQUITETURAS DE REFERÊNCIA DE DOMÍNIO A PARTIR DA COMPARAÇÃO DE MODELOS ARQUITETURAIS DE APLICAÇÕES

Guilherme Zanetti Kümmel

Julho / 2007

Orientadoras: Cláudia Maria Lima Werner
Aline Pires Vieira Vasconcelos

Programa: Engenharia de Sistemas e Computação

O aumento em tamanho e complexidade do software, somado a uma maior demanda pela construção de software em menor tempo e com mais qualidade, são desafios enfrentados pelas empresas atualmente. Além disso, em geral, empresas tendem a desenvolver sistemas de software similares (i.e. no mesmo domínio), o que vem motivando cada vez mais a construção de sistemas com abordagens de reutilização, como Engenharia de Domínio (ED) e Linha de Produtos (LP). Em um dado domínio, arquiteturas de referência de domínio (DSSA – *Domain Specific Software Architecture*) representam um papel fundamental para seu entendimento e para a instanciação de aplicações similares. Nesse contexto, é proposta, nesta dissertação, uma abordagem para a comparação de arquiteturas de aplicações, visando à detecção de suas similaridades, diferenças e variabilidades, para que, com base nestas informações, seja possível apoiar o Engenheiro de Domínio na criação de uma DSSA. A abordagem, denominada ArchToDDSA, e sua ferramenta de apoio, ArchToDSSATool, apresentam diferenciais em relação a outras abordagens, como um dicionário de sinônimos, detecção semi-automática de variabilidades, e apoio à definição dos elementos para compor a DSSA. O trabalho foi desenvolvido no contexto do ambiente Odyssey, que visa apoiar a reutilização de software por meio de abordagens como ED, LP e DBC (Desenvolvimento Baseado em Componentes). Assim, ArchToDSSATool, está integrada com ferramentas que permitem a extração de arquiteturas de sistemas legados e a modelagem da DSSA criada, diferenciando-a em relação às outras abordagens e implementações estudadas.

Abstract of Dissertation presented to COPPE/UFRJ as a partial fulfillment of the requirements for the degree of Master of Science (M.Sc.)

AN APPROACH FOR THE CREATION OF DOMAIN REFERENCE ARCHITECTURES BYCOMPARING APPLICATION ARCHITECTURAL MODELS

Guilherme Zanetti Kummel

July / 2007

Advisors: Cláudia Maria Lima Werner
Aline Pires Vieira Vasconcelos

Department: Computer and Systems Engineering

The increase in software size and complexity, added to a bigger demand for the software construction in less time and higher quality, are challenges currently faced for the companies. Moreover, in general, companies tend to develop similar systems (i.e., software in a specific domain). This fact increases the development of systems using some kind of reuse approach, such as Domain Engineering (DE) and Product Lines (PL). In a specific domain, a DSSA (Domain Specific Architecture Software) represents a basic role for its agreement and for instantiation of similar applications. In this context, it is proposed, in this dissertation, an approach for the comparison of application architectures, aiming at the detection of its commonality and variability. Thus based on this information, it can be possible to support the Domain Engineer in the DSSA creation process. The approach, named ArchToDDSA, and its tool support, ArchToDSSATool, present some differential in relation to other approaches, such as a synonymous dictionary, a semi-automatic process of variability detection, and support to the selection of elements to compose the DSSA. The presented work was developed in the context of Odyssey SDE, which aims to support the software reuse by using approaches such as DE, PL and CBD (Component-Based Development). As a result, ArchToDSSATool is integrated with tools that allow the extraction of legacy systems architectures and the modeling of the created DSSA, which make this approach different from the other approaches and implementations that have been studied.

Índice

Índice de Figuras

Índice de Tabelas

Capítulo 1: Introdução

A reutilização de software, isto é, o processo pelo qual sistemas são criados a partir de software preexistente, ao invés de serem criados do zero (KRUEGER, 1992) vem, cada vez mais, se mostrando uma realidade. A busca pelo aumento da qualidade e redução do esforço no desenvolvimento de software através da reutilização é crescente na comunidade de desenvolvedores. No entanto, para alcançar tal objetivo, a simples reutilização de código-fonte não é suficiente, fazendo-se necessário reutilizar elementos que estão implícitos a essa reutilização, como os elementos de análise e projeto. Faz-se necessário reutilizar todo o conhecimento envolvido no desenvolvimento de software. (BARROCA *et al.*, 2005; WERNER & BRAGA, 2005)

Um dos fatores importantes para a reutilização, juntamente com o processo e a organização, é a arquitetura de software (JACOBSON *et al.*, 1997; BARROCA *et al.*, 2005). A arquitetura de software, estrutura global de um sistema, pode ser definida como: a descrição dos elementos a partir dos quais os sistemas são construídos (componentes), as interações entre estes elementos (conectores), os padrões que guiam sua composição e as restrições sobre estes padrões (SHAW & GARLAN, 1996).

A montagem da arquitetura deve permanecer independente de qualquer implementação ou tecnologia, utilizando apenas especificações de componentes (ou, mais genericamente, de elementos arquiteturais) para representar os serviços existentes na arquitetura (D'SOUZA & WILLS, 1999), (MENDES, 2002). Entretanto, identificar a melhor forma de realizar a decomposição de um sistema em elementos arquiteturais, definindo como eles devem interagir, é uma tarefa difícil. No contexto da reutilização de software, devemos ainda eleger estes elementos de tal forma que possam ser reaproveitados em outros sistemas de mesma natureza.

PRIETO DIAZ (1987) define esta coleção de "sistemas de mesma natureza", ou melhor, de aplicações que compartilham problemas comuns, como um *domínio*, e o processo de identificação e organização do conhecimento sobre um domínio de problema para suportar sua descrição e solução, como *análise de domínio*. A *análise de domínio*, onde o conhecimento existente sobre o domínio é estudado e um "modelo de domínio" é gerado; o *projeto de domínio*, onde é criada uma arquitetura que suporte os requisitos identificados na fase de análise; e, a *implementação do domínio*, onde artefatos reutilizáveis são implementados ou adquiridos para compor a arquitetura

criada, constituem as três fases distintas da engenharia de domínio (ED). A ED, bem como sua vertente industrial, denominada linha de produtos de software (LPS), são métodos de sistematização para a reutilização em um domínio específico.

A arquitetura resultante da fase de projeto da engenharia de domínio, especificada através de elementos especializados no domínio, generalizados para uso efetivo dentro do domínio e compostos em um padrão de estrutura eficaz para a construção de uma família de aplicações, é definida como a arquitetura de referência do domínio, ou DSSA (*Domain Specific Software Architecture*) (HAYES, 1994). A criação de uma arquitetura de referência é importante na engenharia de domínio, pois ela é a base para a instanciação de aplicações em um domínio de aplicação específico.

1.1 - Motivação

Os métodos de engenharia de domínio, em geral, possuem maior ênfase na fase de análise, dando pouco apoio às atividades relacionadas à criação de arquiteturas de referência (KANG *et al.*, 1990; GRISS *et al.*, 1998; BRAGA, 2000). O processo de criação de uma arquitetura de referência requer grande esforço do engenheiro de domínio devido à sua característica de generalização, devendo este recorrer a várias fontes de informação, tal como especialistas no domínio, usuários, livros, documentação do domínio e, principalmente, as aplicações existentes no domínio (i.e., sistemas legados[1]) (BRAGA, 2000). Segundo KANG et al. (1990), sistemas legados representam uma fonte de conhecimento essencial para a engenharia de domínio.

Quando as arquiteturas dos sistemas legados compartilham características comuns, elas podem ser abstraídas para uma arquitetura de referência (BRAGA, 2000), que será composta de *elementos arquiteturais*. Os elementos arquiteturais podem ser definidos como "abstrações responsáveis por representar as entidades que implementam as funcionalidades especificadas (ex. módulo ou componente de software)" (DIAS & VIEIRA, 2000)

A análise e comparação das arquiteturas de sistemas legados ajudam na tarefa de identificação dos principais conceitos e elementos do domínio. Pode-se dizer que os elementos arquiteturais que aparecem em todas as aplicações são considerados "mandatórios" no domínio. Por outro lado, os elementos arquiteturais que aparecem

[1] 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).

somente em algumas aplicações podem ser tidos como “opcionais”. Existem ainda determinados elementos arquiteturais que refletem a parametrização da arquitetura em relação à ocorrência, ou não, de determinados componentes internos em sua configuração. Estes são chamados de “pontos de variação”. A caracterização de elementos como "mandatórios", "opcionais” ou "pontos de variação" reflete as variabilidades do domínio em questão (KANG *et al.*, 1990).

Mesmo sabendo que as aplicações de um mesmo domínio apresentam elementos comuns, a identificação destas similaridades e diferenças não é trivial. Para ilustrar estas dificuldades, tomemos como exemplo o domínio escolar. Podemos encontrar três diferentes aplicações que apresentam o mesmo elemento semântico, ex: “Aluno”, com diferentes nomenclaturas, a saber: “Aluno”, “Estudante” e “Matrícula”. Precisaríamos, então, de um glossário de termos e relações entre termos do domínio para identificar que “Aluno”, “Estudante” e “Matrícula” são sinônimos. Além do problema de nomenclatura, existe ainda o problema de organização e composição dos elementos arquiteturais nas diferentes arquiteturas. O mesmo elemento pode estar em diferentes locais nas arquiteturas e apresentar diferentes conexões. Na medida em que a complexidade e o número de aplicações comparadas crescem, maiores são as dificuldades para se obter estas informações de forma precisa.

Trabalhos existentes na literatura, tais como os que apresentam processos de engenharia de domínio, algoritmos e ferramentas que identificam diferenças entre elementos descritos em arquivos XML (*eXtensible Markup Language*) (XML, 2007), conhecidos como algoritmos de "*Diff*", modelos UML (*Unified Modeling Language*) (OMG, 2007) e arquiteturas descritas em ADL (*Architecture Description Language*) foram inicialmente analisados. Os algoritmos de “*Diff*” para XML (KEIENBURG & RAUSCH, 2001; JIANG & SYSTÄ, 2003) tendem a fazer uma comparação puramente sintática de elementos (*tags*) XML, gerando resultados sem nenhuma semântica para a criação de arquiteturas de referência. Outra limitação encontrada é que estes estão voltados para detectar as diferenças entre versões de modelos de uma mesma aplicação. As ferramentas analisadas que comparam modelos UML (KELTER *et al.*, 2005; FUJABA, 2007) e as que comparam arquiteturas descritas em ADL, como por exemplo a xADL (DASHOFY *et al.*, 2002) também apresentam o mesmo problema descrito anteriormente, pois comparam versões diferentes de um mesmo modelo e/ou arquitetura, ao invés de comparar modelos e arquiteturas que representam diferentes

aplicações. Dessa forma, há uma tendência em se apontar como diferentes, elementos arquiteturais que são semanticamente equivalentes.

1.2 - Objetivos

Tendo em vista os problemas citados anteriormente, essa dissertação tem como objetivo propor uma abordagem baseada na comparação de arquiteturas existentes em um domínio, tais como as recuperadas a partir dos sistemas legados, além de um ferramental automatizado para tal abordagem, visando apoiar o engenheiro de domínio na criação de uma arquitetura de referência (DSSA).

A abordagem proposta neste trabalho, denominada *ArchToDSSA*, visa à identificação de elementos arquiteturais "mandatórios", "opcionais" e "pontos de variação" no domínio, procurando oferecer soluções aos problemas mencionados referentes à comparação de arquiteturas. O apoio automatizado é obtido através da ferramenta *ArchToDSSATool*, implementada no contexto do ambiente de desenvolvimento e reutilização Odyssey (ODYSSEY, 2007).

A abordagem proposta utiliza como fonte de informação modelos arquiteturais de sistemas existentes no domínio, sendo estes legados ou não. A categorização de sistemas legados por domínio e a reconstrução de seus modelos arquiteturais, caso estes não estejam disponíveis, não são objetivos dessa dissertação em particular. No entanto, este trabalho está inserido em uma proposta mais ampla, denominada *LegaToDSSA*. Tal abordagem trata da recuperação de arquiteturas de sistemas legados, através de engenharia reversa estática e dinâmica, com o objetivo de criar arquiteturas de referência em um domínio específico. Maiores detalhes sobre esta abordagem completa podem ser encontrados em (VASCONCELOS, 2007).

O presente trabalho assume como pré-requisito a existência dos modelos arquiteturais dos sistemas do domínio, os quais devem ser orientados a objetos. Assume-se que as arquiteturas estejam descritas através de diagramas de pacotes e de classes da UML, linguagem esta escolhida pela sua grande aceitação e utilização na comunidade de engenharia de software. Além disso, tais arquiteturas podem ser representadas por arquivos XMI (*XML Metadata Interchange*), uma especificação adotada como mecanismo para possibilitar o compartilhamento de modelos UML criados em diversas ferramentas CASE (OMG, 2002b). XMI é o formato padrão de arquivo usado na importação/exportação de modelos baseados em MOF (*Meta-Object Facility*) (OMG, 2002a). Dessa forma, visando a possibilidade de tornar a ferramenta de

apoio genérica em relação a ambientes de desenvolvimento, adotou-se o XMI como forma de representação, ficando a ferramenta vinculada apenas à versão do XMI utilizado. Com base nos arquivos XMI, as arquiteturas são então comparadas e as informações obtidas servem como base para a criação da arquitetura de referência.

1.3 - Organização da Dissertação

Esta dissertação está organizada em seis capítulos. Neste **primeiro** capítulo, de introdução, são apresentados o referencial teórico básico, bem como a motivação, os objetivos e a organização do trabalho.

No **segundo** capítulo, são apresentados os conceitos sobre Arquitetura de Software, sua representação e métodos de comparação entre seus elementos.

No **terceiro** capítulo, são descritos métodos de Engenharia de Domínio e Linha de Produtos de Software, bem como suas abordagens para a especificação de uma arquitetura de referência no domínio.

A abordagem *ArchToDSSA*, proposta deste trabalho, é apresentada no **quarto** capítulo.

No **quinto** capítulo, é detalhada toda a implementação da proposta, no contexto do ambiente de reutilização Odyssey (ODYSSEY, 2007).

A fim de caracterizar a eficiência da abordagem e sua implementação, foram realizados dois estudos de observação, apresentados no **sexto** capítulo.

No **sétimo** e último capítulo, as contribuições e limitações deste trabalho são listadas, bem como apontados os trabalhos futuros.

Capítulo 2: Arquiteturas de Software

A complexidade e o tamanho de sistemas de software vêm crescendo de forma acentuada nos últimos tempos. A popularização da Internet, a necessidade de competir em um mercado cada vez mais globalizado e a disponibilidade de hardware mais potente com preços mais acessíveis estão dentre os fatores que impulsionam cada vez mais a demanda por software com maior número de recursos. Hoje em dia, é comum que sistemas, até mesmo de pequeno porte, sejam capazes de acessar informações e/ou serviços disponíveis de forma distribuída em diferentes pontos da rede mundial. Como exemplo, qualquer aplicação que precise disponibilizar a lista de CEPs e endereços do Brasil para seus usuários não precisa ter em seu projeto a implementação desta característica, pois pode facilmente acessar tais serviços através dos servidores dos correios, disponíveis na Internet.

Para facilitar o entendimento e construção de aplicações complexas, pode-se dividi-las em partes. Além disso, tecnologias disponíveis, dentre elas CORBA (CORBA, 2007), já permitem a divisão de uma aplicação desta forma. Entretanto, definir exatamente quais serão as partes que deverão compor cada divisão de um dado sistema, como elas deverão interagir entre si, que funcionalidades deverão fornecer e quais serviços deverão usar, não é uma tarefa trivial.

Para tentar responder às questões que surgem durante este processo de composição destes sistemas complexos e distribuídos, uma nova disciplina vem ganhando cada vez mais importância: a arquitetura de software.

O objetivo deste capítulo é discutir alguns aspectos sobre arquitetura de software, considerados relevantes para o entendimento deste trabalho, mais especificamente a comparação entre arquiteturas de software. Para tal, o capítulo está organizado da seguinte forma: na seção 2.1, é definido o termo arquitetura de software e o seu papel atual na construção de sistemas; na seção 2.2, são apresentadas as visões arquiteturais e seus modelos; na seção 2.3, são mostradas formas de se descrever arquiteturas através de notações formais, como ADLs; na seção 2.4, são discutidas as evoluções e adaptações pelas quais uma arquitetura passa ao longo da vida útil do software e mecanismos que podem apoiar o acompanhamento dessas mudanças; finalmente, na seção 2.5, são feitas as considerações finais do capítulo.

2.1 - Definição

A especificação da arquitetura representa os primeiros passos para o projeto de qualquer sistema e produz entradas para a maioria dos passos subseqüentes.

Segundo a literatura, existem várias definições para arquitetura de software, dentre elas podemos destacar:

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

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

"A arquitetura de um sistema consiste da(s) estrutura(s) de suas partes (incluindo as partes de software e hardware envolvidas no tempo de execução, projeto e teste), da natureza e das propriedades relevantes que são externamente visíveis destas partes (módulos com interfaces, unidades de hardware, objetos), e dos relacionamentos e restrições entre elas" (D'SOUZA & WILLS, 1999).

Conforme descrito na seção anterior, os sistemas têm se tornado cada vez mais complexos. A representação destes sistemas como arquiteturas de software permite a abstração dos detalhes internos das partes que os compõem, permitindo um melhor entendimento de como se dá a interação entre elas em um nível mais abstrato. Em (WALLNAU *et al.*, 2002), os autores ressaltam ainda que todos os aspectos estruturais de um sistema são definidos por sua arquitetura, incluindo como o software é dividido em componentes, quais funcionalidades são mapeadas para estes componentes e como os componentes interagem entre si.

A montagem da arquitetura deve permanecer independente de qualquer implementação ou tecnologia, utilizando apenas especificações de componentes (ou, mais genericamente, de elementos arquiteturais) para representar os serviços existentes na arquitetura (D'SOUZA & WILLS, 1999) (MENDES, 2002). A estrutura da arquitetura define como "montaremos" ou "encaixaremos" as peças (componentes) do sistema.

Muito embora possam existir diferenças na forma como a arquitetura de software é definida, existe um consenso de que a ênfase está na estrutura e relação entre os elementos de um sistema, ao invés de questões implementacionais como a estrutura de dados e algoritmos.

Em uma arquitetura, os componentes representam as unidades computacionais do sistema, e suas funcionalidades são acessadas através de suas interfaces, que definem um conjunto de assinaturas, isto é, uma descrição sintática de cada método, seus parâmetros, tipo, valores de retorno e possíveis exceções, que indicam como estes serviços poderão ser invocados. As interfaces, desta forma, definem a maneira como os componentes poderão interagir com outros elementos arquiteturais. Os conectores definem as interações entre os elementos da arquitetura.

As possíveis configurações de uma arquitetura são criadas através das diferentes combinações de componentes, interfaces e conectores.

2.2 - Visões Arquiteturais

Conforme pôde ser observado, existe uma grande ênfase na descrição de arquiteturas, destacando-se principalmente características estruturais como: componentes, suas interconexões (conectores), configurações e funcionalidades de cada componente. Muito embora a estrutura receba sempre maior destaque, existem outros aspectos que devem ser considerados durante a tarefa de se obter uma descrição arquitetural completa e compreensível. Para o completo entendimento da arquitetura de um software, é necessário considerar não apenas a perspectiva estrutural, mas também aspectos como distribuição física, processo de comunicação e sincronização, entre outros (CLEMENTS, 1994). Cada um destes aspectos deve ser tratado em uma diferente visão arquitetural.

Em (PENEDO & RIDDLE, 1993), os autores afirmam que, uma arquitetura de software deve ser vista e descrita sob diferentes perspectivas (ou visões) e deve identificar seus componentes, relacionamento estático, interações dinâmicas, propriedades, características e restrições.

Uma visão consiste em um conjunto de modelos focados em um determinado aspecto da arquitetura, omitindo outros detalhes que são menos importantes neste contexto e serve como insumo para cada passo do processo de construção do software.

Alguns autores definiram diferentes modelos de visões. O modelo de KRUCHTEN (1995), conhecido como modelo de visões arquiteturais 4+1 (*The 4+1 View Model*), organiza o modelo de visões em cinco visões concorrentes. Cada visão

mostra um conjunto específico de informações de acordo com o interesse dos diferentes *stakeholders*[2].

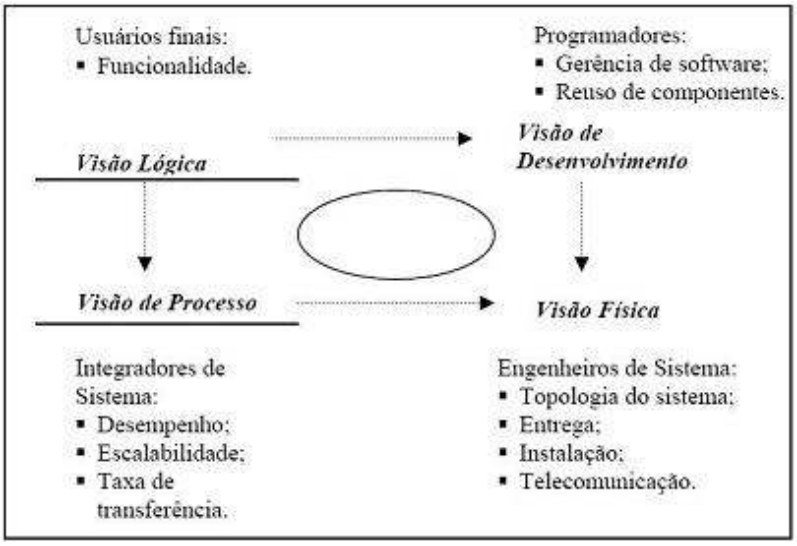

Figura 2.1 - Visões do modelo 4+1
(Adaptado de (KRUCHTEN, 1995))

As cinco visões definidas por KRUCHTEN (1995) são: **visão lógica** (descreve aspectos estáticos: componentes, conectores, interfaces), **visão de processo** (descreve aspectos de processo, incluindo concorrência e sincronização), **visão física** (descreve o mapeamento dos componentes e processos para o hardware e sua distribuição), **visão de desenvolvimento** (descreve a organização estática do software no ambiente de implementação) e **visão de cenários** (descreve os casos de uso do sistema em sua arquitetura). As decisões de projeto são capturadas em quatro visões, usando a quinta visão com objetivo de ilustrar e validar as demais. Tais visões são ilustradas na Figura 2.1

HOFMEISTER *et al.* (1999), por sua vez, dividem a descrição da arquitetura em quatro visões: **visão conceitual** (descreve componentes e conectores, mapeando características funcionais para a arquitetura), **visão de módulo** (decompõe o software em subsistemas - módulos), **visão de execução** (descreve os processos mapeados a partir dos módulos, sua distribuição e taxa de uso), **visão de código** (descreve como os módulos da visão de módulos são mapeados para arquivos de código fonte e como os processos da visão de processos são mapeados para arquivos executáveis).

[2] Pessoas envolvidas ou interessadas em diferentes aspectos do software de acordo com seu papel exercido, exemplo: programadores, engenheiros de domínio, etc.

Conforme observado, tanto KRUCHTEN (1995), quanto HOFMEISTER *et al*. (1999), prevêem a separação dos aspectos estáticos e dinâmicos do sistema em diferentes visões arquiteturais. Enquanto as perspectivas estáticas descrevem os componentes e suas inter-relações, perspectivas dinâmicas tratam de aspectos como comportamento do sistema em sua arquitetura, sincronização e concorrência. A ênfase deste trabalho são os aspectos estáticos, i.e., a Visão Lógica proposta por KRUCHTEN.

2.3 - Representação Arquitetural

Para representar as diferentes perspectivas das arquiteturas, tanto no tocante às suas visões estáticas, quanto às suas visões dinâmicas, diagramas que fazem uso de diversos símbolos gráficos costumam ser utilizados. De uma forma geral, a notação é informal, onde comumente caixas descrevem componentes do sistema sendo representado, e linhas indicam a ocorrência de alguma forma de comunicação, controle ou relacionamento entre estes componentes. Descrições informais como estas, são ambíguas e levam o leitor a interpretações pessoais e conflitantes (PERRY & WOLF, 1992).

O uso de formalização para representar arquiteturas traz várias vantagens para o processo de desenvolvimento de software. Algumas destas vantagens estão descritas em (PERRY & WOLF, 1992), (ABOWD *et al.*, 1993),(CLEMENTS, 1996):

- **Compreensão**: Como a arquitetura de software permite o entendimento de problemas complexos de uma forma mais abstrata, a diminuição das ambigüidades leva à redução da distância semântica existente entre as diferentes fases do processo, como por exemplo, requisitos levantados e projeto concebido.
- **Reutilização**: Além do fato de que os conceitos representados pelas arquiteturas podem ser mais facilmente compreendidos por diferentes *stakeholders*, arquiteturas podem servir de base de conhecimento reutilizável acerca de problemas e/ou requisitos específicos.
- **Evolução**: Permite um melhor entendimento das dimensões em que uma aplicação pode ser expandida, sem que ocorram problemas de violação e instabilidade de sua arquitetura.

A necessidade de se estabelecer essa formalidade na representação das arquiteturas fomentou a pesquisa nessa área e culminou com a criação de linguagens

especializadas que permitem suas descrições sem as ambigüidades e os equívocos produzidos pela descrição informal.

Dentre as abordagens propostas para representar arquiteturas, destacamos duas de interesse para os estudos deste trabalho: as linguagens de descrição da arquitetura (ADLs – *Architecture Description Languages*), como a xADL (DASHOFY *et al.*, 2001), e a utilização da UML (*Unified Modeling Language*). As subseções a seguir detalham cada uma dessas abordagens.

2.3.1 - Representando Arquiteturas através de ADLs

Alternativamente aos diagramas informais descritos anteriormente, uma variedade de linguagens de descrição de arquiteturas (ADLs) capazes de representá-las de uma maneira formal, vêm sendo propostas. Essas linguagens têm por objetivo prover os arquitetos de software de notações para a especificação e análise das arquiteturas (MONROE *et al.*, 1997).

Embora as ADLs pareçam vir a resolver os problemas relacionados à representação arquitetural, seu uso não vem sendo difundido de forma expressiva no projeto de software. Este problema se deve a uma série de fatores, dentre eles: a complexidade das ADLs, a falta da compreensão da comunidade de engenharia de software acerca das abstrações arquiteturais, a indefinição do que de fato representa uma ADL e a não existência de uma linguagem padrão para a descrição de arquitetura de software (VASCONCELOS, 2004).

Os principais elementos descritos em uma ADL são os componentes e conectores. Segundo (SHAW & GARLAN, 1996), esses elementos devem ser tratados de forma independente e considerados como sendo de primeira grandeza na descrição da arquitetura. Eles definem ainda seis áreas que descrevem as principais características de uma ADL:

1. **Composição**: deve permitir que uma arquitetura possa ser descrita através da composição de componentes e conectores. Sistemas complexos devem poder ser divididos em partes menores para melhor entendimento ou devem poder ser construídos a partir da composição de elementos menores e mais gerenciáveis.
2. **Abstração**: deve ser possível descrever em um alto nível de abstração as propriedades dos componentes e conectores, identificando seus comportamentos

sem que, para isto, haja a necessidade de se saber como estes elementos serão implementados.

3. **Configuração**: deve permitir a descrição da estrutura do sistema (sua topologia), independentemente dos elementos sendo estruturados. A configuração define como será feita a correspondência entre os componentes e os papéis definidos pelos conectores. A conexão entre os componentes só pode ser estabelecida quando o protocolo de comunicação e os papéis estabelecidos pelos conectores são compatíveis com as expectativas dos componentes.
4. **Reusabilidade**: deve ser possível a reutilização de componentes, conectores e padrões de relacionamento componente-conector (ex: configuração de topologia), em outras descrições arquiteturais diferentes daquela para a qual os elementos foram originalmente especificados.
5. **Heterogeneidade**: deve permitir a combinação de diferentes configurações arquiteturais em uma mesma descrição arquitetural e ainda deve permitir que seja possível a combinação de componentes descritos em diferentes linguagens de programação.
6. **Análise**: deve ser possível a realização de diversas análises da arquitetura com base em suas descrições, como por exemplo, simulação da arquitetura para análise de seus atributos de qualidade ou análise da evolução de diferentes versões de uma mesma arquitetura.

Diferentes ADLs foram identificadas em (SHAW & GARLAN, 1996) e (MEDVIDOVIC & TAYLOR, 2000), dentre elas se encontram: Unicom, Aesop, Wright e Rapide. Recentemente uma nova ADL, que foi criada na universidade da Califórnia (Irvine), para a modelagem da arquitetura de sistemas de software, vem merecendo atenção: a xADL (DASHOFY et al., 2001). Esta ADL difere das demais por ser flexível e extensível suportando o uso de ferramentas comerciais para manipulação das arquiteturas (ex: visualização de arquiteturas dentro do Microsoft Visio, através de plugins específicos).

2.3.2 - Representando Arquiteturas através de UML

Muito embora as ADLs tenham sido criadas para descrever arquiteturas, vários trabalhos na literatura vêm fazendo o uso da UML (OMG, 2007) para este fim. A UML

representa a notação padrão para sistemas orientados a objetos e seu uso em larga escala, tanto a nível acadêmico quanto industrial, é um dos fatores que impulsionaram seu uso na descrição arquitetural.

HOFMEISTER *et al*. (1999) mostram um exemplo de uso da UML como notação para a representação da arquitetura. Para cada uma das visões definidas em seu modelo de visões, foi associado um tipo de diagrama ou elementos da modelagem UML: na **visão conceitual**, os componentes e conectores são representados por classes estereotipadas e os papéis dos conectores são mapeados para os papéis das associações, as configurações estáticas do sistema são representadas pelos diagramas de classe, e os diagramas de seqüência demonstram as interações entre grupos de componentes; na **visão de módulo**, os módulos são representados como classes estereotipadas e subsistemas como pacotes estereotipados, a decomposição dos sistemas em módulos e suas dependências são representadas por diagramas de pacotes e classes; na **visão de execução**, classes estereotipadas representam a imagem *run-time* do sistema, a configuração estática do sistema e suas comunicações são representadas por diagramas de classes, e o comportamento dinâmico de uma configuração pelo diagrama de seqüência; na **visão de código**, executáveis, arquivos binários e de código fonte, são representadas por classes estereotipadas, e tabelas são utilizadas para descrever o mapeamento entre elementos das visões de módulo e de execução para os elementos da visão de código.

Assim como em (HOFMEISTER *et al.*, 1999), em (KRUCHTEN, 2000), a UML também é utilizada para representar os elementos das visões arquiteturais, propondo um modelo orientado pelo seguinte mapeamento: para a **visão lógica**, ele utiliza os diagramas de pacotes e de classes da UML, onde componentes são representados através de pacotes e de classes e os conectores através dos relacionamentos de associação, composição, dependência e generalização; a **visão de desenvolvimento** é representada através do elemento de modelagem componente e de pacotes da UML, onde cada componente da UML representa um módulo do software (arquivo fonte, binário, executável etc.), e os subsistemas, que definem um conjunto de componentes, são mapeados para pacotes; a **visão física** é representada pelo diagrama de implantação da UML, o qual é capaz de retratar os processadores e suas respectivas cargas de processo; a **visão de cenários** é representada pelos modelos de casos de uso.

2.4 - Evolução e Comparação de Arquiteturas

Na medida em que ocorrem mudanças no ambiente e que demandas por novas funcionalidades são geradas ao longo do ciclo de vida do software, revisões nos seus requisitos se fazem necessárias. Recorrentes alterações em requisitos, por sua vez, muitas vezes implicam em mudanças nas descrições arquiteturais que os representam.

Compreender o comportamento dessas mudanças intrínsecas ao processo de desenvolvimento de software pode ser muito útil para a manutenção e o entendimento da evolução dos sistemas. Como um exemplo, considere um caso em que o engenheiro A, responsável por um dado projeto, acompanha sua evolução e mantém sua descrição arquitetural atualizada. Por um dado motivo, o engenheiro A não pode mais trabalhar no projeto. Felizmente, é possível deslocar um outro engenheiro B, que trabalhava há alguns meses atrás, neste mesmo projeto, para a função. O engenheiro B necessita de uma forma rápida e precisa para entender todas as mudanças ocorridas na arquitetura do projeto ao longo do tempo em que esteve ausente, no intuito de dar continuidade aos trabalhos. Muito embora seja possível para o engenheiro B manualmente descobrir todas essas mudanças, esta tarefa acaba se tornando custosa, principalmente quando se trata de arquiteturas complexas, compostas por inúmeros elementos arquiteturais.

Automatizar a detecção de mudanças em arquiteturas pode, a princípio, sugerir a utilização de algoritmos disponíveis em larga escala na área de gerência de configuração[3]. Neste campo de pesquisa, algoritmos de *Diff* (comparação e detecção de diferenças) e *Merge* (fusão dos diferentes arquivos em um único arquivo consolidado), capazes de detectar e propagar mudanças em diferentes versões de arquivos, têm estado em uso ao longo de anos (BUFFENBARGER, 1995). Entretanto, suas aplicações diretas em arquivos que representam arquiteturas podem levar a resultados pouco satisfatórios (WESTHUIZEN & HOEK, 2002). Um dos motivos levantados por WESTHUIZEN e HOEK (2002) é o fato desses algoritmos operarem em arquivos texto, baseados numa comparação linha a linha. Desta forma, eles não estão a par da semântica específica referente a arquiteturas e, portanto, oferecem pouca ajuda no tocante à detecção de suas mudanças.

---

[3] Disciplina que aplica procedimentos técnicos e administrativos para identificar e documentar as características físicas e funcionais de um Item de Configuração, controlar as alterações nessas características, armazenar e relatar o processamento das modificações e o estágio da implementação e verificar a compatibilidade com os requisitos especificados. (IEEE, 1990)

A seguir, serão analisados algoritmos capazes de comparar arquiteturas descritas em diferentes formas: na seção 2.4.1, será apresentada a comparação de arquiteturas através da gerência de configuração; em 2.4.2 serão mostrados exemplos do funcionamento de algoritmos que detectam diferenças em diagramas UML (ferramenta *Fujaba Tool Suite* (FUJABA, 2007)); a seção 2.4.3 trata de algoritmos que usam o formato XML (ferramenta *XML Diff and Merge Tool* (IBM, 2007)) e, finalmente em 2.4.4, é apresentada "*xADL Diff Tool*" ferramenta de comparação de arquiteturas da linguagem de descrição de arquiteturas xADL (DASHOFY *et al.*, 2001) .

2.4.1 - Comparando Arquiteturas na Gerência de Configuração

Tradicionalmente, os algoritmos de *diff* e *merge* (BUFFENBARGER, 1995) existentes na área de Gerência de Configuração são baseados na comparação linha a linha de arquivos texto. Neste tipo de abordagem, mais tradicional, a linha é considerada a menor unidade de comparação. O algoritmo aceita dois arquivos como entrada e através da comparação linha a linha entre os dois arquivos, gera um arquivo de saída "*diff*" contendo uma lista ordenada das linhas que foram removidas, adicionadas ou substituídas. Este arquivo de saída, normalmente, é um arquivo texto e pode ser visualizado através de uma ferramenta específica.

A ferramenta capaz de visualizar as diferenças entre os dois arquivos comparados e fundi-las, criando um terceiro arquivo que incorpore todas as mudanças detectadas, é normalmente chamada de ferramenta de *merge*. Ela pode, por exemplo, receber como entrada os dois arquivos comparados mais o arquivo de *diff* resultante da comparação anterior e, através de interações com o usuário, gerar um arquivo de saída que representa a fusão dos dois arquivos originais. Os conflitos detectados durante o processo de *merge*, por exemplo, quando existirem alterações em uma mesma linha nos dois arquivos originais, são tidos como falhas ou são expostos para o usuário decidir e/ou alterar como deverá ficar a linha resultante no arquivo a ser gerado.

Estes algoritmos de Gerencia de Configuração tendem a ser independentes em relação às informações contidas nos arquivos sendo comparados e, portanto, não fazem análises na forma e significado de seus conteúdos. Muito embora eles sejam capazes de funcionar com arquivos texto que representem arquiteturas, o resultado não é uma descrição arquitetural e não são capazes de lidar bem com as substituições de elementos arquiteturais neste nível de abstração (WESTHUIZEN & HOEK, 2002).

2.4.2 - Comparando Arquiteturas em UML

A abordagem apresentada por KELTER *et al*, (2005) no ambiente FUJABA, propõe a detecção das diferenças entre arquiteturas representadas através de diagramas UML. A motivação usada para o desenvolvimento do algoritmo está baseada na filosofia de desenvolvimento de software proposta pela OMG (*Object Management Group*): o MDA (*Model Driven Architecture*).

Segundo KELTER *et al* (2005), o foco central do MDA é o modelo do sistema de software que passa a assumir um papel fundamental no processo de desenvolvimento. No MDA, modelos podem ser transformados assumindo diferentes níveis de abstração. Por exemplo: um modelo PIM (*Platform Independent Model*) pode ser transformado em um modelo PDM (*Platform Dependent Model*), para a geração de código fonte e posterior obtenção de um "modelo executável" (i.e., a aplicação). Seguindo esta filosofia, a representação de sistemas pode ser feita através de modelos UML, e as alterações no software podem ser realizadas neste nível mais alto de abstração, para posterior transformação em modelos menos abstratos, como explicado.

Em geral, o desenvolvimento de sistemas é realizado em times, e no caso de se usar uma abordagem MDA, será necessário suporte à detecção de *diffs* e *merge* de partes dos modelos UML modificados por diferentes membros destas equipes.

O ambiente FUJABA propõe não somente a comparação de arquiteturas, mas também a representação visual das diferenças detectadas entre os modelos através de diagramas UML. Nestes diagramas de diferenças, são mostrados todos os elementos resultantes da fusão dos modelos originais, sendo que cores são utilizadas para distinção entre estes elementos. Exemplo: partes similares são apresentadas em preto, os elementos pintados de vermelho representam elementos existentes apenas no primeiro modelo e os verdes existem apenas do segundo modelo. Os modelos suportados são representados através de arquivos XMI (*XML Metadata Interchange*), conforme definido na seção 1.2, que são interpretados como árvores para a computação das diferenças. O arquivo de saída também é no formato XMI, que contém informações dos dois modelos comparados além de informações adicionais, calculadas através do seu algoritmo de *Diff* .

O algoritmo de *Diff* de FUJABA foi dividido em duas fases. Na primeira fase, elementos similares são detectados entre os dois modelos comparados e, na segunda e última fase, as diferenças entre os dois modelos são calculadas e os arquivos de saída são gerados. Na escolha de elementos equivalentes, um dado elemento só pode ter

apenas um equivalente no outro diagrama. A abordagem permite que apenas dois modelos sejam comparados por vez e espera, preferencialmente, que os diagramas de entrada sejam provenientes da evolução de modelos em uma abordagem MDA, ou seja, oriundos de uma abordagem de desenvolvimento *top-down*, com um certo nível de similaridade.

Como observado, esta solução apresenta algumas limitações que impedem o seu uso para a comparação de sistemas diferentes de um mesmo domínio, visto que nestes tipos de sistemas, é natural a existência de elementos equivalentes que possuem sintaxe e disposição totalmente distintas dentro das arquiteturas comparadas.

2.4.3 Comparando Arquiteturas em XML

A ferramenta de *Diff* and *Merge Tool*, criada pela IBM (IBM, 2007), também permitem a detecção de diferenças entre arquivos em uma abordagem que não é baseada na comparação linha a linha. Elas operam em arquivos baseados em *tags* XML e recebem como entrada um arquivo base e uma versão modificada do mesmo arquivo e produzem um terceiro arquivo XML, representando a junção dos dois arquivos processados.

As diferenças são apresentadas através de uma ferramenta visual e o objetivo é fazer o usuário resolver todos os conflitos. Os nós identificados como modificados, inseridos ou removidos são apresentados ao usuário, que deve escolher qual deles deve prevalecer: se o nó procedente do arquivo base ou do arquivo modificado. A escolha feita pelo usuário em um nó é propagada para os nós descendentes.

A comparação é realizada através de uma identificação existente em cada nó, baseada em seus atributos do tipo ID ou em seu conteúdo. Não é permitido que o usuário adapte ou modifique o algoritmo comparação. Esta característica apresenta sérias limitações para a comparação de arquiteturas de software legado, visto que os IDs dos elementos de uma arquitetura dificilmente são iguais aos IDs dos elementos de uma outra arquitetura completamente diferente, o que inviabiliza uma tentativa de usar este algoritmo para este tipo de comparação.

2.4.4 Comparando Arquiteturas em xADL

As detecções de mudanças em representações arquiteturais através da xADL foram implementadas através de dois algoritmos (WESTHUIZEN & HOEK, 2002). O

primeiro descobre as diferenças entre duas arquiteturas em termos de adições e remoções de elementos, e o segundo algoritmo complementa o primeiro, calculando quais adições e remoções representam substituições, a fim de aumentar o grau de entendimento destas mudanças.

O algoritmo de *Diff* recebe como entrada duas arquiteturas no formato xADL e gera um arquivo também no formato xADL (XML seguindo os esquemas da xADL). Este arquivo de saída contém elementos que representam as adições e remoções como descrito a seguir.

Primeiramente, as inserções são calculadas da seguinte forma: cada elemento (componentes, conector...) da primeira arquitetura tem seu identificador único comparado com cada elemento da segunda arquitetura. Caso o elemento na primeira arquitetura não seja encontrado na segunda, o mesmo é adicionado ao arquivo *diff* de saída como uma inserção.

Em seguida, é necessário identificar os elementos que estão presentes somente na segunda arquitetura. Para tanto, verifica-se para cada elemento da segunda arquitetura se o mesmo existe também na primeira arquitetura. Caso não exista, o mesmo é adicionado como uma remoção no arquivo *diff* . Caso exista, nada precisa ser feito.

O arquivo *diff* resultante, gerado ao final deste processo, pode ser visualizado através de qualquer ferramenta capaz de visualizar XMLs. A visualização deste arquivo permite identificar as diferenças entre as duas arquiteturas comparadas, em termos de inserções e remoções.

Segundo WESTHUIZEN et. al (2002), simplesmente representar as diferenças entre duas arquiteturas em termos de inserções e remoções muitas vezes não é suficiente para entender mudanças arquiteturais. Existem alterações, por exemplo, que estão relacionadas com a substituição de um conjunto de componentes por outros componentes que devem ser entendidas. Estas substituições representam um conceito de mais alto nível que uma lista de inserções e remoções e ajudam na obtenção de uma maior compreensão das evoluções arquiteturais.

O algoritmo de detecção de substituições, implementado na xADL, recebe como entrada uma arquitetura e um arquivo *diff* e calcula os elementos e conjuntos de elementos substituídos. Ele opera, procurando por diferentes conjuntos de elementos que são “cercados” pelos mesmos elementos comuns. Baseado no fato de que, na substituição de um componente ou conector, um link teve que ser rompido e substituído

por um novo link, o algoritmo usa links comuns entre elementos como potenciais pontos de partida para detectar substituições. Estes pontos de partida são usados para tentar identificar possíveis substituições simples de elementos (um único elemento) e também percorre as vizinhanças destes elementos identificando outros links comuns, a fim de determinar os limites (vizinhança), da possível substituição sendo analisada. Desta forma, o algoritmo detecta não somente substituições simples de elementos isolados, mas também substituições de um conjunto de elementos de uma só vez. A Figura 2.2 mostra a remoção em inserção de elementos arquiteturais.

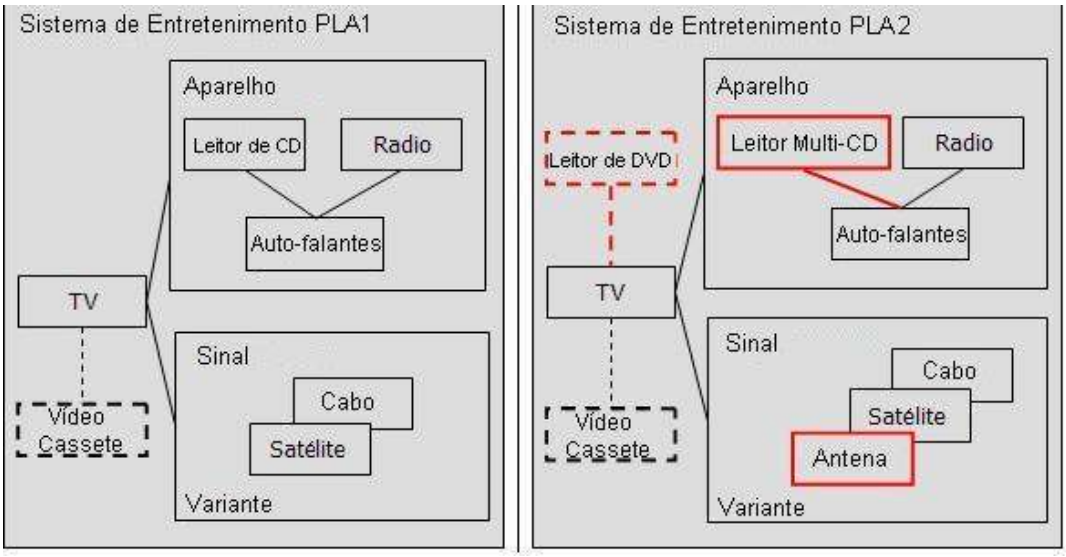

Figura 2.2 - Comparação de arquiteturas em xADL.

(Adaptado da documentação de *Pladiff* (DASHOFY *et al.*, 2002))

A arquitetura da esquerda é considerada a arquitetura original, ao passo que a da direita é considerada sua evolução. As mudanças na nova arquitetura são marcadas em vermelho.

Para complementar o entendimento das mudanças arquiteturais na xADL, o algoritmo de *merge* recebe como entrada uma arquitetura e o arquivo *diff* e gera uma arquitetura resultante da fusão dos dois arquivos de entrada. Este processo é feito através da iteração de cada elemento do arquivo *diff*. Elementos marcados como inserções são inseridos na arquitetura original, elementos marcados como remoções são removidos.

Assim como no ambiente Fujaba, a comparação é sempre realizada entre duas arquiteturas. Espera-se que as arquiteturas sendo comparadas possuam um mínimo de

elementos comuns para que bons resultados sejam alcançados. Para que o *merge*, em particular, possa ser executado entre um arquivo de *diff* e uma arquitetura, é necessário que existam elementos comuns entre esta arquitetura e as outras duas arquiteturas que originaram o arquivo de *diff* usado (WESTHUIZEN, et. al, 2002).

Muito embora, os algoritmos de comparação da xADL possuam as funcionalidades apresentadas, para que possam ser usados na comparação de arquiteturas de sistemas legados, seria necessária que a comparação pudesse contemplar as diferenças arquiteturais existentes nestes tipos de sistemas. A implementação de um dicionário de sinônimos, a ser usado na comparação, a possibilidade de se comparar mais de uma arquitetura ao mesmo tempo, estariam dentre as adaptações necessárias para tratar estas diferenças. Neste sentido, para a comparação eficaz de arquiteturas de sistemas legados, faz-se necessária a criação de uma abordagem que atendam as estes requisitos.

2.5 - Considerações Finais

Como pôde ser observado, a representação de sistemas através de arquiteturas de software permite a abstração dos detalhes internos das partes que os compõem, e fornece um melhor entendimento de como se dá a interação entre elas em um nível mais abstrato.

As descrições dos elementos arquiteturais devem ser feitas através de notações que imponham algum tipo de formalismo. Descrições informais são ambíguas e levam o leitor a interpretações pessoais e conflitantes (PERRY & WOLF, 1992). Neste sentido, abordagens mais formais para descrever arquiteturas vêm sendo criadas e usadas como, por exemplo: as ADLs e a UML.

Além da necessidade de se descrever arquiteturas de uma maneira mais formal, é importante perceber que as arquiteturas de software representam requisitos em constante mutação. Soma-se, ainda, o fato de que o número de elementos arquiteturais e suas relações podem atingir determinada escala em função da complexidade do sistema e de suas evoluções. Desta forma, é necessário que existam mecanismos capazes de ajudar o engenheiro na identificação e entendimento destas mudanças.

Conforme mostrado, algoritmos foram desenvolvidos, nos contextos das diferentes notações arquiteturais, com o intuito de descobrir e propagar estas mudanças. Os algoritmos de *diff*, de uma forma geral, foram implementados com o objetivo de descobrir as diferenças existentes entre duas versões de uma mesma arquitetura. E,

muito embora alguns destes algoritmos sejam capazes de perceber a substituição de um conjunto de elementos de uma versão arquitetural para outra, a forma de determinar se dois elementos são equivalentes é, na maioria das vezes, feita através da comparação da similaridade entre os identificadores únicos que representam cada componente e/ou seus descendentes.

Estas abordagens podem levar a resultados pouco eficazes, no contexto em que os nomes dos elementos arquiteturais (e/ou seus identificadores) apresentem diferenças significativas nas duas arquiteturas sendo comparadas. É importante ressaltar que dependendo da origem das arquiteturas comparadas, ou da forma como foram evoluídas, pode existir um número grande de elementos arquiteturais com nomenclaturas diferentes que possuam o mesmo papel semântico (elementos que representam o mesmo conceito ou característica, sendo equivalentes, porém possuindo sintaxes completamente distintas). Para resolver este problema, este trabalho de mestrado propõe uma abordagem que leva em consideração estas diferenças, tornando possível não somente a comparação destes tipos de arquiteturas, mas a identificação de elementos que deverão compor uma arquitetura de referência do domínio.

Capítulo 3: Engenharia de Domínio, Linha de Produtos e suas Abordagens para Apoiar a Especificação de Arquiteturas de Referência

O conceito de reutilização é conhecido e empregado há muito tempo pela humanidade em diversas áreas. O processo de criação de uma nova invenção ou tecnologia envolve muitas vezes a reutilização e composição de diferentes partes previamente criadas. O automóvel, por exemplo, é criado através da composição de partes existentes como rodas, motor, câmbio etc. Analogamente, na Engenharia de Software, a reutilização é uma abordagem importante na criação de sistemas com maior qualidade e despendendo um menor esforço de desenvolvimento. Pode-se definir a reutilização de software como sendo o processo de criação de sistemas de software a partir de software existente, ao invés de criá-lo do zero (SAMETINGER, 1997).

Em (SAMETINGER, 1997), encontramos alguns benefícios trazidos pelo emprego da reutilização no processo de criação de software:

- **Qualidade**: o uso e reutilização tende a eliminar erros em componentes[4] de uma forma muito mais rápida do que em componentes usados somente uma vez;
- **Produtividade**: menos código é necessário ser implementado e ainda a curva de aprendizado é suavizada na medida em que os mesmos componentes vão sendo reutilizados pelas equipes;
- **Desempenho**: os custos com otimizações passam a ser viáveis, pois estarão sendo compartilhados com todos os projetos onde os componentes são reutilizados;
- **Robustez**: na medida em que componentes são reutilizados várias vezes, livres de bugs, maior é a probabilidade de construção de produtos mais robustos;
- **Interoperabilidade**: através do uso de componentes com interfaces bem definidas, testadas e conhecidas, pode-se construir mais facilmente

[4] Um componente pode ser definido como "uma unidade de software independente, que encapsula, dentro de si, seu projeto e implementação, e oferece serviços, por meio de interfaces bem definidas para o meio externo" (BARROCA *et al.*, 2005).

sistemas que se comunicam de forma mais eficaz entre si;

- **Redução de tempo de desenvolvimento e "time-to-market"**: empregar a reutilização de artefatos para construção do software reduz o tempo de criação de produtos e entrada no mercado;
- **Redução da Documentação**: redução do volume de documentação a ser gerada através da reutilização da documentação dos componentes reutilizados;
- **Redução dos custos de manutenção**: o número de partes do software que precisam ser mantidas diminui, pois o uso de componentes reutilizados e re-testados tendem a não apresentar defeitos. Além disso, ainda que necessária, a manutenção de componentes reutilizáveis pode ter seus custos distribuídos entre os projetos que o usam;
- **Redução dos tamanhos das equipes**: existem muitos problemas de comunicação e produtividade associados com o crescimento das equipes de desenvolvimento; sabe-se que não é verdade que ao se dobrar o tamanho da equipe, dobra-se a produtividade. Desta forma, manter as equipes menores através da reutilização de código pode atenuar estas falhas;

De acordo com WERNER E BRAGA (2005), para que a reutilização de software seja efetiva, esta deve ser sistematicamente considerada em todas as fases do desenvolvimento, desde a análise até a implementação.

Abordagens como engenharia de domínio e linha de produtos de software visam esta sistematização e fazem uso de arquiteturas de software como base para a reutilização de artefatos que, de uma forma geral, devem estar compatíveis com a estrutura estabelecida para o sistema.

Neste contexto, o objetivo deste capítulo é apresentar conceitos de engenharia de domínio (ED) e linha de produtos de software (LPS), mostrando como estas abordagens podem ajudar na criação de arquiteturas reutilizáveis em um domínio. Para tal, o mesmo está organizado da seguinte forma: na seção 3.1, é definido o que é um domínio, como ele pode ser organizado através da engenharia de domínio, quais fases devem ser contempladas para esta engenharia; na seção 3.2, são apresentados os conceitos sobre linha de produtos de software e suas principais atividades; na seção 3.3 e 3.4, são descritas, respectivamente, abordagens de engenharia de domínio e linha de produtos de

software, e seu apoio à especificação de arquiteturas de referência. Considerações Finais são apresentadas na seção 3.5.

3.1 - Engenharia de Domínio

Criar componentes de software que possam ser utilizados em outras aplicações, além daquelas para as quais eles foram originalmente definidos, é um dos principais problemas na reutilização de software (PRIETO-DIAZ & ARANGO, 1991) (BRAGA, 2000). Segundo PRIETO-DIAZ *et al* (1991), é fundamental para a criação destes componentes reutilizáveis, que exista, a priori, um processo de coleta de informações sistemático e confiável. O autor define este processo como sendo a **análise de domínio**, cujo objetivo é identificar e organizar o conhecimento a respeito de uma classe de problemas, de maneira a apoiar a descrição e solução de tais problemas.

O termo domínio é utilizado para referenciar ou agrupar uma coleção de aplicações, existentes ou futuras, que compartilhem características comuns, i.e., uma classe de sistemas que apresentam funcionalidades similares (WERNER & BRAGA, 2005; SEI/CMU, 2007).

As informações coletadas na análise de domínio podem ser obtidas através de diferentes fontes de conhecimento como especialistas no domínio, livros, usuários no domínio e sistemas legados, e servem como subsídio para a criação de um modelo genérico, capaz de descrever as características, similaridades e diferenças dos sistemas analisados, o modelo de domínio. Estas diferenças e similaridades são conhecidas como variabilidade do domínio (OLIVEIRA, 2006).

Refinamentos no modelo de domínio permitem a criação de componentes reutilizáveis, que podem fazer parte de uma infra-estrutura de reutilização. Todas estas atividades fazem parte de um processo denominado **engenharia de domínio** (WERNER & BRAGA, 2005) (ARANGO, 1994)

Os artefatos reutilizáveis gerados na ED formam a base para que a reutilização ocorra. Sistemas específicos podem ser criados a partir destes artefatos e instanciados de acordo com os requisitos da aplicação sendo construída. Este processo é denominado engenharia de aplicação (GRISS *et al.*, 1998) (MILER, 2000). A engenharia de domínio e a engenharia de aplicação são processos complementares. Enquanto a engenharia de domínio tem por objetivo prover artefatos que serão reutilizáveis, a engenharia de aplicação tem por objetivo a construção de aplicações fazendo uso destes artefatos. Desta forma, a engenharia de domínio representa o desenvolvimento **para** reutilização

enquanto a engenharia de aplicação representa o desenvolvimento **com** reutilização. A Figura 3.3 ilustra este ciclo complementar.

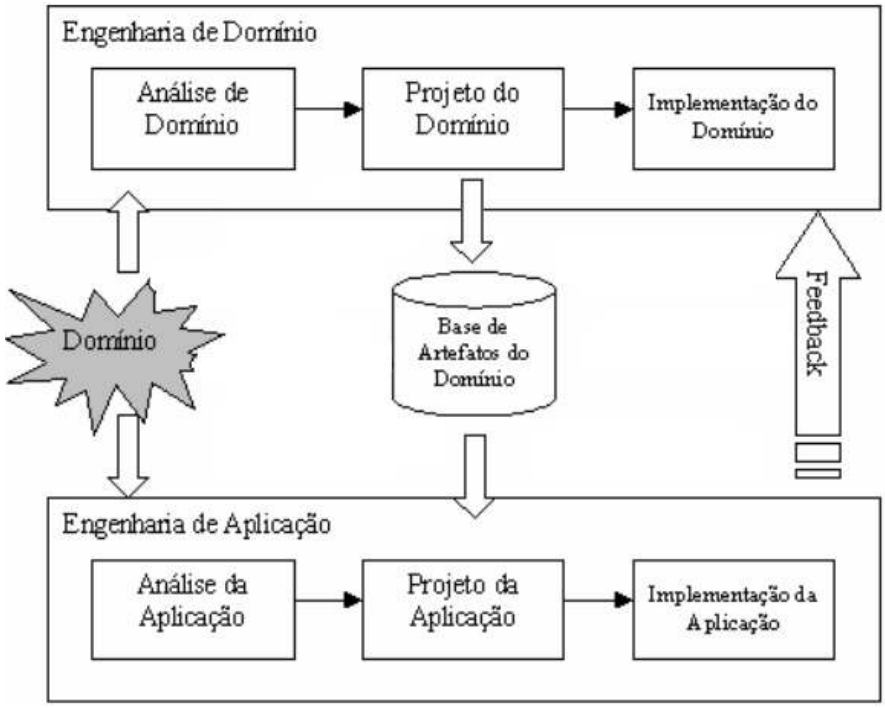

Figura 3.3- Ciclo de Vida da Engenharia de Domínio

(Adaptado de (ATKINSON *et al.*, 2002))

Em (BLOIS, 2006), é mencionada uma seqüência de atividades que devem ser executadas na modelagem de um domínio. Tais atividades visam à construção de artefatos que serão reutilizados por aplicações derivadas do domínio:

1º) a identificação de um domínio e seu escopo, bem como a captura de requisitos do domínio em que são representadas suas variabilidades;

2º) a construção de um projeto adaptável do domínio, através da especificação de uma arquitetura que represente os requisitos funcionais e não-funcionais do domínio, devidamente estruturados;

3º) a definição de mecanismos para a tradução dos requisitos do domínio em sistemas criados a partir de artefatos reutilizáveis.

De uma forma geral, existe um consenso da comunidade a respeito das etapas da engenharia de domínio, a saber: **análise de domínio**, **projeto de domínio** e **implementação do domínio**. (KANG *et al.*, 1990), (GRISS *et al.*, 1998), (SIMOS & ANTHONY, 1998), (BRAGA, 2000).

Os modelos de domínio produzidos durante a *análise de domínio* devem refletir sua **variabilidad**e, ou seja, as similaridades e diferenças entre as aplicações do domínio. Métodos de engenharia de domínio (ex. FODA (KANG *et al.*, 1990) e FeatuRSEB (GRISS *et al.*, 1998)) buscam criar estes modelos para a representação das características comuns e variáveis dentre estas aplicações.

Características, também conhecidas pelo seu nome em inglês *features*, são artefatos propostos por KANG *et al*. (1990) no contexto de reutilização de software, mais especificamente em ED. Características representam as capacidades/abstrações do domínio obtidas por especialistas, usuários e a partir de sistemas já existentes, durante a fase de análise do domínio (BLOIS, 2006).

Do ponto de vista da variabilidade, um elemento no domínio pode ser classificado da seguinte forma (CLAUSS, 2001), (BOSCH, 2004), (OLIVEIRA, 2006):

- **Ponto de Variação**: reflete a parametrização no domínio de uma maneira abstrata e deve ser configurável através de suas variantes;
- **Variantes**: funções/conceitos disponíveis que devem necessariamente estar ligados a um ponto de variação, atuando como alternativas de configuração de seu respectivo ponto de variação;
- **Invariante**: elementos fixos, que não são configuráveis no domínio;

Ortogonalmente, do ponto de vista da opcionalidade, um elemento do domínio pode ser classificado como:

- **Opcional**: elemento que pode ou não pertencer a uma aplicação derivada de um domínio.
- **Mandatória**: elemento que obrigatoriamente deve ser instanciado por todas as aplicações derivadas de um domínio.

O **projeto de domínio** tem como principal objetivo a construção de arquiteturas de software capazes de descrever os requisitos identificados durante a engenharia de domínio e representados através do modelo de domínio. Quando estas arquiteturas são especificadas através de elementos especializados no domínio, generalizados para uso efetivo dentro do domínio e compostos em um padrão de estrutura eficaz para a construção de uma família de aplicações, elas são definidas como a **arquitetura de referência de domínio,** ou **DSSA** (*Domain Specific Software Architecture*) (HAYES, 1994).

A criação de uma arquitetura de referência é importante na engenharia de domínio, pois ela é a base para a instanciação de aplicações em um domínio de aplicação específico. Um de seus principais objetivos é propor uma solução estruturada para a construção de uma família de aplicações capazes de atender aos requisitos de um determinado domínio. Através da reutilização de arquiteturas, o compartilhamento de idéias, métodos, componentes e conhecimento dentro de um conjunto de aplicações similares, tornam-se mais efetivos para se obter níveis superiores de qualidade e produtividade durante o processo de construção de sistemas (XAVIER, 2001). Um dos motivos para que isto ocorra, segundo XAVIER, é que as organizações têm considerado suas arquiteturas como recursos que devem ser mantidos e reutilizados tanto quanto for possível. Cada arquitetura criada representa uma grande quantidade de tempo e esforço especializado e as organizações vêem, como uma forma de maximizar estes investimentos, a reutilização destas arquiteturas na construção de aplicações similares.

É importante salientar a distinção entre uma arquitetura de referência de domínio e a arquitetura de uma aplicação. Enquanto a arquitetura de referência de domínio é criada na fase de projeto de engenharia de domínio para atender a toda uma família de aplicações em um domínio, a arquitetura de uma aplicação é a instância (ou refinamento) de uma arquitetura de referência de domínio, representando uma única aplicação.

Na etapa de **implementação do domínio**, os requisitos levantados durante as fases de análise e projeto de domínio são transformados em um modelo implementacional, envolvendo a identificação, construção ou extração de componentes reutilizáveis no domínio.

3.2 - Linhas de Produto

O termo **linha de produtos de software (LPS)** vem sendo utilizado para definir um novo paradigma em engenharia de software, cujo propósito é guiar organizações no desenvolvimento **para** e **com** reutilização. De fato, é uma vertente da engenharia de domínio, cujo foco foi transferido para o âmbito empresarial (LEE *et al.*, 2002).

Segundo o SEI (*Software Engineering Institute*) (2007), uma **linha de produtos de software** é um conjunto de 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 (*core assets*), de forma predefinida.

Os sistemas que integram uma LPS possuem, normalmente, características peculiares a um determinado domínio e são criados a partir de uma base de componentes comuns, moldados de acordo com as regras definidas pela arquitetura. A linha de produtos de software apresenta muitas semelhanças em relação à ED . Ambas usam uma arquitetura como a base para seus artefatos reutilizáveis. Ao passo que na engenharia de domínio, os componentes são organizados na arquitetura de referência (DSSA), na linha de produtos de software, existe a arquitetura de linha de produtos (*PLA-Product Line Architectures*) que desempenha papel semelhante. A variabilidade também é considerada na linha de produtos, e representa os pontos onde as características dos produtos que a compõem podem se diferenciar. Os princípios de construção são os mesmos, uma vez que ambas representam arquiteturas de referência ou *frameworks* para uma família de aplicações, construídas com base nos requisitos de um domínio ou linha de produtos.

O núcleo de artefatos forma a base do paradigma da linha de produtos de software (NORTHROP, 2002) e pode ser definido como o conjunto de elementos que serão reutilizados pelos sistemas a serem desenvolvidos. Nele estão inseridos artefatos como a arquitetura do software, componentes de software reutilizáveis, modelos de domínio, requisitos, modelos de desempenho, cronogramas, orçamentos, planos e casos de testes, planejamentos e descrição de processos (OLIVEIRA, 2006).

As três principais atividades envolvidas em uma linha de produtos de software são: o **desenvolvimento do núcleo de artefatos**, o **desenvolvimento de produtos** e o **gerenciamento da linha de produtos**, como ilustrado na Figura 3.4.

No **desenvolvimento do núcleo de artefatos**, o principal objetivo é "desenvolver" os artefatos necessários envolvidos na linha de produtos de software. Entede-se por "desenvolver" um artefato, desde sua construção do zero, até a compra de terceiros. Os tipos de artefatos "desenvolvidos" também são variados, podendo abranger desde componentes reutilizáveis, arquiteturas de software, documentação de componentes até *frameworks*. De acordo com NORTHROP (2002), a arquitetura é a peça chave de uma linha de produtos de software e deve ser comum a todos seus produtos relacionados. Em (CLEMENTS, 1999), o autor define os três principais produtos resultantes das atividades desta fase como sendo: e**scopo da linha de produção**, cujo objetivo é definir o escopo da linha de produtos de software e os

produtos que a constituirão; **artefatos do núcleo**, envolve todos os artefatos que formam a base da linha de produtos de software: componentes reutilizáveis (criados ou adquiridos de terceiros), arquiteturas de software, documentação de componentes, casos de teste, dentre outros; **plano de produção**, define como os produtos deverão ser produzidos com os artefatos do núcleo.

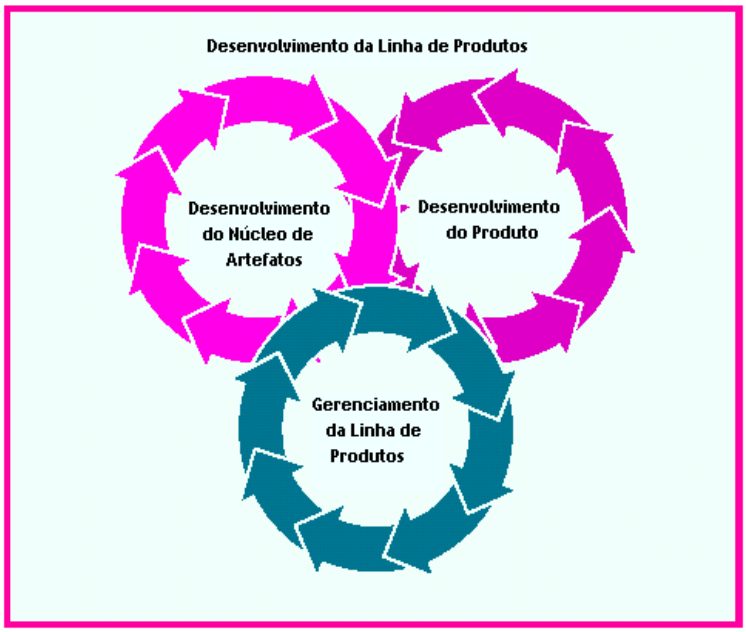

Figura 3.4– Principais Atividades da Linha de Produtos de Software

(Adaptado de (NORTHROP, 2002))

Durante o **desenvolvimento de produtos**, produtos são desenvolvidos a partir dos artefatos do núcleo, como ilustrado na Figura 3.5. Os artefatos mais comuns presentes no núcleo são (GIMENES & TRAVASSOS, 2002):

1. modelo do domínio;
2. framework de arquitetura de linha de produtos, que especifica a arquitetura genérica para todos os produtos;
3. componentes reutilizáveis que serão integrados à arquitetura para gerar um produto.

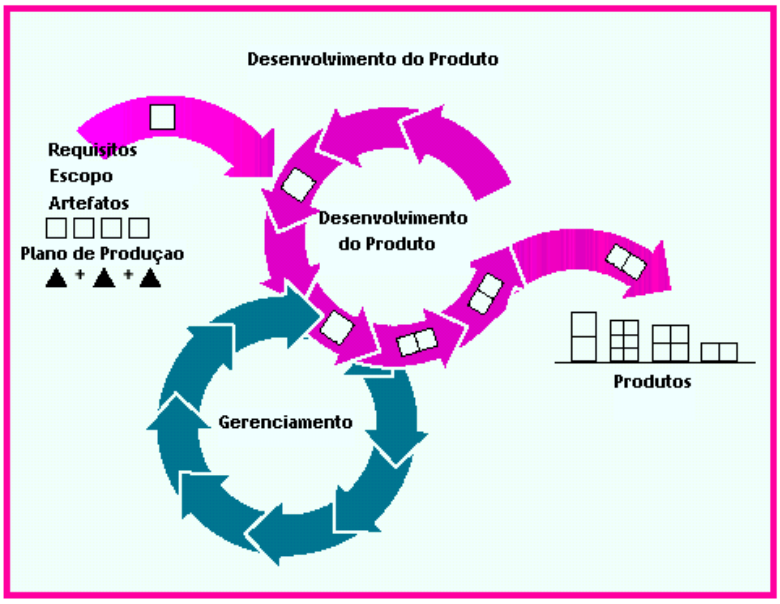

Figura 3.5 – Desenvolvimento do Produto

(Adaptado de (NORTHROP, 2002))

Em (OLIVEIRA, 2006), são citadas as principais atividades durante esta fase: **análise baseada no modelo de domínio**, cujo objetivo é analisar o modelo de domínio para determinar até que ponto os componentes disponíveis na arquitetura genérica atendem as necessidades do cliente, e verificar se será necessário a criação de novos componentes no núcleo; **instanciação da arquitetura do produto**, objetivando a tomada de decisões acerca de quais elementos do núcelo deverão ser selecionados a partir da arquitetura genérica, para se chegar na arquitetura específica do produto a ser desenvolvido; **povoamento da arquitetura**, cuja finalidade é preencher a arquitetura específica do produto com os componentes selecionado do núcleo;

O **gerenciamento da linha de produtos** é responsável pelo sincronismo entre as equipes que criam os artefatos do núcleo e os que geram os produtos. O sucesso e viabilidade da linha de produtos de software depende muito das atividades realizadas nesta fase. Dentre estas atividades, pode-se destacar a organização, supervisão, integração e documentação tanto do desenvolvimento do núcleo quanto do desenvolvimento de produtos.

3.3 - Abordagens de Engenharia de Domínio e o seu apoio à Especificação de Arquiteturas de Referência de Domínio (DSSAs)

O método mais conhecido e considerado como precursor dos demais é o FODA (KANG *et al.*, 1990). Este método está focado mais nas atividades relacionadas à fase de análise de domínio, contemplando atividades de análise de contexto e a modelagem do domínio. No entanto, não existe ferramental apropriado para o desenvolvimento das atividades desta fase. O apoio à construção de arquitetura de referências é previsto apenas de forma conceitual e a literatura disponível trata desta questão de maneira superficial (BLOIS, 2006).

Em (BRAGA, 2000), é proposto um processo de engenharia de domínio cujo propósito é unir os aspectos de reutilização e entendimento do domínio que são providos pelos processos de engenharia de domínio existentes e o detalhamento do desenvolvimento de componentes, provido pelos processos de desenvolvimento baseados em componentes. Ainda que aborde o ciclo de vida completo de engenharia de domínio, observa-se que o foco do processo ainda é na análise de domínio (BLOIS, 2006).

Os processos de criação e instanciação de arquitetura de referências resultantes do projeto de pesquisa ARPA (METTALA & GRAHAM, 1992) foram descritos em (TRACZ & HAYES, 1994). Para a criação da arquitetura de referência, são considerados três 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. Neste método, deve existir pelo menos uma arquitetura de referência capaz de atender aos requisitos funcionais e não funcionais do domínio, muito embora, segundo os autores, nem todos os requisitos sejam satisfeitos por uma única arquitetura, o que pode levar a existência de mais de uma arquitetura de referência para um dado domínio.

O *CBD-Arch-DE*, método de engenharia de domínio proposto por BLOIS (2006), busca oferecer um maior apoio à etapa de projeto do domínio do que os outros métodos de engenharia de domínio pesquisados (i.e. FODA (KANG *et al.*, 1990), FORM (KANG *et al.*, 2002), FODAcom (VICI & ARGENTIERI, 1998), FeatuRSEB (GRISS *et al.*, 1998), ODM (SIMOS & ANTHONY, 1998) e Odyssey-DE (BRAGA,

2000)). Em sua abordagem, o *CBD-Arch-DE* mapeia o modelo de características do domínio para modelos de mais baixo nível como casos de uso e modelos de tipos de negócio, o que permite a geração de componentes de negócio, utilitários e infra-estrutura do domínio. A abordagem *CBD-Arch-DE* visa apoiar a criação de arquiteturas de referência do domínio através de um processo de engenharia de domínio com suporte a DBC (Desenvolvimento Baseado em Componentes).

A abordagem proposta por XAVIER (2001) visa a criação e instanciação de arquiteturas de referência em infra-estruturas de reutilização, através da seleção de determinado padrão arquitetural. Esta seleção é realizada através da avaliação do relacionamento existente entre os requisitos de referência e os padrões arquiteturais existentes, que é calculada pela ferramenta MENTOR, integrada ao ambiente do Odyssey. Como resultado, obtém-se um espaço vetorial com os requisitos do domínio e os requisitos priorizados pelos padrões arquiteturais. Com base nesse cálculo, a ferramenta MENTOR indica os padrões, em ordem de prioridade, mais adequados para guiar a solução computacional para o domínio.

Conforme pode ser observado, nem todas as abordagens pesquisadas de engenharia de domínio suportam a criação de arquitetura de referências e, quando suportadas, todas tratam este problema através de um direcionamento *top-down* (i.e. da análise para o projeto e implementação). Por outro lado, de acordo com KANG *et al* (1990), uma das fontes de informação essenciais para a engenharia de domínio são os sistemas legados disponíveis no domínio. Sistemas legados são ricos em conhecimento sobre o negócio, e é comum ser a única fonte de informação disponível para modelar o domínio. Desta forma, seria de grande utilidade para as empresas que desenvolvem software, a existência de um maior apoio ao engenheiro de domínio na análise, de forma sistemática e, se possível, com um certo grau de automação, das informações destes sistemas legados. Estes sistemas representam grande esforço investido e são uma importante fonte de conhecimento para a construção de um modelo de domínio que possa servir como base para a criação de uma arquitetura de referência.

3.4 - Abordagens de Linha de Produtos de Software e o seu apoio à Arquitetura

O método MAP (*Mining Architectures for Product Line Evaluation*), proposto em (STOERMER & O'BRIEN, 2001), visa extrair e analisar variabilidades de

arquiteturas de sistemas de um mesmo domínio para gerar uma linha de produtos de software. Sua abordagem é *bottom-up* para as tarefas de mineração das arquiteturas dos produtos e *top-down* para definir estilos arquiteturais e atributos para as arquiteturas extraídas. Para suportar a criação de linha de produtos de software, STOERMER e O'BRIEN ressaltam o quão é importante que as tomadas de decisões sejam baseadas em critérios técnicos em detrimento de critérios não-técnicos. Desta forma, dividiram as atividades do MAP em seis fases: **preparação**, **extração**, **composição**, **qualificação**, **avaliação** e **pós-avaliação**.

Na **preparação,** são considerados aspectos técnicos e organizacionais, dentre eles, a avaliação da disponibilidade de recursos na empresa (ex.. desenvolvedores, engenheiros) e a seleção dos produtos mais representativos. Em seguida, na fase de **extração,** são extraidos modelos de implementação a partir dos sistemas candidatos. Tais modelos são na verdade elementos dos códigos-fonte (classes, arquivos,etc) e suas relações. Além do modelo estático, aspectos dinâmicos como tempo de execução de funções e inter-relação entre os processos também são obtidos nesta fase. Os modelos da fase de extração servem como entrada para a fase de **composição**, onde visões de componentes são criadas em um nível de abstração arquitetural. É nesta fase que são identificadas estruturas para posterior detecção das variabilidades. Na fase de **qualificação**, os componentes e suas relações, obtidas na fase de composição, são mapeados para estilos arquiteturais e atributos. Este mapeamento é realizado através da consulta à arquitetos e desenvolvedores, onde são avaliados os atributos de qualidade que motivaram a escolha de determinados estilos.

A comparação entre as arquiteturas candidatas é realizada durante a fase de **avaliação**. São identificadas as variabilidades entre os produtos que deverão compor a linha de produtos, levando-se em consideração os componentes, suas relações, funcionalidades específicas de clientes, protocolos, sistemas operacionais e plataforma de hardware dos produtos. Os autores argumentam que comparações em níveis inferiores ao nível de componentes não são adequadas devido ao fato de que os produtos candidatos terem sido desenvolvidos por diferentes equipes e, portanto, não possuem a mesma convenção de nomes e segmentação de componentes. Com base nas variabilidades identificadas durante a fase anterior, finalmente são realizadas as atividades de **pós-avaliação.** O objetivo desta fase é decidir sobre a implantação ou não de uma linha de produtos na empresa, lembrando que os resultados obtidos na fase de avaliação podem ser usados como subsídios para sua concepção.

GOMAA propõe um processo de desenvolvimento de linha de produtos, denominado PLUS (GOMAA, 2004). Este processo se inicia a partir da especificação dos casos de uso de uma linha de produtos de software, englobando o modelo de características, a modelagem dinâmica por meio de máquinas de estado e de diagramas de interação, e também a, modelagem de classes, modelagem de contexto (i.e. objetos externos ao sistema e a sua comunicação com objetos da linha de produtos) e modelagem dos componentes de software na arquitetura, na qual um padrão arquitetural deve ser adotado.

A definição das variabilidades no método PLUS se inicia no modelo de casos de uso, sendo disseminada para os demais modelos. Segundo o autor, o desenvolvimento é iterativo e evolutivo, permitindo a adoção de uma abordagem de engenharia progressiva ou reversa. Entretanto, no que diz respeito à engenharia reversa, e ao método que seria utilizado na comparação de sistemas legados, ainda que o autor mencione algumas diretrizes que podem ser adotadas, no intuito de realizar a consistência entre os modelos e detecção de variabilidades, fica clara a necessidade de um apoio mais efetivo. Uma desvantagem que pode ser observada no método é, por exemplo, a não-indicação sobre o modo como os modelos podem ser extraídos dos sistemas existentes e não-consideração de algumas questões, tais como diferenças de nomenclaturas entre os diferentes sistemas ou diferenças estruturais entre os modelos.

3.5 - Considerações Finais

Neste capítulo foram apresentados conceitos de Engenharia de Domínio (ED) e Linha de Produtos (LPS), bem como abordagens tanto de ED como LPS e seus respectivos suportes à criação de DSSAs.

A maioria das abordagens de ED e LP estudadas apóiam a especificação de DSSAs no sentido da engenharia progressiva, i.e., de cima para baixo. Elas esperam que o modelo de domínio esteja definido para que a arquitetura de referência seja criada. Entretanto, quando o modelo de domínio não está disponível, como no caso de sistemas legados, é necessário que existam formas de se obter as arquiteturas destes sistemas (ex: engenharia reversa) e métodos para que as mesmas sejam comparadas. Nestes casos, a DSSA pode ser criada a partir da identificação das similaridades e diferenças entre os elementos destas arquiteturas. Dentre as abordagens estudadas, que suportam a comparação de arquiteturas de software, não foram identificadas claramente, como são

tratados os problemas inerentes à comparação de arquiteturas de sistemas diferentes, pertencentes a um mesmo domínio. Como mencionado anteriormente, tais arquiteturas tendem a apresentar um número grande de elementos arquiteturais com sintaxe distintas que possuem o mesmo significado.

Neste contexto, é proposta, neste trabalho, uma abordagem que visa a comparação de arquiteturas de sistemas pertencentes a um dado domínio, identificando suas similaridades, opcionalidades e variabilidades, com o propósito de auxiliar o Engenheiro de Domínio na criação de DSSAs.

Capítulo 4: *ArchToDSSA*: Uma Abordagem para a Criação de Arquiteturas de Referência de Domínio a partir da Comparação de Modelos Arquiteturais de Aplicações

Nos capítulos anteriores foram abordadas as vantagens da criação de arquiteturas de referência no contexto da reutilização de software. Foram ainda apresentados alguns requisitos desejáveis para a criação destas arquiteturas de referência, entre eles a necessidade da representação dos elementos arquiteturais sem ambigüidades. Além disso, foi mostrada a necessidade da criação de mecanismos que auxiliem o engenheiro de domínio na identificação, compreensão e propagação das mudanças que se refletem na arquitetura do software, sejam estas mudanças de requisitos ou diferenças entre diversas versões de uma mesma arquitetura, como diferenças de nomenclatura, por exemplo. Algumas abordagens foram descritas, mostrando que tais abordagens apresentam falhas na tarefa de comparar arquiteturas de sistemas legados em um domínio e, ainda, que muitas delas não possuem como objetivo final apoiar o engenheiro de domínio na construção de arquiteturas de referência a partir dos resultados obtidos nestas comparações.

Neste sentido, o presente capítulo vem apresentar a abordagem denominada *ArchToDSSA*, cujo objetivo é apoiar o engenheiro de domínio na tarefa de criação de uma arquitetura de referência, a partir da análise das informações disponíveis em arquiteturas existentes em um domínio. Tais arquiteturas podem ou não ser provenientes de sistemas legados. Segundo KANG et al (2002), sistemas legados disponíveis no domínio são uma das principais fontes de informação que podem ser usadas na criação de uma arquitetura de referência em um domínio específico (DSSA - *Domain Specific Software Architecture*). No entanto, a adoção da abordagem ArchToDSSA independe do fato de as arquiteturas serem provenientes de sistemas legados.

O capítulo está organizado da seguinte maneira: a seção 4.1 apresenta uma visão geral da abordagem proposta, *ArchToDSSA*; na seção 4.2, é apresentada a primeira fase, i.e., a Detecção de Equivalências e Opcionalidades. A segunda fase, Detecção de Variabilidades, é apresentada na seção 4.3. Em seguida, na seção 4.4, é apresentada a fase de Seleção de Elementos. Finalmente, na seção 4.5, são feitas algumas considerações sobre a abordagem *ArchToDSSA*.

4.1 - A abordagem *ArchToDSSA*

ArchToDSSA é uma abordagem que visa analisar arquiteturas já existentes - de sistemas legados ou não - de maneira comparativa, no intuito de identificar elementos-chave que possam compor uma arquitetura de referência em um domínio. No contexto desta dissertação, tais elementos são denominados **elementos arquiteturais**. Como mencionado na seção 1.1, elementos arquiteturais são, abstrações responsáveis por representar as entidades que implementam as funcionalidades especificadas em um software. A abordagem é direcionada para arquiteturas orientadas a objetos, descritas através de diagramas de pacotes e de classes da UML.

A partir da análise de abordagens da literatura, e de experiências práticas, pode-se definir alguns requisitos desejáveis em uma abordagem para a especificação de uma arquitetura de referência a partir de comparação de arquiteturas existentes, tais como:

- A comparação entre elementos arquiteturais deve levar em consideração o fato de que elementos diferentes, em diferentes arquiteturas, podem ter sido criados com sintaxes diferentes e mesma semântica, isto é, dois ou mais elementos que tenham o mesmo significado podem ter sido criados com nomenclaturas diferentes. Para tanto, mecanismos como dicionário e sinônimos, divisão das palavras em partes menores, e listas de partes de palavras a serem ignoradas, podem ser utilizados como recurso, para auxiliar no processo de comparação;
- Devem ser permitidas ao engenheiro de domínio certas escolhas, tais como: o número mínimo de arquiteturas envolvidas para se considerar equivalência entre elementos, sua opcionalidade, pontos de variação e suas variantes;
- Pode ser definido se todos os elementos poderão ser comparados entre si ou se somente elementos do mesmo tipo serão comparados (ex. comparação apenas entre classes, ou entre classes e pacotes);
- A arquitetura de referência deve ser especificada de forma inteligível, em uma representação que seja familiar ao engenheiro de domínio, onde possam ser identificados com clareza tanto os elementos arquiteturais que compõem a arquitetura de referência, quanto seus atributos de opcionalidade e variabilidade;
- Deve ser previsto um apoio ferramental, no intuito de melhorar a produtividade do engenheiro de domínio e reduzir o esforço humano.

Os requisitos mencionados acima serão abordados neste capítulo, com exceção do último, que trata do apoio ferramental, que será explorado no próximo capítulo.

As arquiteturas dos sistemas legados podem ser extraídas através de um processo de engenharia reversa aplicado sobre sistemas disponíveis para análise (VASCONCELOS, 2007),(RICHNER & DUCASSE, 1999), (RIVA & RODRIGUEZ, 2002). Embora *ArchToDSSA* não contemple a extração destas arquiteturas, ela está inserida em um contexto maior de abordagem de recuperação de arquiteturas de sistema legados para a criação de uma arquitetura de referencia, denominada *LegaToDSSA* (VASCONCELOS, 2007), onde esta etapa de extração das arquiteturas é contemplada. Assim, a extração e comparação de arquiteturas para a criação de uma DSSA são abordadas de forma complementar, oferecendo um maior apoio ao engenheiro de domínio.

Na *ArchToDSSA*, o conjunto de arquiteturas que serão analisadas recebe o nome de conjunto de trabalho *(working set)*. É importante ressaltar que o conjunto de trabalho não tem por objetivo representar todo o conjunto de aplicações de um determinado domínio. Entretanto, espera-se que o engenheiro possa trabalhar com um conjunto o mais completo possível.

O processo sugerido contempla várias tarefas divididas em três fases distintas, a saber:

1. Detecção de Equivalências e Opcionalidades;
2. Detecção de Variabilidades;
3. Criação da Arquitetura de Referência.

O resultado obtido em cada fase serve de entrada para a fase seguinte, como ilustra a Figura 4.6.

Na primeira fase (Fase 1 – Detecção de Equivalências e Opcionalidades), o objetivo é identificar as similaridades (equivalências) e diferenças entre os elementos das arquiteturas analisadas, isto é, as arquiteturas constantes do conjunto de trabalho. Dessa forma, é possível identificar elementos mandatórios e/ou opcionais no domínio, ou seja, as "Opcionalidades" dos elementos do domínio, conforme definido na seção 3.1.

Na segunda fase (Fase 2 – Detecção das Variabilidades), as equivalências obtidas na Fase 1 são analisadas para que se possa definir as "Variabilidades" do domínio, isto é, pontos de variação e suas respectivas variantes, (ver seção 3.1).

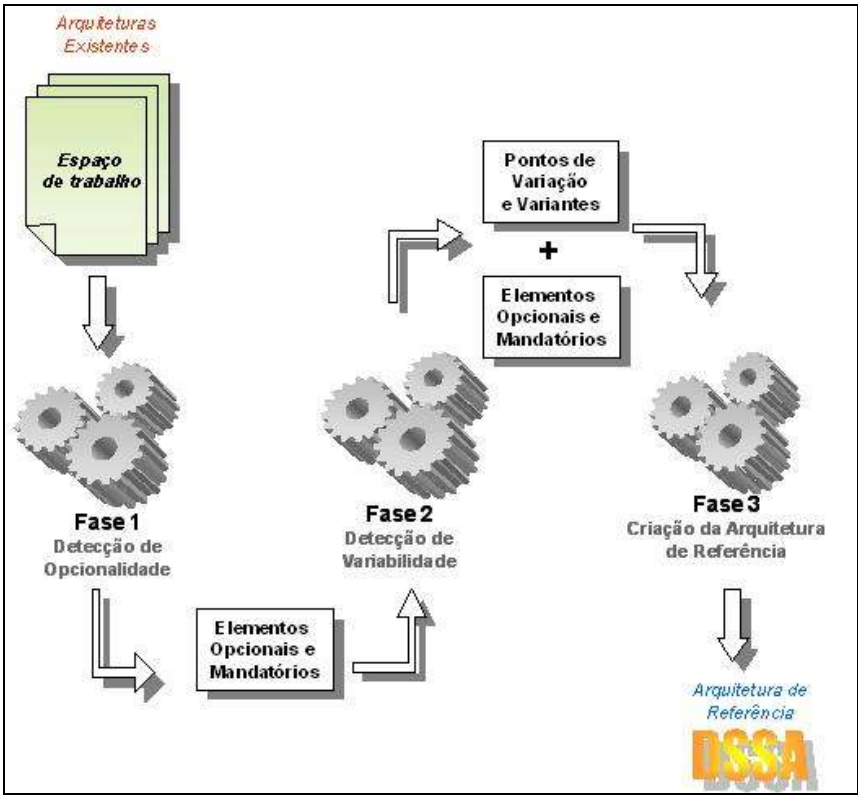

Figura 4.6 – Abordagem *ArchToDSSA*

Finalmente, na última fase (Fase 3 – Criação da Arquitetura de Referência), são definidos elementos arquiteturais, oriundos de arquiteturas disponíveis no conjunto de trabalho, que deverão compor uma possível arquitetura de referência para o domínio. Vale ressaltar que é possível a criação de mais de uma arquitetura de referência para o domínio, a partir da comparação de arquiteturas existentes. Além disso, a arquitetura de referência resultante da abordagem é parcialmente especificada.

Uma vez que a *ArchToDSSA* está inserida no contexto do ambiente Odyssey, optou-se por adotar a notação Odyssey-FEX (OLIVEIRA, 2006). A notação Odyssey-FEX foi proposta mediante a identificação de falhas existentes, em diversas notações da literatura, no que dizia respeito à definição e representação de conceitos como elementos mandatórios e opcionais, pontos de variação, variantes e invariantes em um domínio. Ao longo das fases da abordagem *ArchToDSSA,* faz-se necessário identificar tais elementos.

Para facilitar o entendimento da abordagem, são utilizadas como exemplo, ao longo de todo o capítulo, arquiteturas do domínio de Telefonia Móvel, que será brevemente explicado na seção 4.2.

Cada uma das fases de ArchToDSSA é discutida separadamente, nas seções 4.3, 4.4 e 4.5, respectivamente.

4.2 – Descrição do Domínio de Telefonia Móvel

O domínio de Telefonia Móvel abrange conceitos e funcionalidades que podem estar presentes em um software desenvolvido para um telefone celular[5]. As três arquiteturas do domínio de Telefonia Móvel, utilizadas como exemplo, são apresentadas na Figura 4.7.

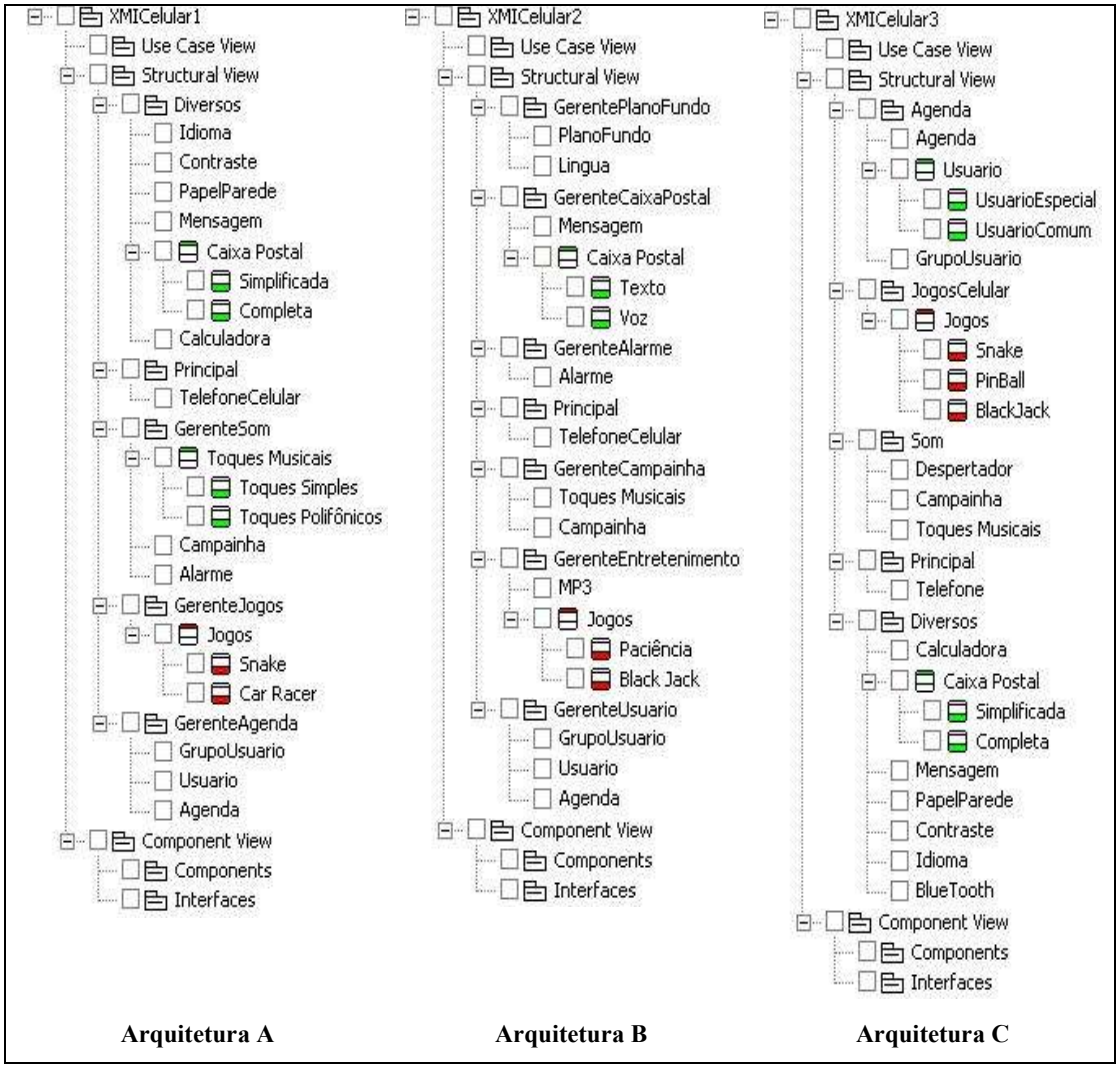

Figura 4.7 – Arquiteturas do domínio de Telefonia Móvel

[5] Os conceitos do domínio de Telefonia Móvel utilizados podem ser encontrados em (NOKIA, 2007)

Atualmente, pode ser encontrada uma grande diversidade de telefones celulares, que apresentam as mais variadas funcionalidades. No entanto, alguns conceitos e funcionalidades são intrínsecos e estão presentes a todos os tipos de software. Dentre estes podem ser citados: Campainha, que representa o toque do telefone, Agenda, onde são armazenados números de telefone, e Caixa Postal, onde mensagens de voz são deixadas para o usuário. De acordo com as definições apresentadas em (OLIVEIRA, 2006), tais conceitos e funcionalidades são denominados **mandatórios**, isto é presentes em todos os tipos de software para telefone celular. Além disso, alguns conceitos, embora sejam mandatórios, apresentam alternativas, dando ao usuário a opção de escolha na hora da compra. A estes conceitos denomina-se **ponto de variação**, e a suas alternativas, denomina-se **variantes**. Como exemplo, tem-se o conceito de Caixa Postal. Este conceito representa um ponto de variação do tipo de caixa postal do aparelho celular, que tem como variantes as opções "simplificada" ou "completa".

Outros conceitos e funcionalidades são oferecidos no domínio de telefonia celular, mas podem ser encontrados em apenas alguns aparelhos, constituindo um diferencial. Tais conceitos são denominados **opcionais**. Dentre estes podem ser citados: *Bluetooth*, que é uma tecnologia que permite que uma conexão sem fio de curto alcance seja estabelecida com outro dispositivo compatível (celulares, laptops, câmeras digitais), Acesso à Internet, Envio de mensagens de texto, como e-mails e recados, Jogos e Recebimento de toques musicais, que representa a funcionalidade de alteração do tipo de campainha, reproduzindo-se um toque especial, como uma música conhecida, geralmente oferecido pela operadora de telefonia móvel.

Alguns destes conceitos também podem apresentar alternativas, i. e., os conceitos opcionais também podem ser pontos de variação e/ou variantes. Como exemplo de pontos de variação, tem-se Recebimento de toques musicais e Jogos. Os toques musicais recebidos podem possuir variantes, i. e., podem ser do tipo monofônicos ou polifônicos. Toques monofônicos são toques musicais que a maioria dos aparelhos celulares pode reproduzir. O celular só precisa executar uma nota por vez para reproduzir um toque monofônico. Toques polifônicos são toques musicais que podem consistir em várias notas de uma vez, reproduzidas por meio de um alto-falante em vez de uma campainha. São toques mais elaborados, com som muito próximo das músicas reais. No caso dos jogos, existe uma variedade deles que pode estar presente em um ou outro aparelho de telefone celular. Como exemplo, tem-se *Car Racer*, que é

um jogo de corrida de carros, *Snake*, jogo cujo objetivo é guiar uma cobra para que seja alimentada, jogos de Tênis, e outros.

4.3 - Fase 1: Detecção de Equivalências e Opcionalidades

As atividades descritas na Fase 1 são retratadas na Figura 4.8, a seguir.

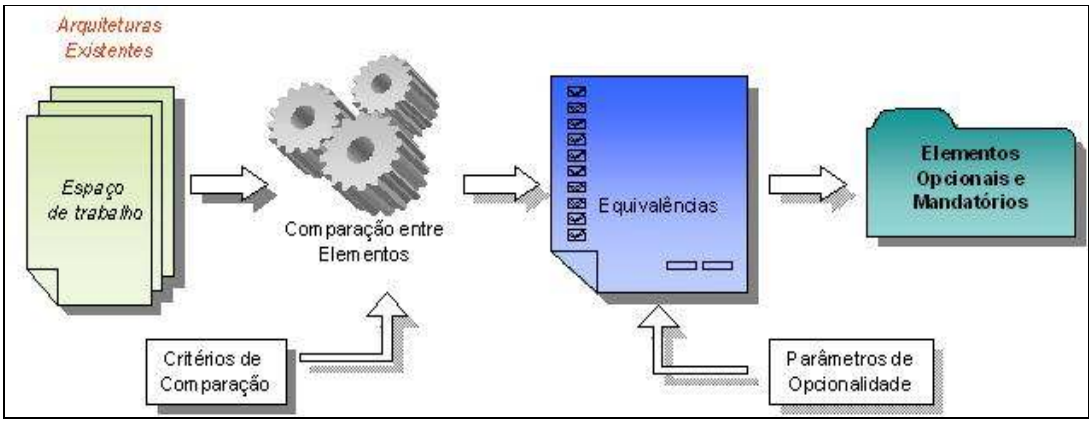

Figura 4.8 ***ArchToDSSA*** **– Fase 1**

De acordo com a Figura 4.8, nesta fase, arquiteturas existentes são comparadas, e, seguindo alguns critérios de comparação descritos nesta seção, equivalências são identificadas. Em seguida, mediante parâmetros de opcionalidade aplicados sobre as equivalências identificadas, são determinados os elementos opcionais e mandatórios que poderão compor uma arquitetura de referência de domínio.

Para a realização da primeira atividade, a detecção de equivalências, é necessário introduzir primeiramente o conceito de equivalência (*match)* e definir um algoritmo para sua identificação. O conceito de equivalência, bem como a forma como as equivalências e Opcionalidades são definidas, são explicados a seguir.

Uma **equivalência** é um elemento identificado unicamente, que representa um elo de ligação entre elementos arquiteturais equivalentes, pertencentes a uma mesma arquitetura ou a arquiteturas diferentes. Neste contexto, o termo "equivalente" é usado com a finalidade de identificar elementos arquiteturais distintos que possuam o mesmo significado. Em relação a uma equivalência, assume-se, ainda que:

a) o número de elementos pertencentes a uma equivalência está compreendido no intervalo (2,...,n);

b) uma vez pertencente a uma equivalência, um elemento não pode pertencer a outra.

c) uma equivalência possui *reciprocidade*, isto é, se a_1 é equivalente a b_1, então b_1 é equivalente a a_1,

d) uma equivalência possui *transitividade*, isto é, se a_1 é equivalente a b_1, e b_1 equivalente a c_1, então a_1 é equivalente a c_1.

Na busca por elementos equivalentes, comparar aplicações distintas em um mesmo domínio pode vir a ser uma tarefa mais árdua do que comparar versões diferentes de uma mesma aplicação. Isso porque, embora um conjunto de trabalho contenha aplicações de um mesmo domínio, podem existir casos onde elementos arquiteturais equivalentes estejam descritos de forma diferente. Tal fato ocorre porque, apesar das aplicações analisadas fazerem parte de um mesmo domínio, elas podem não ter sido necessariamente implementadas por uma mesma equipe, em um mesmo período, ou dentro de uma mesma empresa. Isso pode ocasionar uma grande variedade de elementos equivalentes, porém com nomes distintos, nas diferentes aplicações analisadas. Além disso, à medida que cresce o número de aplicações a serem comparadas e/ou o tamanho das arquiteturas extraídas, cresce também o trabalho envolvido na atividade de comparação dessas arquiteturas.

A identificação de equivalências deve ser realizada analisando-se elementos arquiteturais nas diferentes arquiteturas, e para tanto, assume-se a utilização da comparação dos nomes dos elementos arquiteturais.

Dois ou mais elementos arquiteturais de mesmo nome seriam fortes candidatos a originar uma equivalência. No entanto, quando os nomes não são idênticos, determinar a equivalência de elementos arquiteturais passa a depender muito do conhecimento de um engenheiro de domínio. Sendo assim, assume-se na abordagem, além da equivalência por nomes idênticos, a utilização de alguns critérios de comparação, como indicado na Figura 4.8, tais como: utilização de um dicionário de sinônimos, para nomes não-idênticos; divisão dos nomes a serem comparados em partes; utilização de lista de palavras ou parte de palavras que podem ser ignoradas na comparação; comparação entre elementos do mesmo tipo; e número mínimo de arquiteturas equivalentes. Cada um desses critérios é explicado na próxima seção.

4.3.1 - Critérios para Detecção de Equivalências

Os critérios para detecção de equivalências na ArchToDSSA são provenientes da observação das diferenças que ocorrem nas diversas arquiteturas. Sendo assim, assumiu-se a utilização de cinco critérios, a saber:

1. Utilização do Dicionário de Sinônimos;
2. Utilização de Lista de Palavras Ignoradas;
3. Divisão de Nomes em Partes;
4. Comparação de Elementos do Mesmo Tipo; e
5. Número Mínimo de Arquiteturas Equivalentes.

Cada um desses critérios é explicado a seguir.

Critério 1 – Utilização do Dicionário de Sinônimos

Na abordagem *ArchToDSSA*, o dicionário de sinônimos consiste de uma lista ligando palavras com um mesmo significado, tal como em um dicionário comum. Essa lista pode ser criada e alimentada pelo engenheiro de domínio com base em seu conhecimento sobre o domínio. As palavras do dicionário podem não representar necessariamente o nome de elementos arquiteturais, mas assim como em qualquer dicionário, ligar cadeia de caracteres equivalentes.

Tomando como exemplo as Arquiteturas A e C da Figura 4.7, um engenheiro de domínio poderia, em seu dicionário de sinônimos, afirmar que, no domínio de Telefonia Móvel, o nome “Alarme” seria equivalente ao nome “Despertador”.

Critério 2 – Utilização de Lista de Palavras Ignoradas

Assim como no dicionário de sinônimos, o conhecimento do engenheiro de domínio é usado para detectar a existência de palavras a serem ignoradas no contexto da comparação de nomes. Cria-se, portanto, uma lista de palavras a serem ignoradas.

Tomando como exemplo as Arquiteturas A e B da Figura 4.7, no caso de a palavra “Gerente” constar da lista de palavras ignoradas, os nomes “GerenteJogos” (Arquitetura A) e “Jogos” (Arquitetura B) passam a ser considerados equivalentes.

Critério 3 - Divisão de Nomes em Partes

Quando dois nomes (nome1 e nome2) são comparados, o primeiro passo é decidir se cada nome será tratado como um todo, ou se será dividido em partes. Caso o nome seja tratado como um todo, cada nome será comparado integralmente. Tomando-

se como exemplo as Arquiteturas A e C da Figura 4.7 tem-se respectivamente, nome1 = "GerenteAgenda", e nome2 = "Agenda". Assim, ao se comparar os dois nomes integralmente, tem-se uma situação de não-equivalência. Do contrário, se for decidido dividir os nomes em partes para efetuar a comparação, o nome será dividido em partes.

Seguindo esse critério, um nome é dividido em partes de tal forma que, quando uma letra maiúscula, hífen, ou *underscore* for encontrado no meio da palavra, uma nova parte é definida. Assim, no exemplo da Figura 4.7, no nome "CaixaPostal" poderão existir duas partes - "Caixa" e "Postal" - e no nome "GerenteCaixaPostal", três partes -"Gerente", "Caixa" e "Postal".

Dessa forma, após dividir nome1 e nome2 em partes, deve ser verificado para cada parte, em cada nome, se a referida parte encontra-se na lista de palavras a serem ignoradas. Em caso afirmativo, tal parte deve ser retirada da lista de partes da palavra.

Uma vez decidido se cada nome terá uma ou mais partes, o próximo passo é comparar cada parte da lista de partes da palavra nome1 com cada parte da lista de palavras do nome2. Nesta comparação, uma parte será considerada igual à outra se sua cadeia de caracteres forem exatamente iguais, ou se suas partes tiverem sido consideradas equivalentes por meio do dicionário de sinônimos. Assim, se o engenheiro criou uma entrada no dicionário de sinônimos em que Idioma = Língua, embora as cadeias de caracteres sejam completamente diferentes, tais elementos arquiteturais serão considerados equivalentes na comparação.

Assume-se, no algoritmo utilizado, que, uma vez encontradas partes iguais em dois nomes, estas duas partes são marcadas e não podem mais ser envolvidas na comparação das partes seguintes dos mesmos nomes. Este critério visa reduzir o número de falsos positivos durante a comparação. A Figura 4.9 exemplifica a idéia.

No primeiro exemplo, as cadeias de caracteres são consideradas iguais, pois cada parte da primeira palavra encontrou seu correspondente em uma parte disponível da segunda palavra, isto é, uma parte que ainda não tinha sido usada em uma comparação. No segundo exemplo, as cadeias de caracteres são consideradas diferentes porque, embora a última parte da primeira palavra (Ba) tivesse como correspondente a primeira parte da segunda palavra (Ba), esta já havia sido usada em uma comparação.

Critério 4 - Comparação de Elementos do Mesmo Tipo

Na comparação entre elementos arquiteturais, o engenheiro pode decidir se um elemento de uma arquitetura poderá ser comparado com qualquer elemento de outra

arquitetura (ex: classes com pacotes), ou se deverão ser comparados somente elementos do mesmo tipo (ex: pacotes com pacotes, classes com classes).

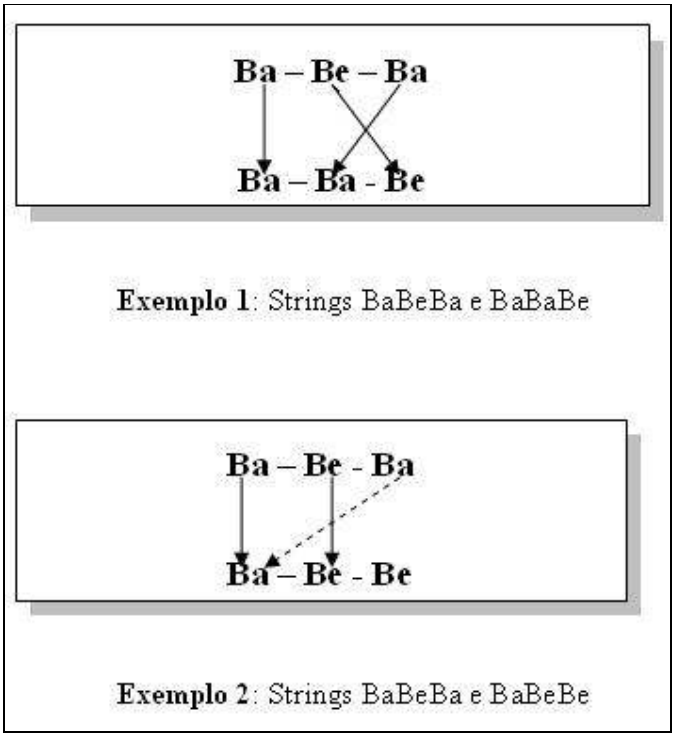

Figura 4.9 – Critério de Comparação entre nomes

Assim, tomando-se como exemplo as arquiteturas da Figura 4.7, caso o engenheiro decida que somente elementos do mesmo tipo podem ser comparadas, ainda que existisse a palavra "Gerente" na lista de palavras a serem ignoradas, o pacote "GerenteJogos", na Arquitetura A, não poderia ser considerado equivalente à classe "Jogos" na Arquitetura B, pois a comparação não poderia ser feita entre pacotes e classes.

Critério 5 - Número Mínimo de Arquiteturas Equivalentes

Neste critério, o engenheiro de domínio pode determinar qual o número mínimo de arquiteturas às quais elementos equivalentes devem pertencer para ser considerada realmente uma equivalência.

Assim, tomando-se como exemplo as arquiteturas da Figura 4.7 caso o engenheiro tenha determinado que uma equivalência existirá realmente somente se forem encontrados elementos equivalentes nas três arquiteturas, o elemento "Alarme"

não seria considerado uma equivalência, pois ele não consta na Arquitetura C, apenas nas Arquiteturas A e B. Do contrário, se o número mínimo de arquiteturas determinado pelo engenheiro fosse 2, o elemento "Alarme" seria considerado realmente uma equivalência.

Levando em conta esses critérios, um algoritmo é sugerido para a identificação de equivalências, como descrito a seguir.

4.3.2 - Descrição do Algoritmo de Detecção de Equivalências

Sejam A, B, e C. arquiteturas que estão sendo comparadas, isto é, participantes do conjunto de trabalho. Sejam a_1, a_2, ...a_n, b_1, b_2, ...b_n e c_1, c_2, ..., c_n elementos arquiteturais das arquiteturas A, B e C, respectivamente. Sejam ce_1, ce_2,...ce_n, elementos candidatos a equivalências.

Para detectar equivalências, é realizada uma comparação entre elementos das arquiteturas da seguinte forma:

- Escolhe-se o primeiro elemento disponível, isto é, que não faça parte de alguma outra equivalência, em qualquer uma das arquiteturas do conjunto de trabalho.
- Define-se este elemento como a_1 (primeiro elemento disponível da arquitetura A).
- Percorrem-se todos os outros elementos da arquitetura A (a_2, a_3,..., a_n) e também cada elemento das outras arquiteturas do conjunto de trabalho que não estejam envolvidas em alguma equivalência (b_1, b_2, b_3, c_1, c_2, c_3...).
- Para cada um destes itens percorridos, compara-se seu nome com o nome de a_1. Esta comparação é realizada seguindo os critérios explicados anteriormente. Caso a comparação indique que os dois elementos arquiteturais são equivalentes, e ainda não exista um candidato à equivalência ligado a esses elementos, é criado então ce_1 – candidato à equivalência 1. Esse candidato recebe o nome de um dos elementos comparados. É importante perceber que o nome da equivalência é usado apenas para identificá-lo unicamente, permanecendo inalterados os nomes dos elementos arquiteturais conectados a ele.

- Assim que ce_1 é criado, o mesmo é conectado à a_1 e ao outro elemento que foi considerado equivalente.
- O processo de procura por equivalentes de a_1 continua e, ao final, tem-se ce_1 conectado a todos os elementos equivalentes à a_1.

  A Figura 4.3 abaixo ilustra a idéia:

Figura 4.10– Identificação de candidatos à equivalência

Seguindo o exemplo da Figura 4.10, a Arquitetura A foi percorrida, e foi encontrado o elemento "Jogos" como sendo a2. Assim, percorre-se os outros elementos da Arquitetura A, B e C, à procura de elementos com o nome "Jogos". Uma vez encontrados elementos arquiteturais equivalentes nas arquiteturas B e C, um candidato a equivalência com o nome "Jogos" é criado. Assim, no exemplo, ce_1 = "Jogos".

Uma vez detectado ce_1, é necessário saber ainda se ce_1 pode ou não ser considerado uma equivalência. Isto é determinado pelo critério "Número mínimo de arquiteturas equivalentes", determinado pelo engenheiro de domínio. Por exemplo, se este número for igual a 2, para que ce_1 seja considerado uma equivalência, seria necessário que ele ligasse elementos arquiteturais de pelo menos duas arquiteturas distintas.

Como já mencionado na seção 4.3, na busca por elementos que podem ser equivalentes a um dado elemento, o elemento a ser procurado não pode ser participante de uma outra equivalência. No entanto, seguindo-se apenas esta regra, elementos de diferentes tipos, como pacotes e classes, poderão ser considerados equivalentes, a menos que o engenheiro de domínio tenha determinado o contrário, na utilização do critério **Comparar elementos do mesmo tipo**, como explicado na seção 4.3.1

4.3.3 - Confirmação de Equivalências

Na descrição do algoritmo de detecção de equivalências, foi tratado o processo de identificação das mesmas, No entanto, as equivalências detectadas, são, na verdade, "*candidatos a* equivalências", isto é, após a detecção de tais candidatos, o engenheiro de

domínio deve confirmar a validade da equivalência. Dessa forma, é proposta uma interação maior do engenheiro no processo, onde cada equivalência deve ser confirmada antes de ser considerada válida. Além disso, a confirmação de uma equivalência pelo engenheiro pode possibilitar que este alimente, caso julgue necessário, o dicionário de sinônimos.

As equivalências identificadas nesta atividade serão usadas na fase de detecção de Variabilidades (seção 4.4), portanto, quanto mais preciso for o resultado desta etapa, melhor será o resultado na fase subseqüente.

4.3.4 - Detecção da Opcionalidades através das Equivalências identificadas

Uma vez identificadas as equivalências e, conseqüentemente, os elementos equivalentes do conjunto de trabalho*,* é previsto, nesta fase, que um elemento seja classificado quanto à sua existência na arquitetura de referência a ser criada, como um elemento mandatório ou opcional. Para tanto, o critério **Número mínimo de arquiteturas equivalentes** volta a ser considerado, indicando que um elemento é mandatório no domínio, quando ele estiver presente em um certo número de arquiteturas distintas, igual ou maior, ao critério estabelecido pelo engenheiro. Caso contrário, o elemento será considerado opcional.

Dessa forma, para se chegar à conclusão de que um elemento é mandatório ou opcional, deve-se analisar se o elemento está conectado a alguma equivalência. Em caso afirmativo, verifica-se se esta equivalência abrange elementos de pelo menos N arquiteturas, sendo N igual ou maior que o número especificado no critério **Número mínimo de arquiteturas equivalentes**. Caso abranja, será considerado mandatório, caso contrário será considerado opcional.

É importante ressaltar que, uma vez que a classificação de um elemento como mandatório ou opcional depende do critério **Número mínimo de arquiteturas equivalentes,** o fato de se identificar uma equivalência não significa que foi necessariamente identificado se o elemento é mandatório ou opcional. Isto porque, o engenheiro pode, por exemplo, julgar necessário ser mais criterioso, e decidir modificar o critério acima mencionado, determinando para ele um número maior de arquiteturas. Neste caso, a opcionalidade de cada elemento deverá também ser redefinida pelo engenheiro, em função das novas equivalências identificadas.

4.4 - Fase 2: Detecção das Variabilidades

Como mencionado anteriormente, cada fase da abordagem *ArchToDSSA* fornece subsídios para a fase seguinte. De posse das equivalências identificadas na Fase 1 da abordagem, dá-se início à segunda fase, i.e. a de Detecção das Variabilidades, como exemplificado na Figura 4.11.

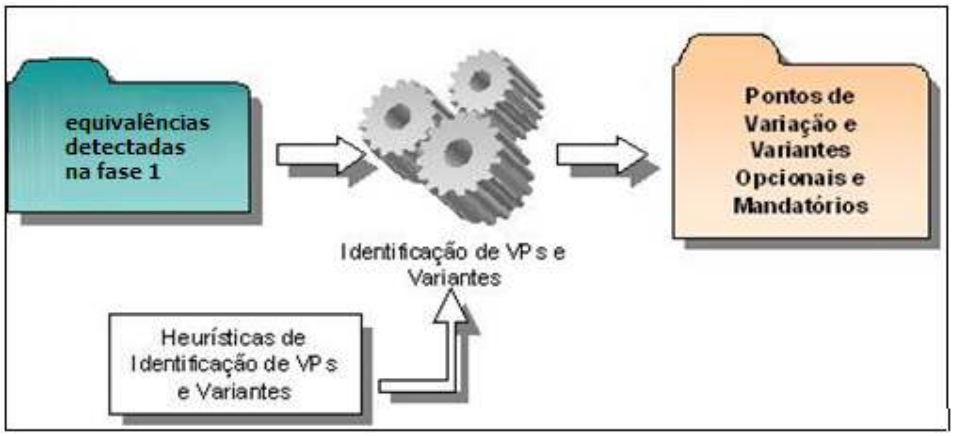

Figura 4.11 *ArchToDSSA* – Fase 2

O objetivo desta fase é identificar Pontos de Variação (*VP – Variation Points*) e suas respectivas variantes. Para tanto, é definido um algoritmo para essa identificação.

São utilizadas, na abordagem, arquiteturas descritas através de diagramas de pacotes e de classes da UML, conforme já mencionado. Levando-se em consideração um dos requisitos apresentados na seção 4.1, que menciona a necessidade da representação de uma arquitetura de referência de maneira clara quanto ao entendimento e identificação dos seus elementos, escolheu-se representar a arquitetura de referência no formato XMI (*XML Metadata Interchange).* Uma vez que, ainda de acordo com os requisitos da seção 4.1, seria necessário um ferramental de apoio, a escolha do XMI foi feita pensando-se na possibilidade de facilitar a intercomunicação entre diversas ferramentas, por meio desse formato de arquivo. Além disso, a UML e o XMI possibilitam a identificação dos relacionamentos entre as classes.

No trabalho de OLIVEIRA (2006), foi definido um mapeamento da representação da opcionalidade e variabilidade entre modelos de características e modelos de classe. Desse mapeamento, foram definidas heurísticas, obtidas através de estudos de caso, que estabelecem a relação de pontos de variação e suas variantes com interfaces e classes que participam de um relacionamento de herança, como apresentado

na Tabela 4.1. Com base nestas heurísticas, nesta fase da abordagem os relacionamentos de herança e implementação, especificamente, são de grande interesse para a descoberta de possíveis pontos de variação e suas respectivas variantes. Isto porque, se uma dada interface ou superclasse possui equivalência em um número mínimo de arquiteturas, ela pode ser considerada um ponto de variação, que pode estar presente na arquitetura de referência.

Tabela 4.1 Heurísticas de mapeamento de Variabilidades e Opcionalidades da notação Odyssey-FEX (Adaptado de (OLIVEIRA, 2006))

Elemento	Heurística
Ponto de Variação (VP)	Se existir, no modelo de características, um ponto de variação, pode existir no diagrama de classes uma classe ou interface que suporte tal elemento, utilizando-se do estereótipo <<vp>>.
Variante	Se existir, no modelo de características, um ponto de variação, pode existir no diagrama de classes uma classe ou interface que suporte tal elemento, utilizando-se do estereótipo <<vp>>.
Relacionamento entre VP e Variantes	Se existir, no modelo de características, um relacionamento do tipo ***Alternativo*** entre um ponto de variação e suas variantes, então, deve existir no diagrama de classes um relacionamento de **herança** entre as classes que suportam tais características, se o ponto de variação estiver mapeado para uma **classe**. No caso de o ponto de variação estar mapeado para uma **interface**, deve existir no diagrama de classes um relacionamento de **realização** entre as classes que suportam as variantes e a interface que suporta o ponto de variação.

Segundo a abordagem, à medida que interfaces ou superclasses são consideradas como pontos de variação, suas respectivas subclasses ou implementações definem possíveis configurações nestes pontos específicos e representam as variantes associadas a esses pontos específicos.

Naturalmente, esta afirmação não é válida para todos os casos, nem tão pouco significa que não possam ser encontrados pontos de variação e variantes em outras estruturas das arquiteturas presentes no conjunto de trabalho. No entanto, é um ponto de partida para a definição de um algoritmo capaz de sugerir pontos de variação e suas respectivas variantes.

4.4.1 - Detecção de Pontos de Variação (VPs) e suas Variantes

Assim como na identificação de equivalências, para nortear o processo de identificação de pontos de variação, foram definidos alguns critérios, descritos a seguir:

Critério 1: **Número mínimo de arquiteturas equivalentes para se considerar uma superclasse como um ponto de variação.** Indica em quantas arquiteturas uma dada superclasse deve possuir equivalência para que seja considerada como um ponto de variação.

Critério 2: Número mínimo de arquiteturas equivalentes para se considerar uma interface como um ponto de variação. Indica em quantas arquiteturas uma dada interface deve possuir equivalência para que seja considerada como um ponto de variação.

A seguir é descrito o algoritmo sugerido para a identificação de pontos de variação na abordagem *ArchToDSSA*.

4.4.1.1 - Descrição do algoritmo de Detecção de Pontos de Variação (VPs) e Variantes

Sejam A, B, e C arquiteturas que estão sendo comparadas, isto é, participantes do conjunto de trabalho. Sejam a_1, a_2, ...a_n; b_1, b_2, ...b_n; e c_1, c_2, ..., c_n elementos arquiteturais das arquiteturas A, B e C, respectivamente. Para a detecção de pontos de variação, deve-se proceder da seguinte forma:

- Em uma arquitetura A, escolhe-se a primeira superclasse ou interface disponível, isto é, uma superclasse ou interface que não seja ponto de variação nem equivalente a um elemento que já seja ponto de variação. Como é utilizada a notação *Odyssey-FEX*, o elemento escolhido também não poderá ser variante nem equivalente de um variante, uma vez que um elemento não pode, pela notação, ser simultaneamente VP e variante.
- Define-se este elemento como a_1 (primeiro elemento disponível da arquitetura A). Percorrem-se todos os outros elementos da arquitetura A (a_2, a_3,..., a_n) e também cada elemento das outras arquiteturas do conjunto de trabalho que sejam interfaces ou superclasses e estejam disponíveis (b_1, b_2, b_3, c_1, c_2, c_3...). Para cada um destes itens, verifica-se se existe equivalência com a_1. Convém ressaltar que tal informação está disponível como resultado da Fase 1.
- Se houver equivalência, então o total de arquiteturas que possuem elementos equivalentes a a_1 é contabilizado. Ao final das iterações, é possível identificar

em quantas arquiteturas distintas a referida superclasse ou interface possui equivalentes.

- Levando-se em consideração o tipo do elemento, isto é, uma interface ou superclasse, e os critérios **Número mínimo de arquiteturas equivalentes para se considerar uma superclasse como um ponto de variação** ou **Número mínimo de arquiteturas equivalentes para se considerar uma interface como um ponto de variação**, é possível determinar se o elemento vai ou não ser considerado um ponto de variação.
- Repete-se o processo para outros elementos disponíveis. Ao final, tem-se uma lista de pontos de variação e seus respectivos elementos (superclasses ou interfaces) equivalentes nas arquiteturas analisadas.
- Passa-se agora à detecção de variantes. Para tanto, percorre-se os elementos identificados como ponto de variação. Para cada interface identificada como VP, seleciona-se suas implementações como variantes associadas. Para cada superclasse identificada, seleciona-se suas subclasses como variantes associadas

A Figura 4.12 e Figura 4.13 ilustram, respectivamente, a idéia da detecção de VPs e de variantes.

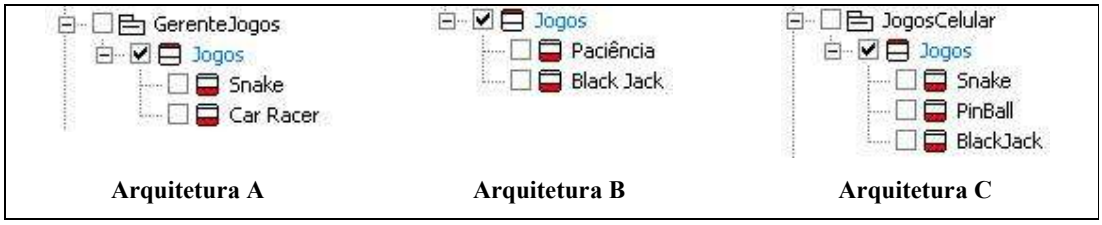

Figura 4.12 – Identificação de Pontos de Variação

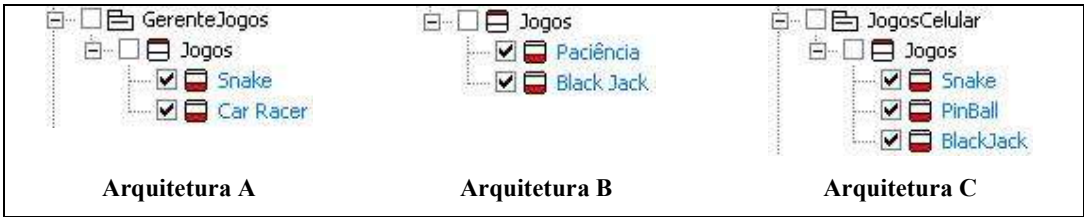

Figura 4.13 – Identificação de Variantes

No exemplo da Figura 4.12, a Arquitetura A foi percorrida e encontrada a superclasse "Jogos" como sendo a_2. Percorreu-se ainda as arquiteturas B e C,

encontrando em ambas a superclasse “Jogos” como sendo elementos equivalentes a a_2. Assim, no exemplo, é definido o elemento “Jogos” como um ponto de variação.

No exemplo da Figura 4.13, são percorridas as arquiteturas A, B e C em busca das superclasses que foram selecionadas como ponto de variação. Em cada arquitetura, e para cada ponto de variação, foram selecionadas suas subclasses como variantes. Assim, os elementos “Snake”, “Car Racer”, “Paciência”, “PinBall” e “Black Jack” são selecionados como variantes na arquitetura de referência.

Além de definir procedimentos e um algoritmo para esta atividade, é permitido que o engenheiro de domínio defina os pontos de variação como melhor lhe convier, não fazendo restrições quanto às suas escolhas. Na prática, isto significa que o engenheiro pode determinar os pontos de variação com base unicamente no seu conhecimento do domínio, sem seguir qualquer critério sugerido pela abordagem proposta. Isto se deve ao fato de que o principal objetivo da abordagem é auxiliar e em alguns casos sugerir ao engenheiro opções para a criação de uma arquitetura de referência, e não impor restrições ao seu trabalho.

4.5 - Fase 3: Criação de uma Arquitetura de Referência

A terceira e última fase da abordagem tem como objetivo selecionar elementos para a criação de uma arquitetura de referência de domínio (DSSA).

Ao final das duas primeiras fases da abordagem *ArchToDSSA*, tem-se como resultado informações sobre que elementos arquiteturais, pertencentes às arquiteturas do conjunto de trabalho, são relevantes para esta fase de criação. A Fase 3 é apresentada pela Figura 4.14, a seguir.

Na Fase 1, foram identificadas as equivalências e, conseqüentemente, as Opcionalidades dos elementos no domínio. Na Fase 2, as equivalências identificadas na Fase 1 foram utilizadas para a determinação da Variabilidades dos elementos. É importante ressaltar a interação do engenheiro em ambas as fases, de forma que, ao entrar na fase de criação, os elementos identificados, bem como suas Opcionalidades e Variabilidades, estejam de acordo com seu entendimento sobre o domínio.

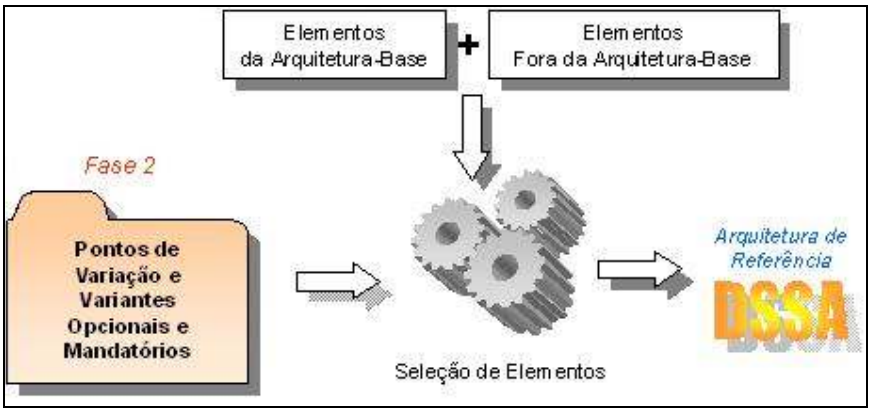

Figura 4.14 - *ArchToDSSA* – Fase 3

Na Fase 3, os elementos selecionados são, como já mencionado, os elementos que vão compor a arquitetura de referência sugerida. Porém, não só os elementos são selecionados, mas também os relacionamentos entre eles. Para evitar a existência de relacionamentos incompletos na arquitetura de referência, isto é, a inexistência de algum dos elementos envolvidos em uma relação, propõe-se a determinação de uma *arquitetura-base*. A arquitetura-base nada mais é do que uma das arquiteturas presentes no conjunto de trabalho que, uma vez escolhida pelo engenheiro, terá todos os seus elementos e relacionamentos incondicionalmente presentes na arquitetura de referência.

Uma vez selecionada a arquitetura-base, a idéia é que o engenheiro possa ter fácil acesso a cada elemento arquitetural das outras arquiteturas do conjunto de trabalho, principalmente aos identificados na fase anterior. Tal acesso deve permitir que se selecione, ou se deixe de selecionar, elementos para a criação da arquitetura de referência (elementos da arquitetura-base devem estar permanentemente selecionados) bem como a identificação imediata de cada propriedade identificada nas fases anteriores. No caso da Opcionalidade, deve ser possível ao engenheiro modificar a propriedade mandatório/opcional, caso necessário.

Os elementos de outras arquiteturas, que não são a arquitetura-base, quando selecionados, não carregam seus relacionamentos, cabendo ao engenheiro especificar os relacionamentos existentes entre esses elementos na arquitetura de referência. A exceção a essa regra são os elementos que foram classificados como variantes, na Fase 2, que devem ser incluídos em seus respectivos ponto de variação, quando estes são selecionados para compor a arquitetura de referência. Todos elementos que não fazem parte da arquitetura-base são classificados como “Não-Definidos”.

Depois de selecionados os elementos arquiteturais e seus relacionamentos, é importante que a arquitetura de referência gerada seja acessível de alguma forma ao engenheiro de domínio. Optou-se por sua representação no formato XMI, visando permitir que a arquitetura de referência criada possa ser manipulada por ferramentas externas.

Para auxiliar o engenheiro na atividade de seleção de elementos, são previstos alguns critérios, como descritos na próxima seção.

4.5.1 - Critérios de Seleção de Elementos para a Arquitetura de Referência

Alguns critérios devem ser observados pelo engenheiro, quando da seleção dos elementos:

- Caso o elemento selecionado já possua um elemento equivalente na arquitetura-base, o engenheiro não deve incluí-lo na arquitetura de referência uma vez que já possui equivalente na mesma;
- Caso o elemento esteja contido em algum *espaço de nomes*[6] (i. e. elemento que contém outro elemento, ex: pacote contendo classes) também marcado para seleção, então, o mesmo pertencerá ao mesmo elemento na arquitetura de referência;
- Caso o elemento não esteja contido em um *espaço de nomes* selecionados, mas este espaço de nomes possua um equivalente que esteja sendo selecionado, o referido elemento passará a ser contido pelo *espaço de nomes* equivalente;
- Caso o elemento não esteja contido em *espaço de nomes* selecionado, nem exista equivalente ao referido *espaço de nomes*, o elemento deverá estar em um pacote situado na raiz da arquitetura de referência;
- Caso a relação entre o *espaço de nomes* e o elemento nele contido seja do tipo herança ou implementação, esta mesma relação deverá ser mantida na arquitetura de referência;
- As variantes das arquiteturas não pertencentes à arquitetura-base, identificadas na Fase 2, devem ser selecionadas. Estas variantes deverão ser inseridas no ponto de variação equivalente ao seu ponto de variação na arquitetura-base e deverão manter o mesmo tipo de relacionamento existente com o seu ponto de variação (i.e. herança ou implementação).

[6]Do inglês *Namespace*, conforme definido na UML (OMG, 2001).

- Todo elemento que não pertença à arquitetura-base é classificado como “não definido”, os demais são classificados como “definidos”.
- Na representação da arquitetura de referência criada, no formato XMI, as propriedades de Opcionalidade e Variabilidade são expressas por meio de etiquetas, que possuem um nome e um valor (i.e., *tagged values*), da seguinte forma:
  - Opcionalidades (determinada por meio de equivalências identificadas na Fase 1):
    - nome = "Opcionalidade";
    - valor = "*opcional*" ou "*mandatório*"
  - Variabilidades (definidas na Fase 2):
    - nome = "Variabilidade";
    - valor = "*Invariante*" ou "*Variante*" ou "*Ponto de Variação"*
  - Elementos não pertencentes à arquitetura-base (definidos na Fase 3):
    - nome = "*Não Definido"*;
    - valor = "*verdadeiro*" ou "*falso*"

4.6 - Considerações Finais sobre a Abordagem *ArchToDSSA*

Este capítulo apresentou a abordagem *ArchToDSSA*, cujo objetivo é apoiar a criação de arquiteturas de referência em um determinado domínio. Foram apresentadas as três fases da abordagem, que utiliza os conceitos da notação Odyssey-FEX (OLIVEIRA, 2006), identificando opcionalidades e variabilidades.

Na primeira fase da abordagem, foi visto que detecção das Opcionalidades pode ser obtida através da identificação de equivalências e da análise da abrangência das mesmas nas arquiteturas do conjunto de trabalho. Os resultados obtidos nesta fase são usados nas fases seguintes.

Esta fase pode ainda ser incrementada através da modificação ou adoção de novos critérios para o algoritmo de detecção de equivalências. Como exemplo, poderia ser proposta a definição do percentual de arquiteturas em que um elemento deve estar como equivalente para ser considerado mandatório.

Outra característica interessante que pode ser introduzida no algoritmo é o cálculo de equivalência entre elementos, tomando-se como base elementos relacionados (vizinhança). Por exemplo, verificar se um pacote é equivalente a outro pacote com base

no percentual de equivalência entre suas respectivas classes. Neste último caso, critérios de pesos e proporcionalidade de equivalência poderiam ser estabelecidos.

Na Fase 2, a qualidade da detecção da Variabilidades, conforme observado, depende muito da precisão com que a identificação das equivalências e, conseqüentemente, das Opcionalidades é realizada.

O conjunto de heurísticas estabelecidos na fase 2 (e em todas as fases da abordagem) não pretende ser completo. O conjunto inicial de heurísticas é derivado da notação Odyssey-FEX, porém, diferentes heurísticas podem ser incorporadas à medida que estudos forem realizados e o devido *feedbac*k for obtido, por meio de interações com o engenheiro.

Na Fase 3, apesar do processo de seleção ser flexível, é interessante que o engenheiro manipule os elementos selecionados em algum ambiente de modelagem antes de se obter a arquitetura de referência final. Isto porque, nesta definição inicial, a abordagem não contempla a seleção de relacionamentos entre elementos não pertencentes à arquitetura-base, e nem sugere a remoção de elementos da arquitetura-base, deixando a cargo do engenheiro tais decisões. Isto se fez necessário para que não existam relacionamentos incompletos na arquitetura de referência.

Assim como em um *framework*, onde uma arquitetura básica é estabelecida e componentes vão sendo introduzidos e aprimorados em sua arquitetura, foi estabelecida uma seqüência de fases, com o objetivo de direcionar o engenheiro na criação de uma arquitetura de referência. Uma vez que, em todas as fases, todas as atividades sugeridas requerem uma interação do engenheiro, pode-se inferir que tal interação acaba por exercer grande influência sobre os resultados obtidos. Estas fases podem sofrer evoluções na medida em que estudos são realizados, a abordagem é utilizada, e novas heurísticas são consideradas em cada fase. É um trabalho extenso e iterativo. Ao longo das evoluções, o processo vai ganhando maturidade.

Muito embora exista espaço para evoluções, a abordagem trata os problemas levantados, que não foram resolvidos pelas abordagens estudadas no capítulo anterior. Ela trata, por exemplo, da comparação de arquiteturas de sistemas legados, onde elementos sintaticamente diferentes representam o mesmo conceito, introduzindo para isto conceitos com dicionário de sinônimos e lista de palavras ignoradas. Além disto, define um método claro para identificação da variabilidade e apoiar o engenheiro na seleção de elementos que deverão compor a DSSA.

Ainda no contexto desta dissertação, foi realizada a implementação de uma ferramenta de apoio à abordagem, denominada ArchToDSSATool, conforme descrito no próximo capítulo.

Capítulo 5: ArchToDSSATool: Uma Ferramenta de Apoio para a Abordagem ArchToDSSA

No Capítulo 4 foi apresentada a abordagem *ArchToDSSA*, cujo objetivo é apoiar o engenheiro de domínio na criação de arquiteturas de referência a partir de arquiteturas já existentes, como as de sistemas legados.

A ferramenta *ArchToDSSATool* foi implementada no sentido de automatizar o apoio à abordagem proposta, viabilizando a sua prática, em sistemas existentes, como os sistemas legados de um domínio. Ela foi desenvolvida no contexto do projeto Odyssey (ODYSSEY, 2007), que visa o desenvolvimento do ambiente de reutilização de mesmo nome. A ferramenta *ArchToDSSATool* pode ser utilizada de forma isolada ou integrada ao ambiente Odyssey, sendo que, neste último caso, os resultados gerados pela mesma contribuem ao apoio à engenharia de domínio atualmente provido pelo ambiente. A *ArchToDSSATool* é apresentada neste capítulo.

Assim, partindo desta introdução, o restante do capítulo está organizado da seguinte forma: a seção 5.1 dá uma idéia geral do ambiente de reutilização Odyssey, ao qual a ferramenta *ArchToDSSATool* pode ser integrada; a seção 5.2 descreve a arquitetura da ferramenta desenvolvida; a seção 5.3 explica sua utilização; e as considerações finais são apresentadas na seção 5.4.

5.1 - O ambiente Odyssey

O ambiente Odyssey (ODYSSEY, 2007) representa uma infra-estrutura de reutilização baseada em modelos de domínio que engloba tanto o apoio à linha de produtos (LPS) e engenharia de domínio (ED) quanto à engenharia de aplicação (EA).

O Odyssey apresenta uma série de ferramentas e características que podem apoiar a utilização da abordagem proposta, como por exemplo as ferramentas de modelagem, que envolvem tanto os modelos da UML, necessários para contemplar a modelagem da arquitetura de referência gerada pela abordagem *ArchToDSSA*, bem como a implementação da notação Odyssey-FEX (OLIVEIRA, 2006), que suporta os atributos de opcionalidade e variabilidade no domínio.

Além disso, outras funcionalidades do ambiente Odyssey podem ser acessadas por meio de *plugins*, i.e., ferramentas que podem ser acopladas ao ambiente através de um mecanismo de carga dinâmica, como descrito em (MURTA *et al.*, 2004). Nesse

mecanismo, uma lista de *plugins* fica armazenado no servidor da COPPE, e as ferramentas são baixadas e acopladas ao ambiente sob demanda.

Muito embora possa funcionar como um *plugin* do ambiente Odyssey, a ferramenta *ArchToDSSATool* também pode ser executada de forma independente. As arquiteturas de referência criadas à partir da ferramenta, são exportadas para arquivos no formato XMI. Mecanismos internos detectam automaticamente caso a ferramenta tenha sido executada como *plugin*. Neste caso, é possível uma maior interação com o ambiente Odyssey e opcionalmente pode-se optar por visualizar os elementos exportados diretamente no ambiente de modelagem do Odyssey.

Conforme já mencionado na seção 4.1, a ferramenta *ArchToDSSATool* se insere na proposta mais ampla descrita em (VASCONCELOS, 2007), servindo de apoio ao processo proposto, cuja ênfase é a recuperação de arquiteturas baseada em engenharia reversa dinâmica. No entanto, é implementada de maneira a permitir a avaliação de arquiteturas de forma independente da proposta de VASCONCELOS (2007).

Detalhes da implementação da ferramenta são explorados nas próximas seções.

5.2 - Arquitetura da Ferramenta *ArchToDSSATool*

Seguindo a divisão da abordagem em fases, a ferramenta, em sua arquitetura, também foi dividida em pacotes distintos. Uma visão geral da sua arquitetura é apresentada na Figura 5.15

A ferramenta é representada pelo pacote “Phases”, que contém três pacotes principais, representando as três fases: “Matches”, “VP” e “Export”. Cada um destes pacotes contém classes que implementam as funcionalidades de cada fase. Em cada pacote, classes foram nomeadas com os prefixos “BUS” e “GUI”. O termo “BUS” indica que a classe está relacionada com as regras de negócio (*business*) explicadas no capítulo anterior (regras de comparação...), enquanto o termo “GUI” (Graphical User Interface) representa a interface gráfica com a qual o usuário interage. Esse artifício foi utilizado no intuito de facilitar a identificação das implementações relacionadas com cada tipo de funcionalidade e interação.

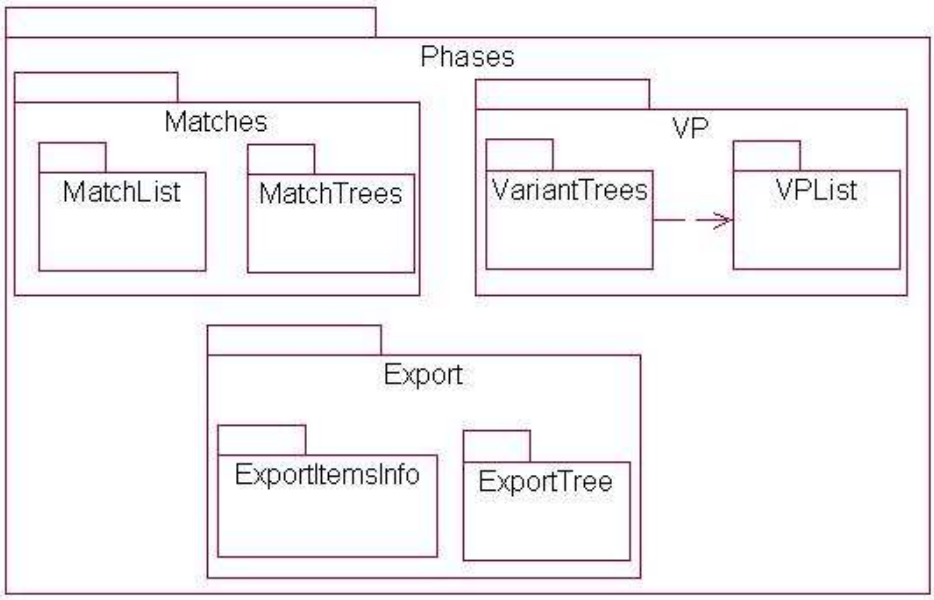

Figura 5.15 – Arquitetura da *ArchToDSSATool*

A Figura 5.16 apresenta o pacote "Matches". O pacote "Matches" contém classes que contemplam toda a funcionalidade relacionada à detecção de equivalências. Na interface gráfica da ferramenta, foi efetuada uma divisão em duas grandes áreas: uma lista com as equivalências sugeridas e árvores para exibir as arquiteturas analisadas. De acordo com esta divisão de interface, foram criados dois conjuntos de classes dentro do pacote, as classes "MatchLists" e as classes "MatchTrees". O primeiro conjunto é usado na manipulação da lista de equivalências e o segundo trata das arquiteturas exibidas em forma de árvore.

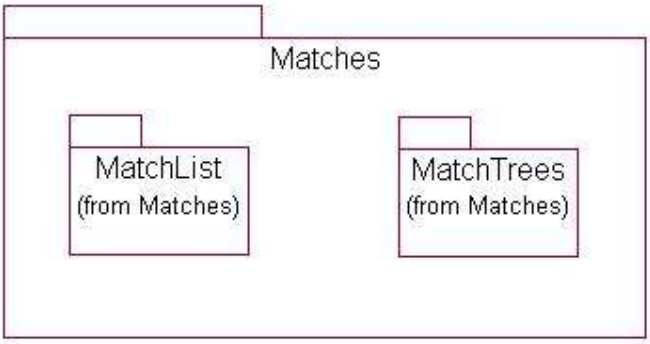

Figura 5.16 - Pacote Matches

Dentro do conjunto de classes MatchList, temos a classe BUSMatchList, e a classe GUIPnlMatchList. O pacote MatchList é apresentado na Figura 5.17.

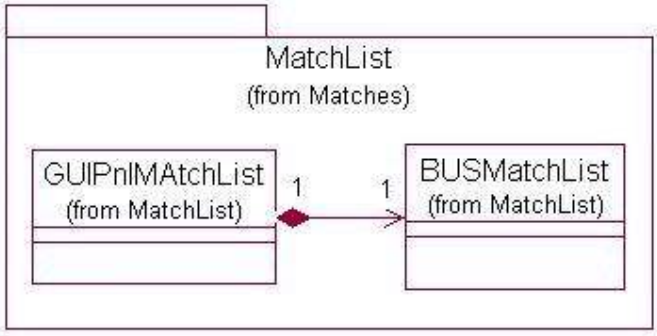

Figura 5.17 – Pacote MatchList

O conjunto "MatchTrees", apresentado na Figura 5.18, possui, analogamente, duas classes: BUSMAtchTrees, que trata das regras de negócio deste conjunto (regras de comparação, alimentação do dicionário,...), e GUIPnlMatchTree que trata da interface e interação com o usuário. As duas classes implementam funcionalidades que permitem que o usuário selecione manualmente elementos arquiteturais equivalentes, a detecção automática de equivalências, dentre outras atividades.

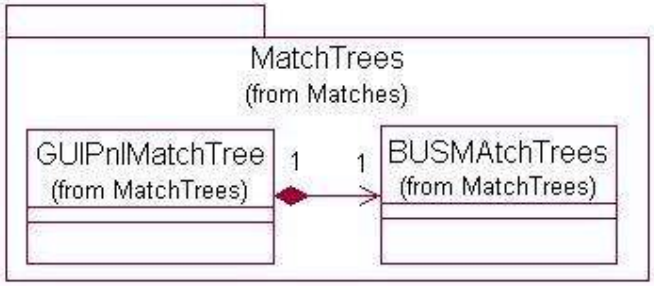

Figura 5.18 – Pacote MatchTrees

O segundo pacote, "VP", segue a mesma lógica de divisão existente no pacote "Matches", ou seja, a interface também foi dividida em duas grandes áreas: a lista de pontos de variação (VPs) e as árvores contendo elementos arquiteturais candidatos a variantes do VP selecionado. As classes deste pacote também foram divididas em dois conjuntos "VPList" e "VariantTrees", como mostra a Figura 5.19.

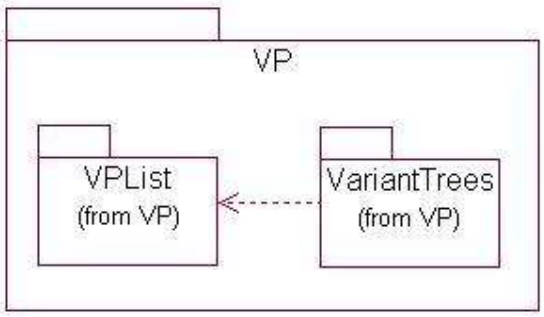

Figura 5.19 – Pacote VP

O conjunto VPList, mostrado na Figura 5.20, é representado pelas classes BUSVPList e GUIPnlVPList. A classe GUIPnlVPList trata das funcionalidades referentes a interação com o usuário no que diz respeito à lista de pontos de variação e a BUSVPList implementa a maioria das regras de negócio envolvidas nesta interação (regras para a detecção de pontos de variação, variantes, etc, conforme explicado no capítulo anterior).

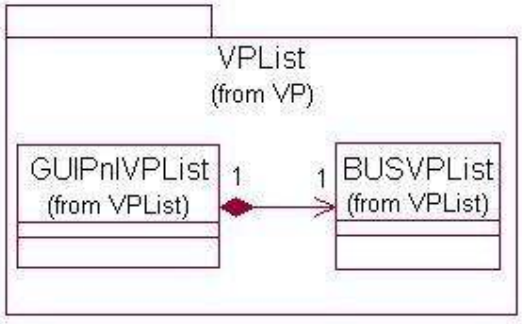

Figura 5.20 – Pacote VPList

No segundo conjunto de classes do pacote, o conjunto "VariantTrees", mostrado na Figura 5.21, a classe GUIPnlVariantTrees trata das interações do usuário no que diz respeito a seleção de variantes, enquanto a classe BUSVariantTrees implementa as suas respectivas regras de negócio. Neste pacote existem, ainda, duas classes auxiliares: BUSCheckVPListItem e BUSCheckVPListItems que são usadas em BUSVPList para permitir a seleção e confirmação de pontos de variação (VPs).

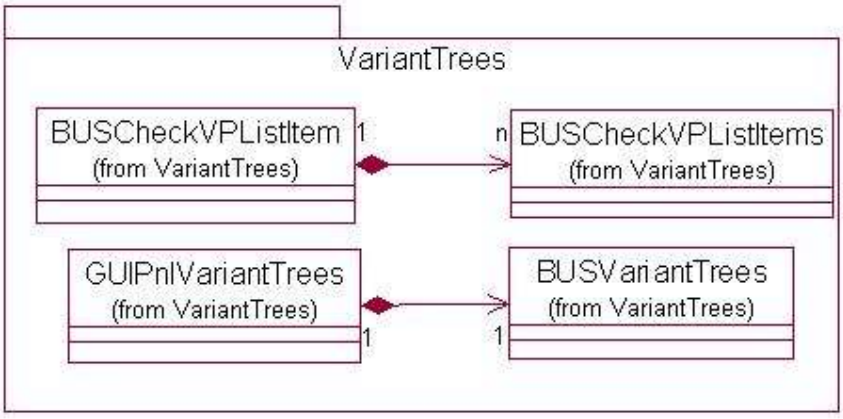

Figura 5.21 – Pacote VariantTrees

O terceiro pacote, Export, representando a terceira e última fase da abordagem, da mesma forma que nas outras fases, contém dois conjuntos de classes que se referem às duas principais partes da interface desta fase, como mostra a Figura 5.22.

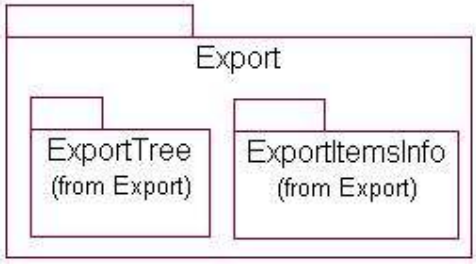

Figura 5.22 - Pacote Export

No primeiro conjunto, o "ExportTree', a classe GUIPnlExportTrees trata da iteração com o usuário, no que diz respeito à seleção da arquitetura-base, bem como à seleção de elementos arquiteturais fora da arquitetura-base, que deverão compor a arquitetura de referência. A classe BUSExportTrees trata das regras de negócios definidas no capítulo anterior, que dizem respeito a seleção de itens para a composição da arquitetura de referência, como mostra a Figura 5.23.

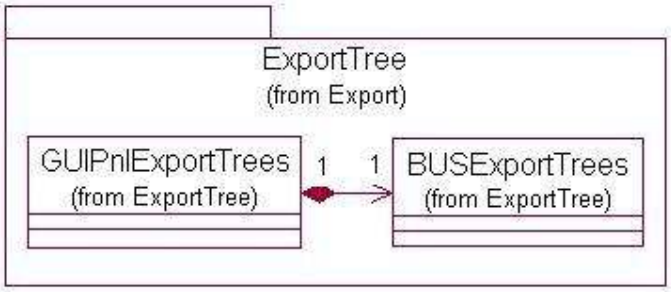

Figura 5.23 - Pacote ExportTree

No segundo conjunto, o "ExportItemsInfo", mostrado na Figura 5.24, a classe GUIPnlExportItemsInfo implementa funcionalidades que permitem ao usuário visualizar e/ou redefinir a opcionalidade de cada elemento arquitetural, e visualizar a variabilidade detectada na Fase 2. As classes BUSExportItemsInfo e BUSExportItemInfo auxiliam GUIPnlExportItemsInfo em suas atribuições. Nelas são encontrados por exemplos os algoritmos que exportam os elementos que irão compor a arquitetura de referência para o formato XMI ou diretamente dentro do Odyssey.

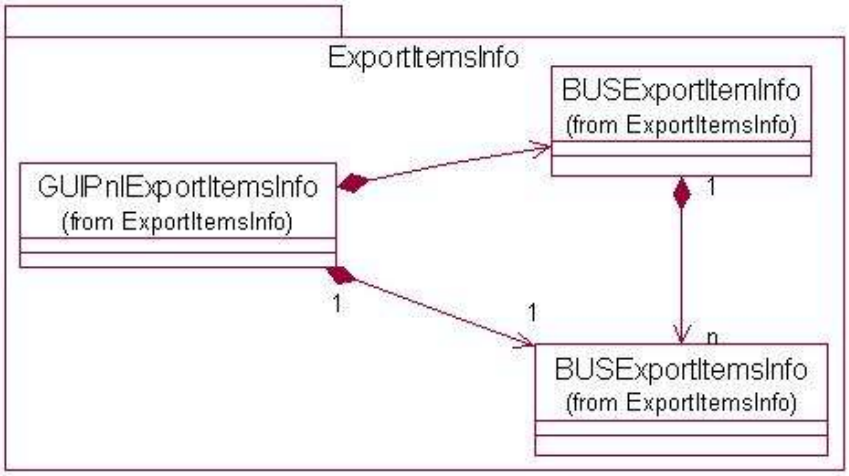

Figura 5.24 – Pacote ExportItemsInfo

Além dos pacotes principais, envolvendo as fases da abordagem, foram criados outros pacotes contendo classes e estruturas auxiliares, como as classes "GuiSupport", "Components" e "Utils", que permitem o encapsulamento de determinadas tecnologias e a reutilização de componentes básicos e de interface. O pacote denominado "Componentes", por exemplo, define uma série de componentes usados na interface, tais como listas com checkbox, árvores com checkbox, grupo de radiobuttons, etc.

O pacote “GUISupport”, por sua vez, implementa partes da interface de uso geral, que não necessariamente estão atreladas a uma determinada fase. Como exemplo, tem-se as telas de configuração dos parâmetros da ferramenta, o dicionário de sinônimos e a lista de palavras a serem ignoradas. As classes deste pacote são de uso menos genérico que as classes encontradas no pacote “Componentes” e, muito embora sejam utilizados de uma forma geral na aplicação, estão voltadas especificamente para atender às funcionalidades desta ferramenta específica.

Finalmente, no pacote “Utils”, foram implementadas classes que representam estruturas de dados, suporte à manipulação genérica de janelas da interface e o encapsulamento de repositórios MOF.

A ferramenta *ArchToDSSATool* possui ainda uma classe que representa seu “ponto de entrada”, isto é, a classe que permite o acesso às funcionalidades da ferramenta, que é a “*ArchToDSSA*Facade”. Esta classe contém uma instância da classe “FormMain”, que, por sua vez, contém as instâncias das classes contidas nos pacotes que implementam as três fases. A classe “FormMain” representa a interface principal da ferramenta. Além desta, existe a classe “FormStandAlone”, que é usada apenas para instanciar “*ArchToDSSA*Facade” e permitir que a ferramenta seja executada de forma independente do ambiente Odyssey. A Figura 5.25 ilustra a idéia:

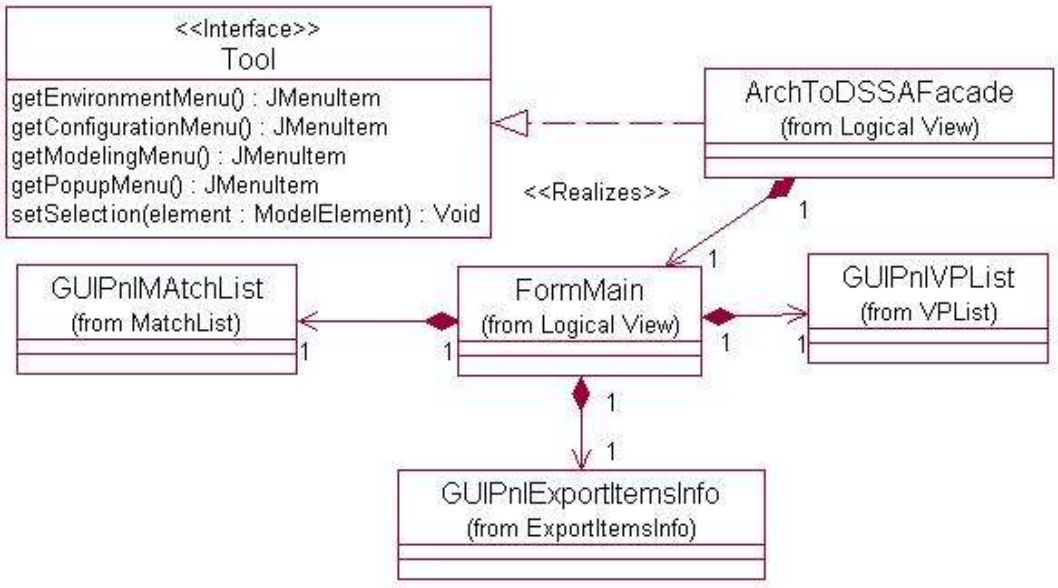

Figura 5.25 – Ponto de entrada da arquitetura da *ArchToDSSATool*

A integração com o Odyssey é feita através da classe *ArchToDSSATool*. Uma vez acionada, esta classe cria uma instância de "*ArchToDSSA*Facade" para executar a ferramenta como um plugin do Odyssey. A exemplo dos outros plugins do Odyssey, a classe *ArchToDSSA*Facade implementa a interface Tool, apresentada também na Figura 5.25, a fim de que suas opções de menu sejam disponibilizadas no Odyssey. Essa interface especifica serviços que disponibilizam opções de menu e menu popup, por exemplo, além de fornecer os serviços para o acesso do Odyssey às ferramentas baixadas.

A interface gráfica, bem como o processo de utilização da abordagem através da ferramenta *ArchToDSSATool*, são apresentados a seguir.

5.3 - Utilização da *ArchToDSSATool*

No capítulo 4 foi apresentado como requisito para uma abordagem de apoio à criação de uma arquitetura de referência um apoio ferramental, a fim de viabilizar sua prática. Como requisito desejável para este ferramental, pode-se destacar a possibilidade de se ter uma ferramenta que seja executada integrada a um ambiente de modelagem, porém não dependente desta.

Nesse sentido, a ferramenta *ArchToDSSATool* pode ser acessada das duas formas: internamente ao ambiente Odyssey, ou de forma independente. Sua execução de forma "*stand alone*", isto é, independente do ambiente Odyssey, se dá através de um arquivo executável, como mostra a Figura 5.26 a seguir.

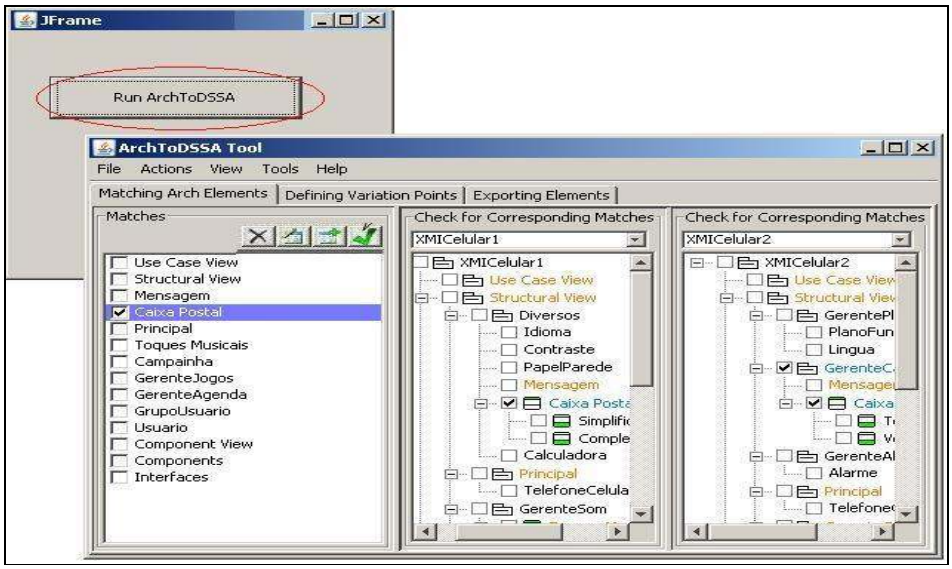

Figura 5.26 - Acesso à ferramenta *ArchToDSSATool* através de arquivo executável

Para a sua execução integrada ao ambiente, deve-se executar o Odyssey e acessar o menu *File* ▸ New *Domain* Na tela de modelagem de domínio, *Model Environment*, do Odyssey, tem-se acesso à ferramenta através do Menu *Tools* ▸ *ArchToDSSATool*, como mostra a Figura 5.27 abaixo.

Como pode ser visto na Figura 5.27, o ambiente de modelagem do Odyssey possui, do lado esquerdo da tela, as diversas visões complementares pelas quais um domínio pode ser modelado. Cada elemento criado na árvore da esquerda apresenta suas características detalhadas no lado direito da tela. Na Figura 5.27 é mostrado, do lado direito da tela, parte do diagrama de classes do domínio de Telefonia Móvel.

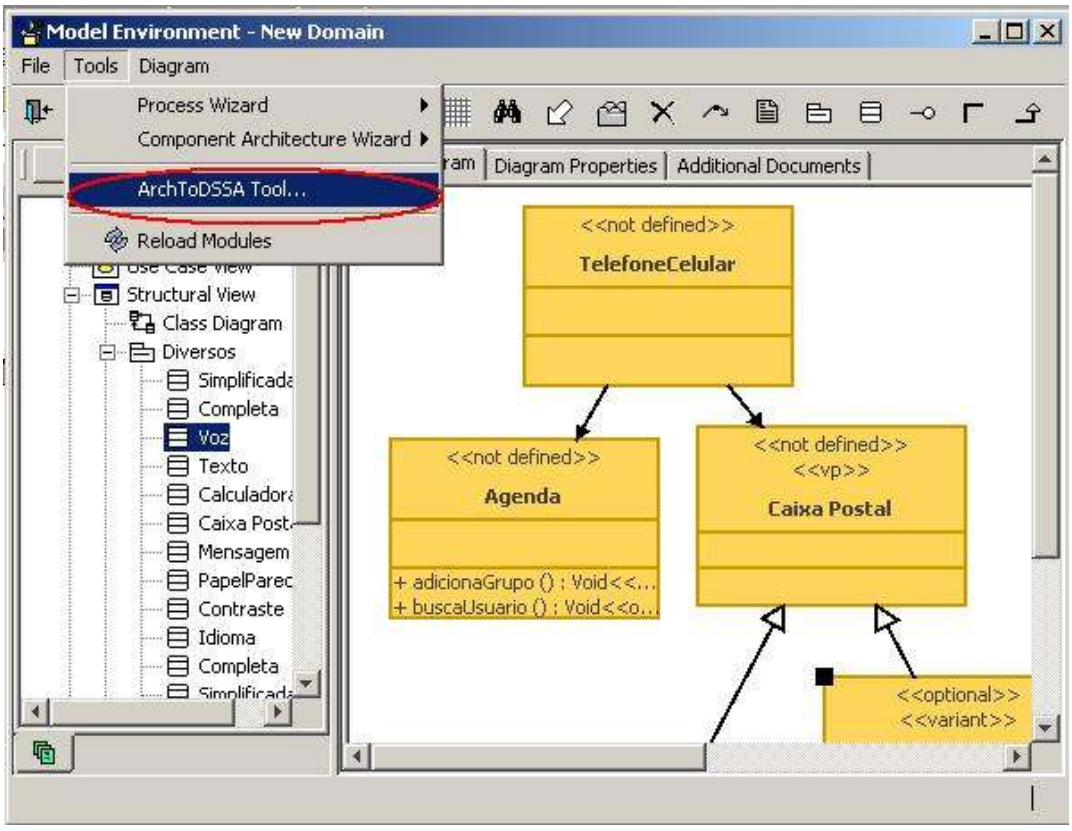

Figura 5.27– Acesso à ferramenta *ArchToDSSATool* através do Odyssey

Ao ser acionada, pode-se observar as três fases da ferramenta, representadas por abas, como ilustra a Figura 5.28

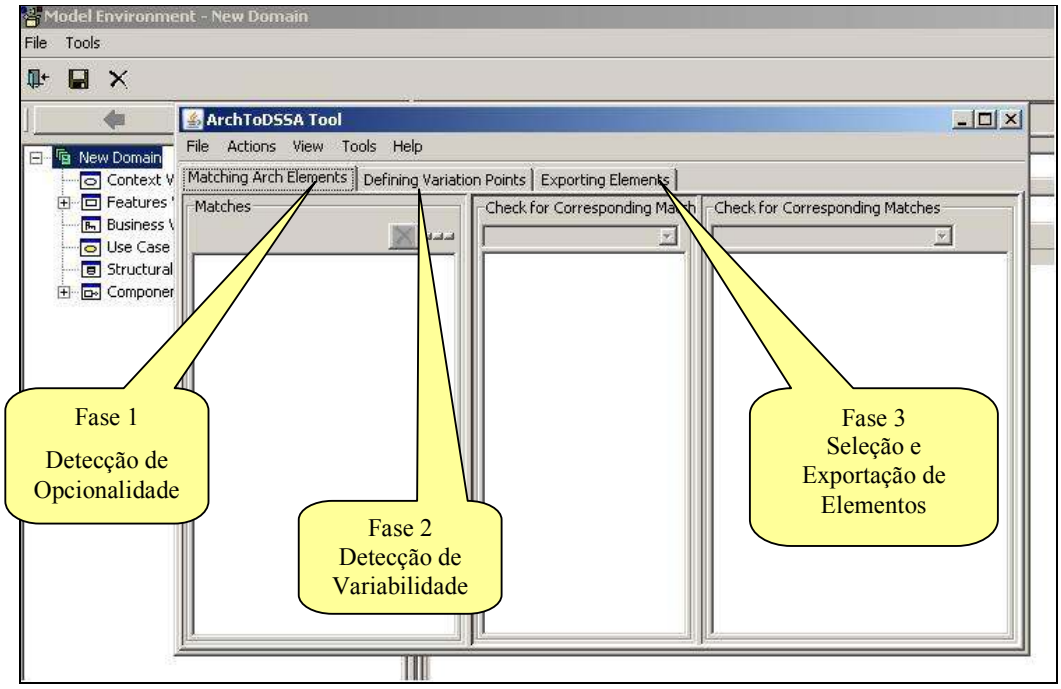

Figura 5.28 - Fases da abordagem *ArchToDSSA*

Assume-se, ainda, que as arquiteturas existam em arquivos no formato XMI e que elas sejam carregadas para o funcionamento da ferramenta.

Dessa maneira, o primeiro passo é carregar as arquiteturas que se deseja analisar, isto é, montar o conjunto de trabalho. Isso pode ser feito através do menu *File* ▸ *Select Architectures*, como mostra a Figura 5.29. A tela de seleção permite que sejam escolhidas arquiteturas existentes, no formato XMI, para que sejam comparadas na ferramenta.

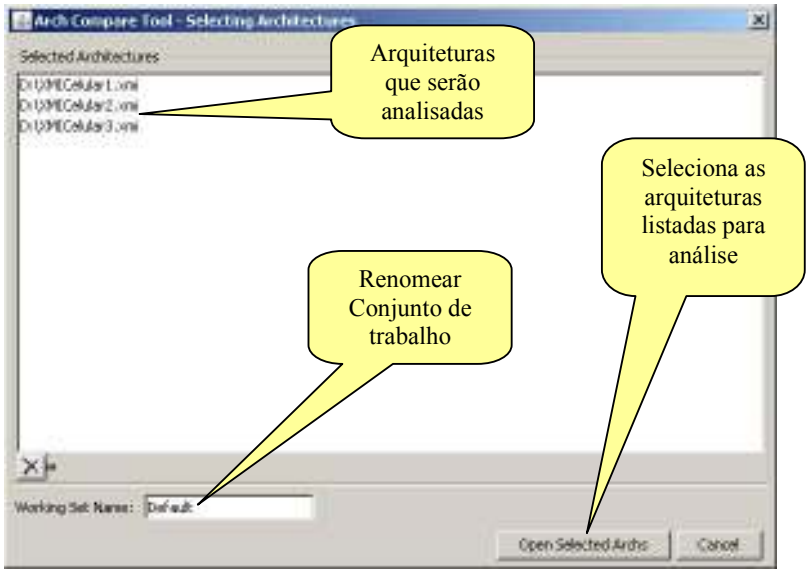

Figura 5.29 - Definição do Conjunto de trabalho

Uma vez selecionadas as arquiteturas, é iniciada a execução de cada fase prevista na abordagem, como detalhado nas subseções a seguir.

5.3.1 - Primeira Fase

A aba *Matching Arch Elements* da tela principal da ferramenta *ArchToDSSATool* representa a primeira fase da abordagem, como mostra a Figura 5.30.

As arquiteturas selecionadas e seus respectivos elementos arquiteturais são apresentados no formato de árvore. Uma das configurações da ferramenta é permitir que as arquiteturas sejam exibidas duas a duas, lado a lado, no intuito de facilitar a comparação entre seus elementos, ou, caso o engenheiro prefira, as arquiteturas podem ser exibidas em uma coluna única. Tal configuração pode ser acessada pelo menu *View* ► *Archs in Two Columns*. Todas as arquiteturas ficam disponíveis em cada uma das colunas para seleção.

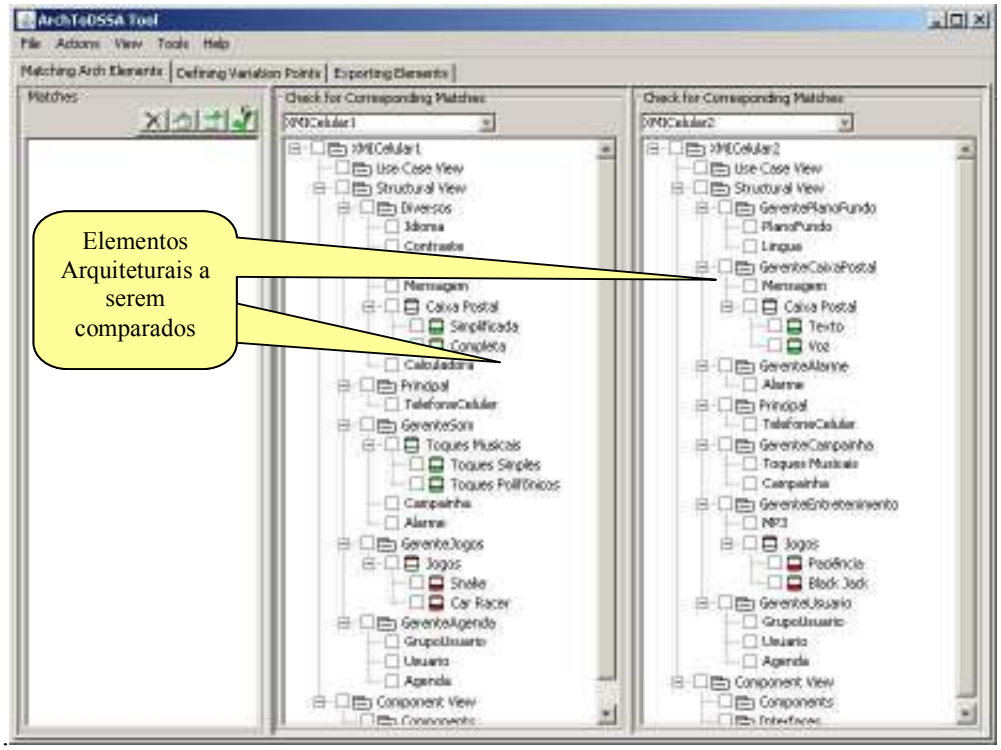

Figura 5.30 - *ArchToDSSA* - Primeira Fase

Conforme descrito no capítulo anterior, a primeira atividade a ser realizada é a identificação e criação de equivalências *(matches)*. Uma vez que o objetivo da ferramenta é oferecer apoio automatizado ao engenheiro de domínio, o algoritmo de detecção de equivalências foi implementado de forma a sugerir tais equivalências ao

engenheiro automaticamente. Essa geração automática leva em consideração os critérios citados no capítulo anterior, como o uso de dicionário de sinônimos e da lista de palavras ignoradas, bem como o critério para comparação de nomes através da divisão em partes, comparação de elementos do mesmo tipo e número mínimo de arquiteturas equivalentes, conforme descrito na seção 4.2.1. Tais critérios podem ser configurados acessando-se o menu *Tools* ▸ *Options* , como mostra a Figura 5.31.

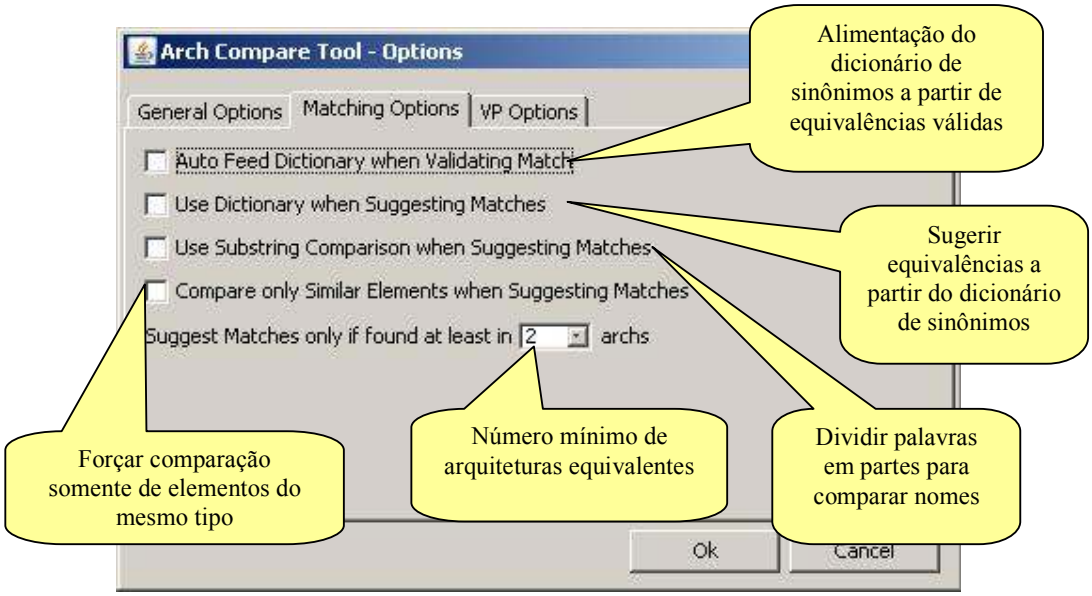

Figura 5.31 - Configuração de critérios para detecção de equivalências

Acessando o menu *Actions* ▸*Generate All Suggested Matches*, a lista de equivalências candidatas é gerada, conforme o algoritmo sugerido na seção 4.2.2, como mostra a Figura 5.32.

É importante ressaltar que os menus da *ArchToDSSATool* são sensíveis ao contexto, isto é, nesta fase do processo, o menu *Actions* reflete a ação da geração automática de equivalências, porém, o menu *Actions* pode refletir outros tipos de ações, em função da fase em que o usuário se encontra.

Embora automatize a detecção das equivalências, a ferramenta permite que o engenheiro defina uma equivalência manualmente. Os passos são os seguintes:

1. O engenheiro deve primeiro criar uma equivalência, acessando o botão  - *Adicionar novo match* - na barra de botões;
2. Deve então escolher um elemento de uma arquitetura e buscar por elementos equivalentes ao elemento escolhido. Os elementos encontrados devem ser

selecionados através da caixa de seleção do item (*checkbox* do item na árvore);

3. Repete-se este procedimento até que todas as *equivalências* sejam identificadas. A diferença é que os critérios a serem seguidos para determinar se um elemento é ou não equivalente a outro são determinados exclusivamente pelo engenheiro.

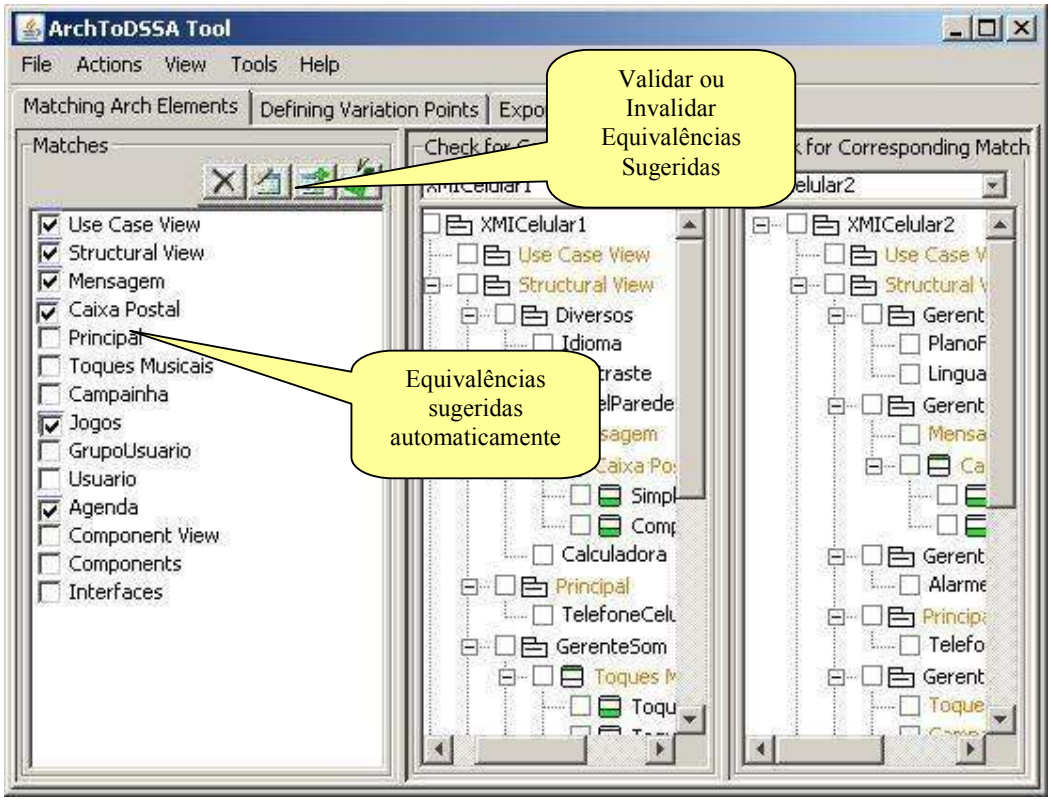

Figura 5.32 - Equivalências geradas automaticamente pela ferramenta

Se configurado na ferramenta, conforme apresentado na Figura 5.31, a escolha manual de equivalências pode alimentar o dicionário de sinônimos, que será utilizado para determinar outras equivalências em uma nova iteração com a ferramenta.

A partir daí, tanto no procedimento automático quanto no manual, o próximo passo é a validação das equivalências, como previsto na abordagem. Para tanto, clica-se na equivalência sugerida e através do botão  - Validar/Invalidar Match, executa-se a ação de validar ou invalidar tal equivalência, como mostrado na Figura 5.32.

Uma vez identificadas as *equivalências*, é possível detectar os elementos opcionais e mandatórios no domínio, que é o objetivo final desta primeira fase.

Elementos serão classificados como mandatórios se estiverem relacionados a uma equivalência que possua relação em pelo menos N arquiteturas, sendo N maior ou igual ao número configurado no critério em "**número mínimo de arquiteturas equivalentes"**, e opcionais, caso contrário. Essa verificação é feita automaticamente pela ferramenta. Assim, pode-se passar à segunda fase, cuja tela é apresentada na Figura 5.33

5.3.2 – Segunda Fase

Seguindo a abordagem proposta, na segunda fase, deverão ser detectados os pontos de variação (VPs) e suas respectivas variantes. De acordo com o algoritmo proposto na seção 4.3.1.1, interfaces e superclasses devem ser identificadas como candidatos em potencial. Neste contexto, o papel da ferramenta *ArchToDSSATool* é simplesmente implementar esse algoritmo, possibilitando a sugestão automática de tais pontos de variação.

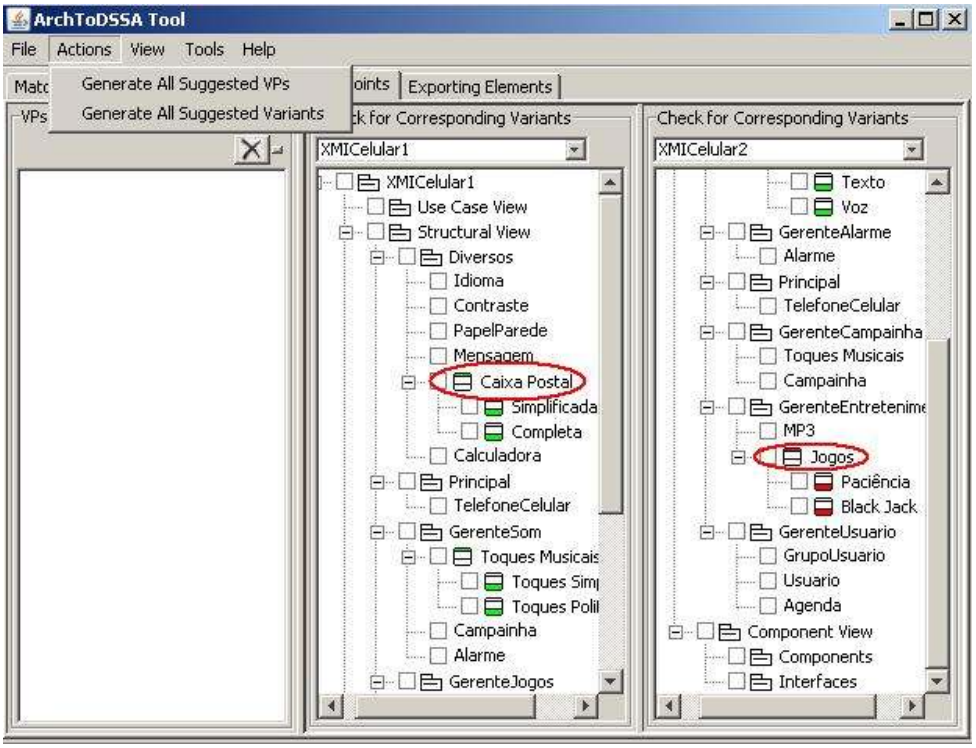

Figura 5.33 - *ArchToDSSA* - Segunda Fase

Uma vez que as arquiteturas analisadas são arquivos no formato XMI, é possível identificar, na estrutura do arquivo, que elementos da arquitetura são interfaces ou superclasses. A ferramenta faz uso de tal facilidade para sugerir os pontos de variação automaticamente. Além disso, é facilitada ao usuário da ferramenta a identificação desses elementos por meio de cores diferenciadas: os ícones dos elementos que

representam interfaces são coloridos de vermelho, enquanto os ícones dos elementos que representam superclasses são coloridos em verde, como destacado na Figura 5.33.

Por meio do menu *Actions* ► *Generate All Suggested VPs,* é possível gerar automaticamente todos os pontos de variação sugeridos com base em interfaces e superclasses. Da mesma forma, pode-se sugerir automaticamente as respectivas variantes através do menu *Actions* ► *Generate All Suggested Variants*. Na Figura 5.33, as variantes do ponto de variação "Caixa Postal" seriam "Simplificada" e "Completa", enquanto que as variantes do ponto de variação "Jogos" seriam "Paciência" e "BlackJack".

Assim como no caso das *equivalências*, é possível configurar os critérios **Número mínimo de arquiteturas equivalentes para se considerar uma superclasse como um VP** e **Número mínimo de arquiteturas equivalentes se considerar uma interface como um VP**, previstos na abordagem e descritos na seção 4.3.1. Para tanto, o menu *Tools* ►*Options* deve ser acessado, e na aba *VP Options* as configurações podem ser feitas, como mostra a Figura 5.34.

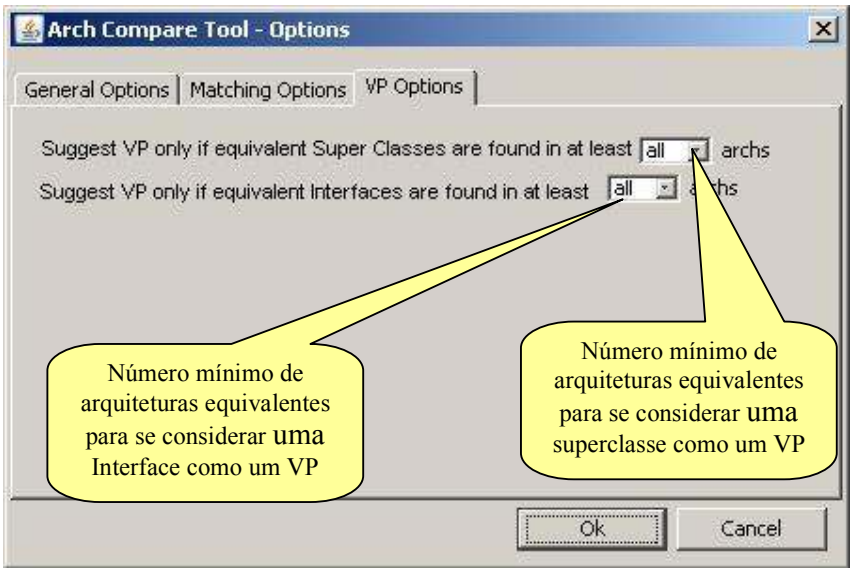

Figura 5.34 - Configuração de critérios para VPs

Também nesta fase, o engenheiro de domínio pode manipular a ferramenta, baseado em seu conhecimento, sugerindo novos pontos de variação. Para isso, basta clicar com o botão direito do mouse sobre o elemento para defini-lo como um VP, como mostra a Figura 5.35.

Uma vez identificados os pontos de variação (VPs), é interessante que, de alguma forma, seja facilitado ao engenheiro a escolha de suas respectivas variantes, no

caso do engenheiro preferir selecionar os elementos de forma não-automática. Isto pode ser feito, por exemplo, destacando-se os pontos de variação nas diferentes arquiteturas através de cores, mostrando de forma hierárquica seus respectivos candidatos a variantes e permitindo que se faça a seleção ou não destes candidatos. Uma vez selecionados, os elementos variantes são automaticamente associados aos pontos de variação para a fase de exportação.

Mesmo no caso de um ponto de variação ser gerado manualmente, é possível gerar automaticamente suas variantes. Tal geração automática se baseia na identificação de subclasses, no caso do ponto de variação ser uma superclasse, ou de classes que implementem a interface que foi definida como ponto de variação. Para essa geração, o menu *Actions* ▸ *Generate All Suggested Variants* deve ser acessado, como mencionado anteriormente.

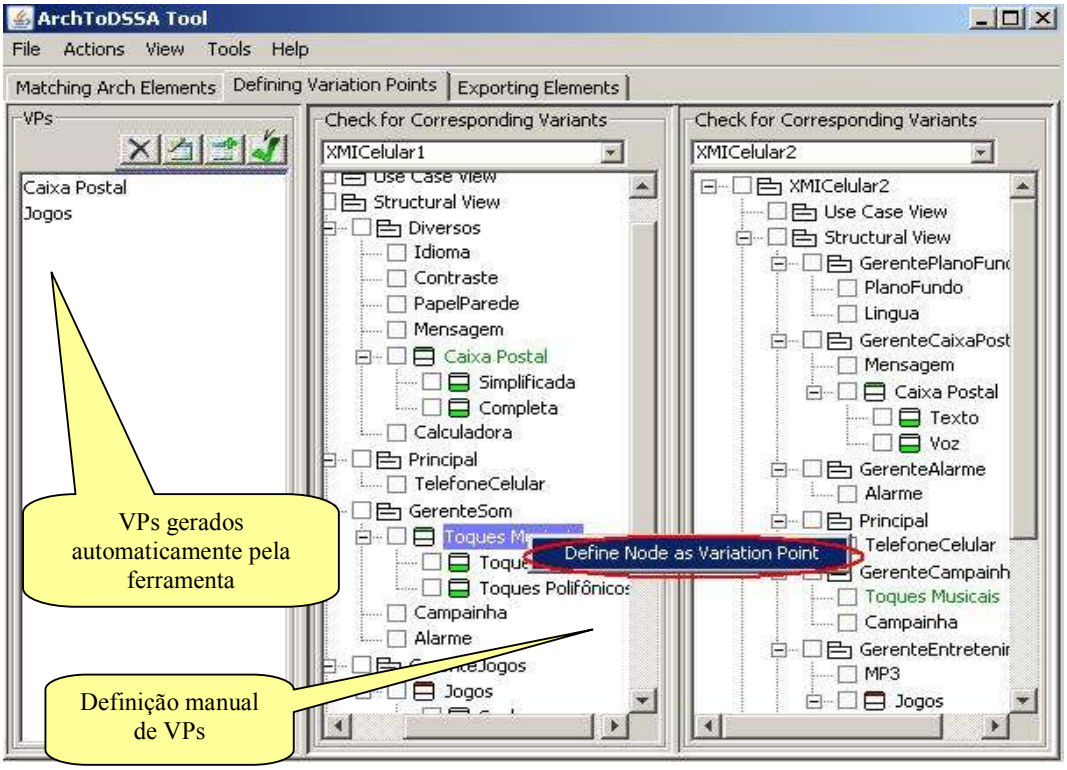

Figura 5.35 - Definição manual de pontos de variação

Uma vez gerados os pontos de variação e suas variantes, o engenheiro está apto a passar à terceira fase do processo. A tela referente à fase de exportação de elementos é apresentada na Figura 5.36.

5.3.3 - Terceira Fase

Na terceira fase da abordagem, deve-se escolher a arquitetura que vai servir de base para a arquitetura de referência (DSSA). Como mostra a Figura 5.36, a ferramenta possibilita essa escolha e, uma vez escolhida a arquitetura, todos os seus elementos arquiteturais são selecionados automaticamente, participando incondicionalmente da arquitetura de referência, como previsto na abordagem.

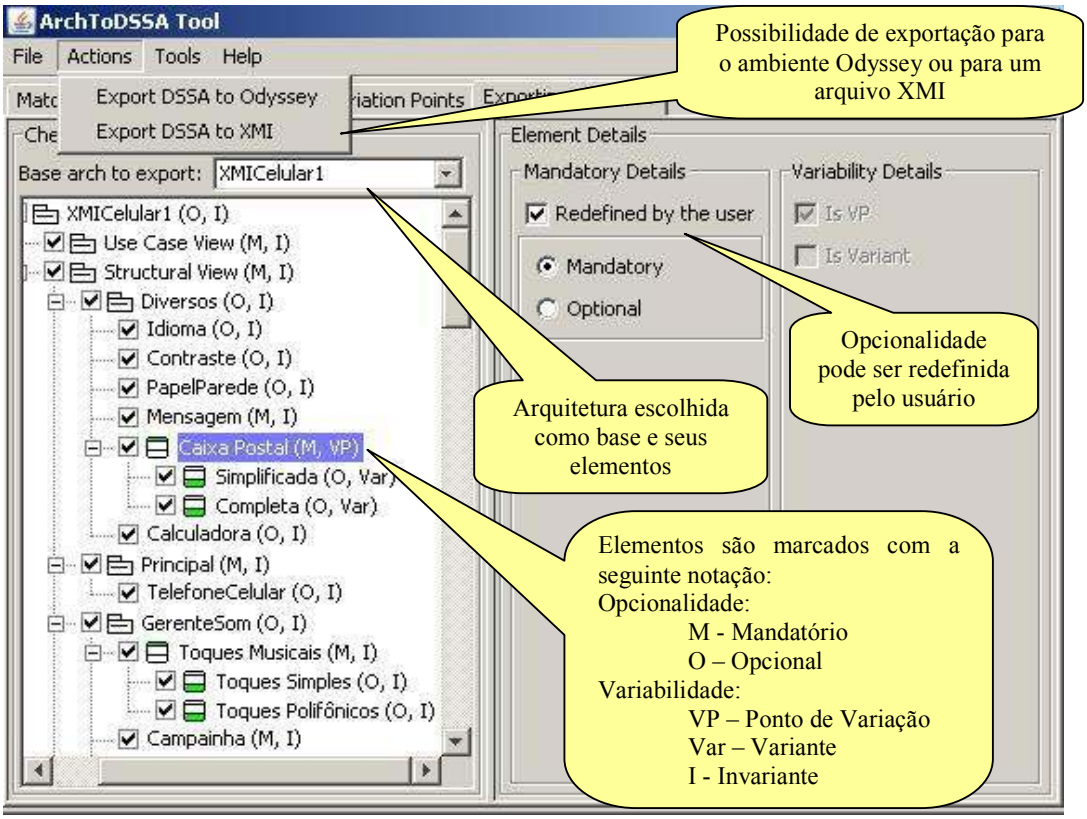

Figura 5.36 - *ArchToDSSA* - Terceira Fase

Esses elementos são assinalados na árvore que mostra os elementos arquiteturais. Além deles, são assinalados ainda os elementos de outras arquiteturas, diferentes da arquitetura-base, que possuem elementos que também farão parte da arquitetura de referência, conforme selecionados pelo engenheiro. Assim, seja, por exemplo, uma Arquitetura A como sendo a arquitetura-base e uma Arquitetura B diferente da arquitetura-base. Se a Arquitetura B possui um ponto de variação equivalente a um ponto de variação da arquitetura A, mas suas variantes são diferentes, tais variantes da Arquitetura B são também selecionadas para participar da arquitetura de referência.

As propriedades dos elementos arquiteturais que se referem a sua opcionalidade e variabilidade são visíveis nesta fase. Porém, para tornar a ferramenta o mais flexível possível, as propriedades de opcionalidade identificadas na primeira fase podem ser redefinidas pelo engenheiro na fase de exportação. No entanto, as propriedade de variabilidade são apresentadas na tela para conferência, sem a possibilidade de sobreposição nesta fase.

Além disso, a flexibilidade da ferramenta abrange a possibilidade de exportação da arquitetura de referência sugerida, de duas maneiras: a primeira opção está sempre presente e permite que a arquitetura exportada dê origem a um arquivo no formato XMI, que poderá ser acessado por outras ferramentas que trabalhem em uma versão de XMI compatível com a versão utilizada em *ArchToDSSATool*.; a segunda opção esta presente somente quando a ferramenta é acionada como um *plugin* do Odyssey. Neste caso é criado um arquivo XMI e automaticamente o ambiente Odyssey é comandado para que ele importe o referido arquivo, disponibilizando a DSSA gerada em seu ambiente de modelagem. A escolha da forma pela qual a exportação será feita é realizada através do menu Actions, que apresenta as duas opções: "Export DSSA to XMI" e "Export DSSA to Odyssey".

A Figura 5.37 mostra uma DSSA que foi exportada para XMI e pode ser manipulada através da ferramenta *Poseidon for UML* (GENTLEWARE, 2007).

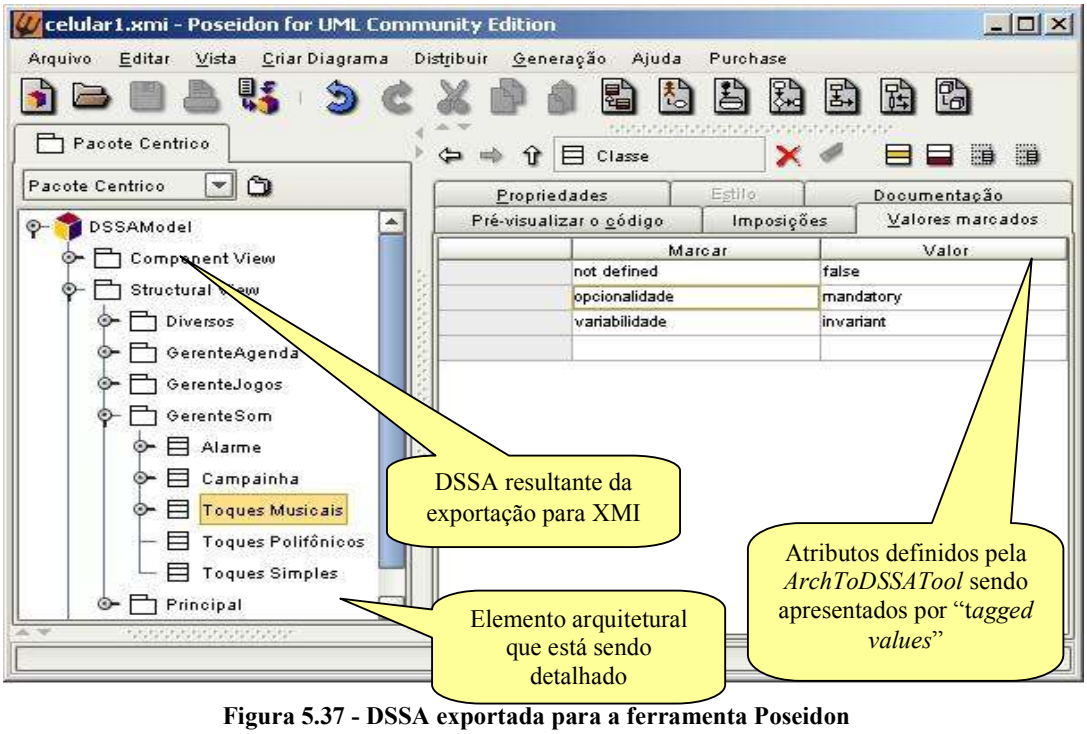

Figura 5.37 - DSSA exportada para a ferramenta Poseidon

A Figura 5.37 evidencia o elemento arquitetural ToquesMusicais e apresenta seus atributos de opcionalidade = mandatório e variabilidade = invariante como valores marcados (*tagged values*) no arquivo XMI gerado. O valor "*not defined*" apresentado na figura indica se o referido elemento é definido na arquitetura-base ou não. Sendo *not defined = false*, nota-se que o elemento fazia parte da arquitetura-base.

No caso da exportação para o Odyssey, a arquitetura resultante pode ser lida e manipulada no ambiente, sendo representada nos moldes da notação Odyssey-FEX, como mostra a Figura 5.38 a seguir. Após a exportação, os relacionamentos existentes na arquitetura-base são mantidos e também podem ser manipulados no ambiente Odyssey.

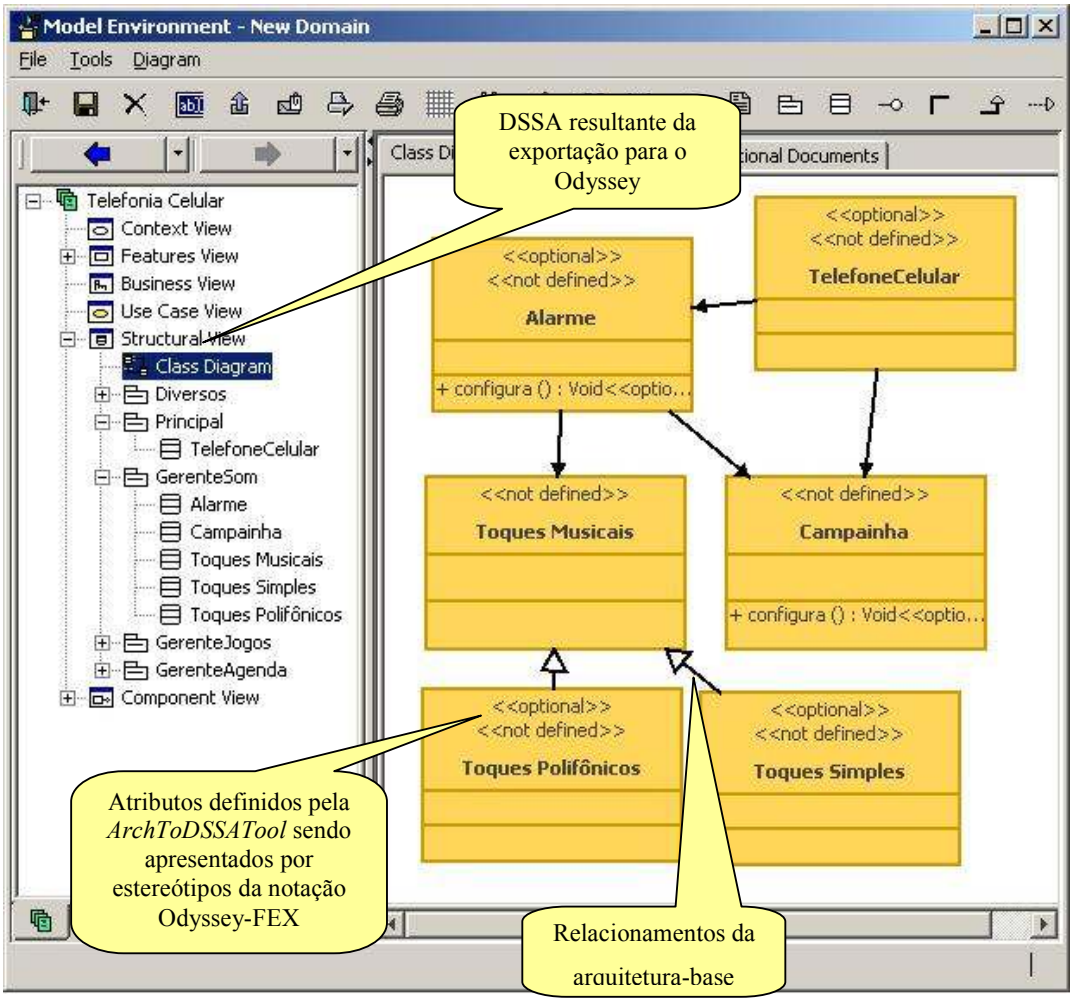

Figura 5.38 - DSSA exportada para o ambiente Odyssey

5.4 - Considerações Finais

Foi apresentada nesse capítulo a ferramenta *ArchToDSSATool*, no intuito de apoiar a execução da abordagem *ArchToDSSA*, proposta neste trabalho. Tal ferramenta tem por objetivo a automatização das atividades envolvidas no processo de definição de uma arquitetura de referência, a partir de arquiteturas existentes, oriundas de sistemas legados, ou não.

Foi apresentado, ainda, o protótipo da ferramenta desenvolvida, incluindo a sua relação com o ambiente Odyssey. No entanto, é importante relembrar a independência da ferramenta em relação ao ambiente, através de seu mecanismo de exportação da DSSA criada para arquivos XMI. No decorrer do capítulo, foi apresentada a maneira pela qual as etapas da abordagem proposta, i.e. detecção de equivalências e opcionalidades, detecção de variabilidades e criação de uma arquitetura de referencia no domínio, são tratadas pela ferramenta e como a automatização pode auxiliar o engenheiro de domínio no seu trabalho.

Uma possível melhoria do protótipo da ferramenta diz respeito aos critérios utilizados na escolha da arquitetura-base, que atualmente são arbitrários, cabendo ao engenheiro identificar qual das arquiteturas disponíveis mais se adequa a este papel. Uma alternativa seria a adaptação do protótipo à utilização em conjunto com a ferramenta MetricTool (MELO JR, 2005), que é a ferramenta de métricas do ambiente Odyssey, a fim de que tal definição pudesse ser guiada baseada em métricas que assegurem uma melhor escolha. A ferramenta MetricTool extrai métricas para a avaliação da qualidade de projetos OO através do modelo de classes ou do código e, embora disponibilize métricas do conjunto descrito em (CHIDAMBER & KEMERER, 1994), incluindo LCOM (*Lack of Cohesion in Methods*), CBO (*Coulping Between Object Classes*), NOC (*Number of Children*) e DIT (*Depth of Inheritance Tree*), permite a configuração de novas métricas. Para apoio à seleção da arquitetura-base, métricas apropriadas deveriam ser estudadas.

À medida que se for obtendo um maior *feedback* através da utilização não só da ferramenta, mas também da abordagem como um todo, evoluções poderão ser identificadas e realizadas, no intuito de se aprimorar não só o protótipo desenvolvido, como também a abordagem proposta neste trabalho.

Um primeiro passo nesse sentido foi dado com a realização de estudos de observação para averiguar a eficiência da abordagem e de seu ferramental de apoio. Tais estudos serão descritos no próximo capítulo.

Capítulo 6: Avaliação da abordagem *ArchToDSSA*

A abordagem *ArchToDSSA* tem por objetivo apoiar o engenheiro de domínio na criação de uma arquitetura de referência. Este apoio é realizado através de tarefas específicas e bem definidas, como a comparação de arquiteturas, identificação de suas opcionalidades, variabilidades e, finalmente, a criação e exportação de elementos arquiteturais que irão compor a arquitetura de referência do domínio. A especificação destas tarefas e, conseqüentemente, suas implementações através da ferramenta *ArchToDSSATool* foram realizadas com base em experiências pessoais, observações em ambiente empresarial e acadêmico, comparações com técnicas semelhantes e identificação de necessidades com base na análise da literatura, dentre outras fontes. Embora a concepção da abordagem tenha sido baseada em todas estas diferentes fontes de informação, não se pode assegurar que a abordagem seja eficiente, sem a realização de estudos experimentais (BABAR *et al.*, 2004).

Para que seja possível verificar se os objetivos propostos foram atingidos, dois estudos experimentais foram conduzidos. Estudos deste tipo consistem basicamente em uma observação sistemática dos efeitos de uma tecnologia em aplicações práticas (WOHLIN *et al.*, 2000).

Segundo a descrição encontrada em (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.

De acordo com WOHLIN *et al* (2000*)*, existem três diferentes estratégias de estudo experimental que podem ser adotadas, de acordo com o propósito da avaliação e das condições para o estudo empírico, tais como disponibilidade de recursos. São elas:

- **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 e, estudos

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.

A escolha a respeito do método de pesquisa experimental a ser utilizado depende dos pré-requisitos da investigação, do propósito, da disponibilidade de recursos e de como se pretende analisar os dados coletados (WOHLIN *et al.*, 2000). Neste trabalho, foi utilizado um experimento da categoria "estudo de caso". Tal escolha se deve a uma série de fatores, dentre eles: a escassez de tempo e recursos (ex: participantes) para o planejamento e execução do estudo; o tipo de avaliação que se deseja realizar, isto é, a caracterização da viabilidade da abordagem proposta; e a impossibilidade de controle total sobre as variáveis do estudo (ex: perfil dos participantes, que avaliaram os objetos do estudo de forma subjetiva).

Foram realizados dois estudos de observação, onde o objetivo do primeiro estudo foi investigar a eficiência da abordagem proposta, no que diz respeito ao apoio na criação de uma arquitetura de referência a partir da comparação de arquiteturas de software em um domínio. O segundo estudo de observação teve como objetivo investigar a eficiência do ferramental desenvolvido no apoio ao engenheiro de domínio na criação da DSSA. Os estudos realizados visaram medir a eficiência, tanto da abordagem quanto da ferramenta, em relação à corretude dos resultados obtidos como descritos a seguir.

6.1 - Primeiro Estudo de Observação: Abordagem

Nesta seção é apresentado o primeiro estudo de observação conduzido. Além da descrição do estudo, são apresentados também os principais resultados obtidos.

6.1.1 - Descrição do Primeiro Estudo

Como resultado da condução desse primeiro estudo de observação, esperava-se obter, dos participantes envolvidos, informações que contribuíssem para o refinamento da proposta deste trabalho. Além disso, esperava-se também obter indícios de que a utilização da *ArchToDSSA* atinge o objetivo proposto, i.e., apoiar a criação de uma arquitetura de referência no domínio de forma correta.

O primeiro estudo de observação foi conduzido em um ambiente acadêmico, no grupo de Reutilização de Software do Programa de Engenharia de Sistemas e Computação da Coordenação dos Programas de Pós-Graduação em Engenharia – COPPE/UFRJ. O estudo empregou quatro participantes, sendo dois de Graduação, um de Mestrado e um de Doutorado, que receberam treinamento na utilização da abordagem através de material distribuído previamente, conforme os Anexos II e III. O resultado das atividades propostas para esse estudo foi observado pelo pesquisador autor deste trabalho. A seguir, o primeiro estudo de observação é detalhado.

Objetivo: O estudo teve como *objetivo* analisar a abordagem *ArchToDSSA; com o propósito* de caracterizar; *com respeito à* eficiência da abordagem no apoio à tarefa de criação de uma DSSA; *do ponto de vista* de engenheiros de software; *no contexto* da comparação de arquiteturas de aplicações em domínios específicos.[7]

Participantes: O estudo empregou quatro estudantes de Engenharia de Software com diferentes níveis de experiência em conceitos como domínio e arquitetura de software. A experiência de cada participante foi aferida segundo um formulário de caracterização (Anexo VII), sendo caracterizada segundo quatro aspectos:

- Experiência prática no desenvolvimento de software;
- Experiência com requisitos;
- Experiência em arquiteturas de software;
- Experiência em projetos.

Para todos os aspectos, foi utilizada uma escala ordinal envolvendo quatro opções, a saber: (1) Nenhum, (2) Estudei em aula ou em livro, (3) Pratiquei em um projeto em sala de aula, (4) Usei em um projeto na indústria, e (5) Usei em vários projetos na indústria. Dos quatro participantes, apenas um tinha alguma experiência avançada. Os outros três participantes relataram possuir experiência, em média, nos níveis intermediários da escala.

[7] Objetivo definido segundo o método GQM (BASILI *et al.*, 1994).

Material Utilizado: Para a realização desse estudo, primeiramente foi entregue aos engenheiros de software o formulário de consentimento (Anexo I), juntamente com as instruções para execução dos procedimentos e preenchimento dos formulários (Anexo III). Foram disponibilizadas arquiteturas, objetos de estudo, nos domínios de Telefonia Celular e também no domínio Escolar, da seguinte forma: foram entregues, em momentos diferentes, diagramas das arquiteturas de cada domínio (Anexo V), e um formulário para que os resultados dos procedimentos fossem anotados (Anexo IV). Além disso, para facilitar o entendimento da abordagem, foi entregue um documento introdutório, explicando aos participantes sobre os propósitos do estudo, bem como um exemplo de utilização da abordagem (Anexo II).

Como o objetivo é avaliar a eficiência da abordagem, i.e., a corretude dos resultados obtidos com a *ArchToDSSA*, as arquiteturas foram criadas de forma que simulassem arquiteturas extraídas de software legado, ou seja, elas contêm nomes distintos para elementos arquiteturais que têm o mesmo papel semântico. Conforme dito anteriormente, isto ocorre porque muitas vezes, as diferentes aplicações, mesmo tratando de um mesmo domínio, são criadas por equipes, departamentos, empresas ou em momentos distintos. Para evitar discrepâncias nos resultados, em função dos diferentes entendimentos que engenheiros do domínio possuem sobre um mesmo domínio, foi disponibilizado um único dicionário de sinônimos (Anexo VI).

Descrição do Exercício: Para a condução do estudo foram apresentados aos participantes os conceitos de *ArchToDSSA,* juntamente com um exemplo ilustrando sua aplicação no domínio de MP3 Player, além dos procedimentos a serem seguidos pelos participantes durante a condução do estudo. Para cada participante, foi distribuído o material já citado anteriormente, como forma de possibilitar o esclarecimento de eventuais dúvidas em relação à aplicação da abordagem proposta. Era esperado, como produto desse exercício, as equivalências, opcionalidades e variabilidades dos elementos que comporiam a arquitetura de referência em cada domínio apresentado.

O exercício consistiu do seguinte procedimento: em um primeiro momento, para a primeira execução do exercício, cada participante recebeu as arquiteturas de exemplo de um domínio específico, i.e., o domínio de Telefonia Celular ou o domínio Escolar. Os domínios foram previamente explicados pelo pesquisador condutor do estudo. A tarefa de cada um dos participantes era seguir as fases propostas em ArchToDSSA, obtendo as equivalências, os elementos arquiteturais mandatórios e opcionais no domínio, bem como os pontos de variação e suas variantes. Em seguida, os participantes realizaram

novamente os procedimentos em um segundo domínio, diferente do primeiro que havia sido apresentado, conforme ilustrado na tabela 6.2.

Tabela 6.2 – Distribuição das atividades aos participantes do primeiro estudo de observação.

Participante	Primeira Execução	Segunda Execução
1	Domínio Escolar	Domínio Telefonia Celular
2	Domínio Escolar	Domínio Telefonia Celular
3	Domínio Telefonia Celular	Domínio Escolar
4	Domínio Telefonia Celular	Domínio Escolar

A repetição do procedimento tem como objetivo atenuar a curva de aprendizado do participante, permitindo que ele se concentre mais na utilização da abordagem do que em eventuais dificuldades nesta utilização. A idéia da utilização de domínios distintos nesta repetição tem como objetivo evitar vícios eventualmente adquiridos no domínio, que pudessem prejudicar a avaliação do participante na utilização da abordagem, caso ele realizasse o procedimento duas vezes com o mesmo domínio. Desta forma, espera-se avaliar a evolução do participante no uso da abordagem, independente do domínio no qual ele estaria trabalhando. Como existiam dois domínios distintos e quatro participantes, decidiu-se alternar os domínios entre cada dois participantes nos momentos da execução.

Após a realização do exercício, os resultados foram coletados e comparados com um gabarito previamente estabelecido, no intuito de avaliar o percentual de corretude, i.e, a eficiência da utilização da abordagem em cada fase. Esse percentual foi calculado segundo as seguintes métricas:

Métrica Fase 1: Percentual de equivalências identificadas de forma correta através da utilização da abordagem em relação ao número de equivalências existentes no gabarito.

$$\textbf{EficienciaMatches\%} = \frac{\textbf{NumMatchesOk} * 100}{\textbf{NumMatchesGabarito}}$$

onde:

EficienciaMatches%: percentual de acertos na identificação de equivalências pelo participante;

NumMatchesOk: número de equivalências identificadas corretamente pelo participante;

NumMatchesGabarito: número total de equivalências existentes, de acordo com o gabarito existente.

Métrica Fase 2: Percentual de pontos de variação e variantes ligadas a estes identificados de forma correta através da utilização da abordagem:

$$\textbf{EficiênciaVP\%} = \frac{\textbf{NumVPOk} * 100}{\textbf{NumVPGabarito}}$$

$$\textbf{EficiênciaVar\%} = \frac{\textbf{NumVarOk} * 100}{\textbf{NumVarGabarito}}$$

onde

EficiênciaVP%: percentual de acertos na identificação de pontos de variação pelo participante;

NumVPOk: número de pontos de variação identificados corretamente pelo participante;

NumVPGabarito: número total de pontos de variação existentes, de acordo com o gabarito;

EficiênciaVar%: percentual de acertos na identificação de variantes pelo participante;

NumVarOk: número de variantes identificadas corretamente pelo participante;

NumVarGabarito: número total de variantes existentes, de acordo com o gabarito.

A criação da DSSA, resultado da Fase 3 da abordagem, pode ser inferida a partir dos elementos identificados corretamente nas duas primeiras fases. Além disso, a criação da DSSA é impactada tanto pela escolha da arquitetura-base, quanto pela escolha dos elementos opcionais pelo participante. Por serem essas escolhas subjetivas, a comparação do resultado final com um gabarito não faria sentido. Por essas razões, as métricas utilizadas contemplam apenas as Fases 1 e 2.

6.1.2 - Resultados do Primeiro Estudo

A análise do primeiro estudo foi feita da seguinte forma: para cada domínio analisado, foi criado um gabarito contendo, em função de seus dicionários de sinônimos e da estrutura de suas arquiteturas, as equivalências, opcionalidades e variabilidades esperadas. Após a execução do estudo de observação, foram coletados os resultados de cada participante. Estes resultados foram comparados com os gabaritos, e um percentual de corretude, i.e., a eficiência, pôde ser calculado para os resultados de cada fase, conforme as métricas já apresentadas.

Para cada fase da abordagem, foi calculado o percentual de acertos de cada participante em cada domínio. Em seguida foi calculada a média aritmética do percentual de acerto em cada uma das fases, em relação aos dois domínios estudados. Conforme já mencionado, tal média é calculada a fim de equilibrar eventuais discrepâncias causadas pela diferença na curva de aprendizado de cada participante durante a utilização da abordagem. Considera-se que, na segunda execução, o participante já esteja mais familiarizado com os procedimentos do estudo e também com o primeiro domínio utilizado. Por isso, troca-se o domínio e calcula-se a média das duas execuções. Os resultados obtidos na primeira e segunda fases da abordagem são apresentados, respectivamente, na tabela 6.3, tabela 6.4 e tabela 6.5.

Tabela 6.3 – Média de acertos dos participantes na primeira fase da abordagem

Participante	EficienciaMatches% Primeira Execução	EficienciaMatches% Segundoa Execução	Média
1	100	100	100
2	44,83	92,30	68,57
3	100	100	100
4	84,62	93,10	88,86
		Média geral:	**89,35**

Tabela 6.4 - Média de acertos dos participantes na segunda fase da abordagem - VP

Participante	EficiênciaVP% Primeira Execução	EficiênciaVP% Segunda Execução	Média
1	100	100	100
2	48,28	100	74,14
3	87,5	100	93,75
4	87,5	86,20	86,85
		Média geral:	**88,68**

Tabela 6.5 - Média de acertos dos participantes na segunda fase da abordagem - Variantes

Participante	EficiênciaVar% Primeiro Domínio	EficiênciaVar% Segundo Domínio	Média
1	84,93	100	92,46
2	19,18	100	59,59
3	88,24	100	94,12
4	88,24	73,97	81,10
		Média geral:	**81,81**

Para auxiliar na avaliação da abordagem, algumas perguntas subjetivas foram feitas aos participantes do estudo, como mostrado a seguir:

1) *Os conceitos de Arquiteturas e DSSA já eram de seu conhecimento antes do estudo realizado? E depois?*

Um participante respondeu que já tinha conhecimento dos conceitos e os outros três participantes responderam que melhoraram o conhecimento dos conceitos depois do estudo realizado.

2) Qual o grau de dificuldade ou facilidade em realizar as etapas sugeridas? Explique.

Todos os participantes alegaram que a tarefa não tinha muitas dificuldades, porém era muito cansativa e suscetível a erros.

3) A abordagem de alguma forma o ajudou na comparação de arquiteturas e criação da DSSA?

Um participante alegou que a abordagem não facilitou em nada, pelo contrário, criou dificuldades que, no seu entender, não existiriam se o processo de comparação fosse feito de maneira *ad hoc*. Os outros três participantes alegaram ter sido mais fácil identificar pontos de variação e variantes a partir das equivalências encontradas, como sugerido na abordagem.

4) Como a abordagem poderia ser melhorada?

Um participante não respondeu a essa pergunta. Um segundo não sugeriu melhorias. Outros dois sugeriram melhorias tais como automatização do processo, ou alguma ordenação das entradas no dicionário de sinônimos.

5) Acredita que uma ferramenta iria ajudá-lo na execução das tarefas? De que forma?

Todos os participantes alegaram que uma ferramenta auxiliaria no processo, automatizando repetições excessivas e tornando a execução da tarefa menos cansativa e suscetível a erros.

De acordo com os resultados obtidos neste primeiro estudo, pôde-se concluir que a média geral de acertos, i.e, a eficiência da abordagem, ultrapassou a marca de 80%. Esse valor, somado às respostas dos participantes ao formulário de questões subjetivas, oferece indícios da viabilidade da abordagem proposta Além disso, de um modo geral, a média obtida na segunda execução foi mais alta do que a da primeira execução. Isso leva a crer que a eficiência da abordagem cresce com sua utilização de forma continuada.

Um outro fato observado também neste primeiro estudo foi que grande parte dos erros cometidos pelos participantes na identificação de pontos de variação e variantes se deve à dificuldade causada quando um elemento é, ao mesmo tempo, variante em relação a um elemento, e ponto de variação em relação a outros. Uma vez que tal situação não havia sido prevista inicialmente, os participantes não receberam instruções para tal classificação, transformando tal limitação na principal causa de erros na execução das tarefas do estudo realizado.

Com relação à automatização da abordagem, foi constatado que os participantes, em sua totalidade, concordam que seria válida a construção de uma ferramenta de apoio. Uma vez que tal ferramental de apoio foi implementado no contexto dessa dissertação, foi realizado um segundo estudo de observação, no intuito de verificar a eficiência da ferramenta ArchToDSSAtool, como apresentado a seguir.

6.2 - Segundo Estudo de Observação : Ferramenta

Prosseguindo na avaliação da abordagem *ArchToDSSA* e do seu ferramental de apoio, nesta seção, é apresentado o segundo estudo de observação conduzido.

6.2.1 - Descrição do Segundo Estudo

O segundo estudo de observação foi conduzido com o objetivo de avaliar a eficiência do ferramental de apoio proposto, denominado neste trabalho

ArchToDSSATool. Entende-se por eficiência, o percentual de corretude que o participante do estudo obtém na criação de uma DSSA utilizando a ferramenta.

Tal estudo foi realizado em um ambiente empresarial, onde os participantes são desenvolvedores de software com experiência em desenvolvimento, porém com pouca familiaridade com conceitos como domínio e arquiteturas de referência. Como resultado desse estudo, era esperado que os participantes conseguissem identificar elementos que pudessem compor uma DSSA, da maneira mais correta possível, com suas equivalências, opcionalidades e variabilidades, em dois domínios distintos, com o auxílio da ferramenta, mesmo tendo pouca experiência com os conceitos envolvidos na criação de uma arquitetura de referência. A exemplo do primeiro estudo, o desenvolvimento das atividades propostas para esse estudo também foi observado pelo pesquisador autor deste trabalho. A seguir, o segundo estudo de observação é detalhado.

Objetivo: O estudo teve como *objetivo* analisar a ferramenta *ArchToDSSATool; com o propósito* de caracterizar; *com respeito à* eficiência da ferramenta no apoio à tarefa de criação de uma DSSA; *do ponto de vista* de desenvolvedores de software; *no contexto* da comparação de arquiteturas de aplicações em domínios específicos.

Participantes: O estudo empregou quatro desenvolvedores de software com diferentes níveis de experiência em desenvolvimento de software na indústria e em conceitos como domínio e arquitetura de software. Todos os participantes possuem graduação completa, e relataram níveis de experiência em desenvolvimento de software na indústria compreendidos entre 2 anos e meio e 11 anos. Um deles trabalha na área de desenvolvimento web, dois com desenvolvimento *desktop* e um com a área de engenharia de software.

A exemplo do primeiro estudo, a experiência de cada participante foi aferida segundo o formulário de caracterização do Anexo VII, e também sendo caracterizada segundo os aspectos:

- Experiência prática no desenvolvimento de software;
- Experiência com requisitos;
- Experiência em arquiteturas de software;
- Experiência em projetos.

Também aqui foi utilizada a escala ordinal envolvendo as quatro opções: (1) Nenhum, (2) Estudei em aula ou em livro, (3) Pratiquei em um projeto em sala de aula, (4) Usei em um projeto na indústria, e (5) Usei em vários projetos na Indústria. Dos

quatro participantes, dois deles tinham experiência no nível 5 da escala em arquiteturas de software e projetos Os outros dois participantes relataram possuir experiência, em média, nos níveis 3 e 4 da escala para a maioria dos aspectos investigados.

Material Utilizado: Para a realização desse estudo, assim como no estudo anterior, foram entregues aos desenvolvedores de software, primeiramente, o formulário de consentimento (Anexo I), juntamente com as instruções para a execução dos procedimentos e preenchimento dos formulários (Anexo III). Foram entregues também as arquiteturas de exemplo nos domínios Escolar e de Telefonia Celular no formato XMI e o dicionário de sinônimos do domínio com algumas entradas preenchidas. No intuito de se obter os mesmos critérios de comparação nos dois estudos realizados, o dicionário de sinônimos utilizado no segundo estudo foi o mesmo utilizado no primeiro estudo.

Descrição do Exercício: A condução do segundo estudo foi precedida por uma sessão de treinamento, onde foram apresentados aos participantes os conceitos de *ArchToDSSATool*, um exemplo ilustrando sua aplicação, além dos procedimentos a serem seguidos pelos participantes durante a execução do estudo com a utilização da ferramenta. Este treinamento teve o intuito de esclarecer eventuais dúvidas em relação à aplicação da abordagem proposta ou do uso da ferramenta. Era esperado, como produto desse exercício, as equivalências, opcionalidades e variabilidades dos elementos que comporiam a arquitetura de referência em cada domínio apresentado, em arquivos no formato XMI, de forma mais correta possível, independente do grau de familiaridade dos participantes com os conceitos envolvidos em uma DSSA.

O exercício consistia do seguinte procedimento: em um primeiro momento, para a primeira execução do estudo, cada participante recebeu as arquiteturas de exemplo de um domínio específico. A tarefa de cada um deles era seguir as fases propostas em *ArchToDSSA*, obtendo as equivalências, os elementos arquiteturais mandatórios e opcionais no domínio, bem como os pontos de variação e suas variantes. Em seguida, os participantes realizaram novamente os procedimentos em um segundo domínio, diferente do primeiro que havia sido apresentado, conforme ilustrado na tabela 6.6.

Tabela 6.6 - Distribuição das atividades aos participantes do segundo estudo de observação

Participante	Primeira Execução	Segunda Execução
1	Domínio Telefonia Celular	Domínio Escolar
2	Domínio Telefonia Celular	Domínio Escolar
3	Domínio Escolar	Domínio Telefonia Celular
4	Domínio Escolar	Domínio Telefonia Celular

6.2.2 – Resultados do Segundo Estudo

A análise do segundo estudo foi feita de forma semelhante à do primeiro estudo: os gabaritos contendo as equivalências, opcionalidades e variabilidades esperadas, criados para o primeiro estudo, foram aqui utilizados. Uma vez que o resultado final da utilização da ferramenta é a criação de um arquivo no formato XMI, após a execução do segundo estudo de observação, as informações contidas em tal arquivo foram comparadas com os gabaritos e um percentual de corretude, i.e., a eficiência, pôde ser calculado para os resultados de cada fase, segundo as mesmas métricas apresentadas na seção 6.1.1.

Novamente, para cada fase da abordagem, foi calculado o percentual de acertos de cada participante em cada domínio, e calculada uma média aritmética do percentual de acerto nos dois domínios distintos. Os resultados obtidos na primeira e segunda fases da abordagem são apresentados, respectivamente, nas tabelas 6.7, 6.8 e 6.9.

Tabela 6.7 – Média de acertos dos participantes na primeira fase da abordagem com o uso da ferramenta

Participante	EficienciaMatches% Primeira Execução	EficienciaMatches% Segunda Execução	Média
1	100	86,20	93,1
2	100	86,20	93,1
3	86,20	100	93,1
4	86,20	100	93,1
		Média geral:	**93,1**

Tabela 6.8 - Média de acertos dos participantes na segunda fase da abordagem com o uso da ferramenta

Participante	EficiênciaVP% Primeira Execuçào	EficiênciaVP% Segunda Execução	Média
1	100	48,27	74,13
2	100	48,27	74,13
3	48,27	100	74,13
4	48,27	100	74,13
		Média geral:	**74,13**

Tabela 6.9- Média de acertos dos participantes na segunda fase da abordagem com o uso da ferramenta

Participante	EficiênciaVar% Primeira Execução	EficiênciaVar% Segunda Execução	Média
1	100	89,04	94,52
2	100	89,04	94,52
3	89,04	100	94,52
4	89,04	100	94,52
		Média geral:	**94,52**

Uma observação que foi feita, durante a execução deste estudo, foi que as médias de acerto foram exatamente iguais em cada domínio, conforme mostram as tabelas 6.7, 6.8 e 6.9. Isso se deve ao fato de que, embora os participantes tivessem liberdade para criar equivalências, apontar pontos de variação e variantes manualmente, nenhum deles o fez. Todos optaram por usar as opções de "sugestões automáticas" disponíveis em *ArchToDSSATool*. Esse comportamento pode ter ocorrido em função de alguma eventual insegurança, motivada pela pouca experiência dos participantes na criação de DSSAs, mesmo tendo estes recebido treinamento para a execução do estudo Dessa forma, a ferramenta apresentou o mesmo comportamento em todas as execuções, uma vez que não houve diferença de configuração entre os participantes do estudo.

É importante ressaltar, ainda, que, no caso da detecção de pontos de variação e variantes, a ferramenta teve um desempenho abaixo do esperado. Isso se deve ao fato de que, conforme mencionado anteriormente, a ferramenta, assim como a abordagem, não trata situações onde um elemento é, ao mesmo tempo, ponto de variação e variante. Isso ocasionou divergências nos resultados obtidos pelo uso da ferramenta, em relação ao gabarito pré-estabelecido, especificamente na detecção automática de pontos de variação. Ainda assim, a média geral de cada fase ficou acima da marca de 90%, o que oferece indícios sobre a eficiência da ferramenta *ArchToDSSATool*.

A exemplo do primeiro estudo, algumas perguntas subjetivas foram feitas aos participantes, no intuito de auxiliar na avaliação da ferramenta, como mostrado a seguir:

1) Os conceitos de Arquiteturas e DSSA já eram de seu conhecimento antes do estudo realizado? E depois?

Três participantes alegaram ter conhecimento do conceito de arquitetura de software, mas não de DSSA, e afirmaram terem assimilado seu significado após o

estudo realizado. Um dos participantes alegou ter adquirido o conhecimento sobre tais conceitos somente depois do estudo.

2) Qual o grau de dificuldade ou facilidade em realizar as etapas sugeridas? Explique.

Todos os participantes alegaram não ter dificuldades com a ferramenta, classificando-a como intuitiva e de fácil utilização. No entanto, todos eles ressaltaram que a dificuldade com a ferramenta, caso existisse, provavelmente seria proveniente de eventuais dificuldades de entendimento dos conceitos do domínio utilizado. Porém, como todos receberam treinamento a respeito do domínio, tais dificuldades não foram encontradas.

3) A Ferramenta de alguma forma o ajudou na comparação de arquiteturas e criação da DSSA?

Todos os participantes responderam sim a esta pergunta, ressaltando o recurso de apresentação dos elementos arquiteturais em níveis hierárquicos e a facilidade de visualização dos resultados obtidos.

4) Como a ferramenta poderia ser melhorada?

As melhorias sugeridas pelos participantes incluem aprimoramento da detecção de equivalências, por exemplo, com a inserção de um campo de busca dentro das arquiteturas, facilitando a localização dos seus elementos, que eram muitos, e modificações na interface no intuito de deixá-la mais "bonita".

5) Como seria este processo sem a ferramenta? Fácil, Difícil, Rápido, Demorado? Explique

Todos os participantes responderam que o processo sem a ferramenta seria muito demorado, além de suscetível a erros e com alto custo operacional.

6) A ferramenta apresentada pode ser utilizada em alguma etapa do desenvolvimento de software da sua área de trabalho?

Um participante não respondeu a essa pergunta. Dois participantes responderam que não vêem ligação direta com sua área de trabalho, e um dos participantes, que

trabalha com engenharia de software, apontou que, no seu entender, a ferramenta poderia ser útil.

Uma vez coletados os resultados do segundo estudo, as médias de acertos obtidos neste estudo, no qual os participantes eram "leigos", isto é, com pouca experiência nos conceitos de domínio e arquiteturas de referência, e dependiam do auxílio da ferramenta, foram comparados com as médias de acertos obtidas no primeiro estudo de observação, no qual os participantes já detinham algum conhecimento sobre o assunto. Na Tabela 6.10, é apresentado um comparativo destas médias.

Tabela 6.10 – Comparação dos resultados dos dois estudos de observação

Fase	Média Geral Primeiro Estudo (Manual Com Experiência)	Média Geral Segundo Estudo (Ferramenta Sem Experiência)
EficienciaMatches%	89,35	93,10
EficiênciaVP%	88,68	74,13
EficiênciaVar%	81,81	94,52

De acordo com os resultados apresentados na Tabela 6.10, é possível obter indícios sobre a eficiência da abordagem, uma vez que possibilitou a identificação de equivalências, opcionalidades e variabilidades com alto percentual de corretude em diferentes domínios. Além disso, há indícios também sobre a eficiência do ferramental de apoio, uma vez que possibilitou a usuários com pouca experiência nos conceitos de domínio e arquiteturas de referência, a identificação de tais componentes com um percentual de corretude relativamente alto. Embora a média geral do procedimento com a utilização da ferramenta não tenha sido tão superior à media do processo manual, pôde-se observar o aumento de produtividade proporcionado pela ferramenta, no sentido de possibilitar a execução das tarefas em um tempo consideravelmente menor do que o tempo gasto no processo manual. O processo manual levou, em média, duas horas e meia, e o processo utilizando a ferramenta levou, em média quinze minutos, mesmo considerando-se o fato de que os participantes que fizeram o processo manualmente estavam melhor preparados para a execução do estudo, devido ao fato de serem acadêmicos da área de reutilização de software. Levando-se em consideração que a ferramenta *ArchToDSSATool* é a implementação do processo proposto na abordagem *ArchToDSSA*, é possível obter indícios de que a proposta deste trabalho se mostra

eficiente no auxílio a engenheiros de domínio, independente de sua experiência e conhecimento em arquiteturas de referência.

6.3 - Considerações Sobre os Estudos de Observação

Embora os estudos conduzidos tenham conseguido avaliar a eficiência do processo proposto para a criação de uma DSSA, algumas questões sobre estes devem ser consideradas. Como exemplo, tem-se a existência de um dicionário de sinônimos para o domínio, previamente distribuído ao participante, o que poderia induzi-lo à criação da uma DSSA correta. No entanto, tal medida foi necessária para que não houvesse discrepância nos resultados obtidos, uma vez que o entendimento do domínio pode ser muito diferente na concepção dos diversos participantes. Outra questão é a escolha do perfil dos participantes para o estudo: por um lado, para a avaliação da abordagem, foram escolhidos somente participantes do ambiente acadêmico (estudantes da área de reutilização de software). Com isto esperava-se um senso mais crítico em relação aos procedimentos propostos, devido ao maior grau de conhecimento e familiaridade destes com os conceitos envolvidos. Por outro lado, para avaliar a ferramenta, optou-se apenas por participantes do ambiente empresarial. A idéia era justamente tentar provar a corretude da ferramenta em relação à abordagem, mostrando que participantes com menos conhecimento sobre os assuntos envolvidos eram capazes de executar as tarefas de forma correta usando a ferramenta. Uma vez que a relação de cada grupo com o autor do estudo (i.e., colegas de estudo no primeiro grupo e funcionários no segundo grupo) é diferente, o nível de envolvimento de cada grupo com o autor também é diferente, o que levou à opção de usar dois grupos distintos, no intuito de evitar qualquer comprometimento na condução dos estudos, contrabalançando o conhecimento sobre o assunto e envolvimento com o autor.

Limitações foram identificadas durante os estudos. Eles apontaram a necessidade de tratamento especial aos elementos que se comportam como ponto de variação e variante. Além disso, as arquiteturas utilizadas nos estudos, embora tenham sido realmente extraídas de sistemas legados, estão longe de serem completas, do mesmo modo que os participantes de tais estudos não são especialistas no domínio.

Os estudos de observação realizados no contexto dessa dissertação não têm a pretensão de serem estudos completos e com total abrangência para a avaliação da abordagem e ferramenta propostas. Foram usadas arquiteturas de tamanhos limitados e de apenas dois domínios. É necessário que estudos mais profundos, envolvendo

domínios mais complexos e maiores sejam realizados no futuro, no intuito de apurar a necessidade de outras melhorias em *ArchToDSSA* e/ou *ArchToDSSATool*.

Capítulo 7: Conclusões e Trabalhos Futuros

7.1 - Contribuições

Foi apresentada, nessa dissertação, a abordagem *ArchToDSSA*, cujo objetivo é apoiar a criação de arquiteturas de referência no domínio, a partir da análise de arquiteturas de aplicações em um domínio específico. Além disso, foi apresentada a ferramenta *ArchToDSSATool*, protótipo desenvolvido visando a automatização da abordagem proposta.

Tendo em vista as deficiências encontradas em trabalhos existentes na literatura, a pesquisa realizada nessa dissertação apresenta como principais contribuições:

1. *Uma abordagem para a criação de uma arquitetura de referência em um domínio a partir da comparação de arquiteturas existentes, de sistemas legados, ou não*. A abordagem *ArchToDSSA*, proposta neste trabalho, apresenta as seguintes características que representam suas contribuições:
    - **Comparação de diferentes arquiteturas**: diferentemente de algumas abordagens analisadas, cuja ênfase é na comparação entre versões diferentes de uma mesma arquitetura, a abordagem *ArchToDSSA* permite a comparação de arquiteturas diferentes em um domínio. Através da identificação de elementos arquiteturais semanticamente equivalentes, possibilita uma maior precisão na escolha dos elementos que farão parte da DSSA, reduzindo a ocorrência de redundância na arquitetura de referência final.
    - **Criação de um dicionário de sinônimos**: a criação e alimentação de um dicionário de sinônimos para o domínio possibilitam não só uma maior flexibilidade ao trabalho do engenheiro, como também dão origem a uma base para um maior conhecimento do domínio.
    - **Comparação de elementos através da divisão de nomes**: o critério de comparação de elementos por meio da divisão de nomes representa mais uma alternativa para lidar com a diferença de nomenclatura entre tais elementos, possibilitando uma maior corretude na especificação da DSSA final.
    - **Identificação de variabilidades e opcionalidades entre os elementos arquiteturais:** uma vez que a abordagem *ArchToDSSA*

consegue identificar elementos mandatórios e opcionais, além de pontos de variação com as suas variantes, em um conjunto de arquiteturas de domínio, uma grande contribuição está sendo dada à reutilização de software, tornando mais consistentes as aplicações que serão derivadas da DSSA especificada.

2. *Uma ferramenta de apoio à abordagem ArchToDSSA, denominada ArchToDSSATool, que visa o aumento de produtividade do engenheiro de domínio na criação de uma DSSA.* O protótipo implementado possui as seguintes características:
    - **Integração a um ambiente concreto de desenvolvimento de software:** a implementação do protótipo de modo integrado ao ambiente Odyssey possibilita etapas adjacentes à comparação das arquiteturas, tais como a extração de tais arquiteturas por meio de engenharia reversa e a modelagem e manipulação da DSSA especificada. Tais atividades, embora estejam além do escopo desta proposta, complementam o processo de criação de uma arquitetura de referência em um domínio.
    - **Possibilidade de execução do protótipo de forma independente (i.e., *stand alone)*:** a alternativa de execução da *ArchToDSSATool* de forma independente traz uma grande flexibilidade ao trabalho do engenheiro, que não se atém a um ou outro ambiente de desenvolvimento.
    - **Possibilidade de integração com outras ferramentas:** a utilização do formato XMI para a comparação das arquiteturas e especificação da DSSA permite que várias ferramentas que utilizem tal formato possam ser integradas à abordagem *ArchToDSSA*.

3. *Dois estudos de observação com o objetivo de caracterizar a eficiência da abordagem proposta e do seu ferramental de apoio.* Os estudos de observação conduzidos ofereceram indícios que levam a crer que tanto a abordagem quanto o ferramental de apoio implementado são eficientes no seu propósito de auxiliar um engenheiro de domínio na criação de uma DSSA. Além disso, através dos estudos conduzidos, pôde-se perceber que

até mesmo os engenheiros menos experientes, ou com menos conhecimento do domínio, conseguem criar uma DSSA, graças ao auxílio da ferramenta ArchToDSSATool.

7.2 - Limitações

Algumas limitações puderam ser detectadas ao longo dessa pesquisa, como listado a seguir:

- **Versão da UML e XMI**: inserida na proposta do trabalho de (VASCONCELOS, 2007), que se iniciou em 2002, a abordagem utiliza a versão corrente da especificação da UML, à época, que era a 1.4, e versão XMI 1.1. Embora outras versões da UML e do XMI tenham surgido após esse momento, o ambiente Odyssey ainda trabalha com tal especificação. Assim, o arquivo XMI gerado para a DSSA, bem como modelos UML resultantes de sua manipulação, somente podem ser lidos por ambientes de desenvolvimento ou ferramentas que sejam compatíveis 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.
- **Definição arbitrária na escolha da arquitetura-base**: atualmente, a abordagem não define nenhum critério para a escolha da arquitetura que servirá de base para a DSSA final. Uma possível melhoria seria o estabelecimento de heurísticas para a escolha dessa arquitetura-base, ou mesmo a integração da abordagem com uma ferramenta de métricas, a fim de que tal arquitetura-base possa ser selecionada através de métricas estabelecidas;
- **Suporte apenas aos modelos estáticos**: atualmente, trata-se somente a comparação de modelos estáticos, i.e., diagramas de classe da UML. Em futuras pesquisas, outros modelos poderiam ser tratados, como os modelos dinâmicos, como o diagrama de sequência da UML.
- **Tratamento de elementos que sejam apenas pontos de Variação ou Variantes:** a abordagem e, por conseqüência, seu ferramental de apoio, não trata situações em que um elemento arquitetural é , ao

mesmo tempo, uma variante em relação a um elemento e um ponto de variação em relação a outros. Tal limitação foi revelada após a condução dos estudos de observação**.**

- **Limitação a Arquiteturas Orientadas a Objetos:** a abordagem e, por conseqüência, seu ferramental de apoio, suportam apenas arquiteturas de sistemas orientados a objetos. Esta limitação se dá mais em função das heurísticas usadas na detecção de pontos de variação e variantes.

7.3 - Perspectivas Futuras

Diante do trabalho aqui apresentado, abre-se uma gama de perspectivas futuras, como:

- **Granularidade**: em trabalhos futuros, pode-se definir um novo método de cálculo para as equivalências de um elemento em função das equivalências de outros elementos nele contidos. Por exemplo, a equivalência entre pacotes poderia ser inferida em função das equivalências entre suas classes. Atualmente, a comparação não leva em consideração sub-elementos.
- **Mecanismos de Avaliação:** desenvolver formas de identificar e sugerir mecanismos de avaliação que possam validar a DSSA criada, tais como inspeções, etc;
- **Refinamento da abordagem e do ferramental de apoio com base nas limitações encontradas:** baseado nas limitações já citadas anteriormente e também em novos estudos experimentais, a abordagem e seu ferramental de apoio podem ser refinados**.**
- **Identificação de Variabilidades e Opcionalidades em termos de métodos e atributos:** mecanismos que pudessem identificar a opcionalidade e variabilidade dos métodos e atributos das classes podem ser pesquisados, tornando a comparação e a criação da DSSA mais precisa**.**
- **Utilização de outros tipos de algoritmos como alternativas para a detecção de similaridades:** alternativas tais como lógica fuzzy, modelo probabilístico, ontologias e até mesmo algoritmos para

tratamento de palavras truncadas, como apresentado em (DURAN, 1999) podem ser estudadas para auxiliar a identificação de semelhanças e diferenças entre elementos arquiteturais.

- **Identificação de novos tipos de comparações:** dentre os aspectos de uma arquitetura que poderiam ser comparados, estamos delimitando nosso escopo aos elementos arquiteturais como pacotes e classes. Em trabalhos futuros, outras partes das arquiteturas podem ser usadas no critério de comparação como conexões, padrões etc.
- **Verificação da escalabilidade da abordagem:** novos estudos experimentais podem ser realizados com o intuito de se obter dados mais precisos a respeito da quantidade de arquiteturas suportadas pela abordagem, bem como o tamanho destas arquiteturas.
- **Utilização de um dicionário de referência:** a partir de estudos mais aprofundados a respeito da comparação de palavras, um dicionário de referência no idioma predominante das arquiteturas analisadas (inglês, português,...) poderia ser usado na abordagem.
- **Escolha da arquitetura-base:** a escolha da arquitetura-base poderia ser realizada através do uso de métricas, conforme mencionado anteriormente.

À medida que se for obtendo um maior *feedback* através da utilização da abordagem como um todo, outras melhorias podem ser identificadas e realizadas, contribuindo assim para o avanço das pesquisas em Reutilização de Software, especialmente no campo de arquitetura de referência para domínios específicos.

Referências Bibliográficas

ABOWD, G., ALLEN, R., GARLAN, D., 1993, "Using style to understand descriptions of software architecture". In: *Proceedings of the 1st ACM SIGSOFT symposium on Foundations of software engineering*, pp. 9-20 Los Angeles, California, USA.

ARANGO, G., 1994, "A brief introduction to domain analysis". In: *Proceedings of the 1994 ACM symposium on Applied computing* pp. 42-46, Phoenix, Arizona, USA

ATKINSON, C., BAYER, J., BUNSE, C*., et al.*, 2002, *Component-based Product Line Engineering with UML*, Boston, Addison-Wesley Longman Publishing Co., Inc.

BABAR, M.A., ZHU, L., JEFFERY, R., 2004, "A Framework for Classifying and Comparing Software Architecture Evaluation Methods". In: *Proceedings of the Australian Software Engineering Conference*, pp. 309-318, Malbourne, Australia, April.

BARROCA, L., GIMENES, I.M.D.S., HUZITA, E.H.M., 2005, "Conceitos Básicos". In: GIMENES, I.M.S., HUZITA, E.H.M. (eds), *Desenvolvimento Baseado em Componentes: Conceitos e Técnicas*, Rio de Janeiro, Ciência Moderna.

BASILI, V., CALDIEIRA, G., ROMBACH, H., 1994, *Goal Question Metrics Paradigm, Encyclopedia of Software Engineering*, John Wiley & Sons.

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.

BOSCH, J., 2004, "Software Variability Management". In: *Proceedings of the 26th International Conference on Software Engineering (ICSE'04)*, pp. 720-721, Scotland, UK.

BRAGA, R., 2000, *Busca e Recuperação de Componentes em Ambientes de Reutilização de Software*, Tese de D.Sc., COPPE, UFRJ, Rio de Janeiro, Brasil.

BUFFENBARGER, J., 1995, "Syntactic Software Merging". In: *Selected papers from the ICSE SCM-4 and SCM-5 Workshops, on Software Configuration Management* pp. 153-172 Seattle, Washington, USA.

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.

CLAUSS, M., 2001, "Generic Modeling using UML Extensions for Variability". In: *DSVL 2001 (OOPSLA Workshop on Domain Specific Visual Languages), Proceedings*, pp. 11-18, Finland.

CLEMENTS, P., 1994, "From Domain Models to Architectures". In: *Workshop on Software Architecture, USC Center for Software Engineering*, Los Angeles, California, USA.

CLEMENTS, P.C., 1996, "A Survey of Architecture Description Languages". In: *IWSSD '96: Proceedings of the 8th International Workshop on Software Specification and Design*, pp. 16-25, Los Alamitos, California, USA.

CLEMENTS, P.C., 1999, "Essential Product Line Practices". In: *Proceedings of the Ninth Workshop on Institutionalizing Software Reuse - WISR9*, Austin, Texas, USA, January.

CORBA, 2007, "Common Object Request Broker Archtecture". In: http://www.corba.org, accessed in 21/06/2007.

D'SOUZA, D.F., WILLS, A.C., 1999, *Objects, components, and frameworks with UML: the catalysis approach*, Addison-Wesley Longman Publishing Co., Inc.

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, Florida,USA, May.

DASHOFY, E.M., HOEK, A.V.D., TAYLOR, R.N., 2001, "A Highly-Extensible, XML-Based Architecture Description Language ". In: *Proceedings of the Working IEEE/IFIP Conference on Software Architecture (WICSA'01)*, pp. 103-112, Amsterdan, The Netherlands, August.

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.

DURAN, M.L.F., 1999, *Comparação Difusa de Classes Baseada no Comportamento*, Dissertação de MSc., COPPE, UFRJ, Rio de Janeiro, Brasil.

FUJABA, 2007, "Fujaba Suite Tool". In: www.fujaba.de, accessed in 29/05/2007.

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

GENTLEWARE. In: http://www.gentleware.com/, accessed in 17/05/2007.

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.

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.

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.

IBM, 2007, "XML Diff and Merge Tool". In: http://www.alphaworks.ibm.com/tech/xmldiffmerge, accessed in 29/05/2007.

IEEE, 1990, "Std 610.12 - IEEE Standard Glossary of Software Engineering Terminology", *Institute of Electrical and Electronics Engineers*.

JACOBSON, I., GRISS, M., JONSSON, P., 1997, *Software Reuse: architecture, process and organization for business success*, 1st ed., Addison-Wesley

JIANG, J., SYSTÄ, T., 2003 "Exploring Differences in Exchange Formats - Tool Support and Case Studies ". In: *Proceedings of the Seventh European Conference on Software Maintenance and Reengineering* pp. 389-398, Benevento, Italy.

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.

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.

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.

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.

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.

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.

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.

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.

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.

NOKIA, 2007, "Nokia Support Glossary". In: http://europe.nokia.com/support/glossaryCDAMaster/0,,lang_id=49&letter=0,00.html, accessed in 19/06/2007.

NORTHROP, L., 2002, "SEI's Software Product Line Tenets", *IEEE Software*, v. 19, n. 4 (July/August, 2002), pp. 32-40.

ODYSSEY, 2007, "Projeto Odyssey". In: http://reuse.cos.ufrj.br/odyssey, accessed in 03/05/2007.

OLIVEIRA, R.F.D., 2006, *Formalização e Verificação de Consistência na Representação de Variabilidades*, Dissertação de M.Sc., COPPE, UFRJ, Rio de Janeiro, Brasil.

OMG, 2001, *Unified Modeling Language (UML) Specification, version 1.4*, formal/01-09-67, Object Management Group.

OMG, 2005, "MetaObjectFacility (MOF) Specification - Version 1.4". In: http://www.omg.org/technology/documents/formal/mof.htm, accessed in 08/09/2005.

OMG, 2002b, *XML Metadata Interchange (XMI) Specification, version 1.2*, formal/02-01-01, Object Management Group.

OMG, 2007, "Unified Modeling Language". In: http://www.uml.org/, accessed in 29/05/2007.

PENEDO, M., RIDDLE, W., 1993, "Process-Sensitive SEE Architecture (PSEEA)", *Software Engineering Notes, ACM SIGSOFT*, v. 18, n. 3 (July), pp. A78-A94.

PERRY, D.E., WOLF, A.L., 1992, "Foundations for the study of software architecture", *SIGSOFT Softw. Eng. Notes*, v. 17, n. 4 (1992), pp. 40-52.

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.

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.

SAMETINGER, J., 1997, *Software Engineering with Reusable Components*, Springer-Verlag New York, Inc.

SEI/CMU, 2007, "A Framework for Software Product Line Practice - Version 4.2". In: http://www.sei.cmu.edu/productlines/framework.html, accessed in 21/06/2007.

SHAW, M., GARLAN, D., 1996, *Software Architecture: Perspectives on an Emerging Discipline*, New Jersey, Prentice-Hall.

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.

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.

VASCONCELOS, A.P.V., 2004, *Uma Abordagem para Recuperação de Arquitetura de Software Visando sua Reutilização em Domínios Específicos*, Exame de Qualificação, COPPE, UFRJ, Rio de Janeiro, Brasil.

VASCONCELOS, A.P.V., 2007, *Uma Abordagem de Apoio à Criação de Arquiteturas de Referência de Domínio Baseada na Análise de Sistemas Legados*, Tese de DSc., 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.

WALLNAU, K.C., HISSAM, S.A., SEACORD, R.C., 2002, *Building Systems from Commercial Components*, 1st ed., Boston, USA, Addison-Wesley.

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.

WESTHUIZEN, C.V.D., HOEK, A.V.D., 2002, "Understanding and Propagating Architecutural Changes ". In: *Proceedings of the IFIP 17th World Computer Congress - TC2 Stream / 3rd IEEE/IFIP Conference on Software Architecture: System Design, Development and Maintenance*, pp. 95-109 Montréal, Québec, Canada, August.

WOHLIN, C., RUNESON, P., HÖST, M*., et al.*, 2000, *Experimentation in Software Engineering: an Introduction*, USA, Kluwer Academic Publishers.

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.

XML, 2007, "Extended Markup Language". In: http://www.xml.org, accessed in 11/06/2007.

Anexo I: Formulário de Consentimento

CRIAÇÃO DE UMA ARQUITETURA DE REFERENCIA PARA UM DOMÍNIO

Eu declaro ter mais de 18 anos de idade e concordar em participar de um estudo conduzido por Guilherme Zanetti Kümmel, como parte das atividades para obtenção do título de Mestre, no Programa de Engenharia de Sistemas e Computação da COPPE/UFRJ. Este estudo visa compreender a aplicabilidade da abordagem ArchToDSSA no apoio na tarefa de criação de uma arquitetura de referencia no domínio;

PROCEDIMENTO

Este estudo acontecerá em duas etapas. Primeiro é apresentado o aparato necessário para a criação de uma arquitetura de referência no domínio e as arquiteturas a serem analisadas, bem como instruções para a realização da tarefa, documentos que serão utilizados neste estudo. Na segunda etapa os participantes deverão utilizar o aparato apresentado previamente para a criação de uma arquitetura de referência. Eu entendo que, uma vez o experimento tenha terminado, os trabalhos que desenvolvi, serão estudados visando entender a eficiência dos procedimentos e as técnicas que me foram ensinadas.

CONFIDENCIALIDADE

Toda informação coletada neste estudo é confidencial, e meu nome não será identificado em momento algum. Da mesma forma, me comprometo a não comunicar os meus resultados enquanto não terminar o estudo, bem como manter sigilo das técnicas e documentos apresentados e que fazem parte do experimento.

BENEFÍCIOS, LIBERDADE DE DESISTÊNCIA

Eu entendo que os benefícios que receberei deste estudo são limitados ao aprendizado do material que é distribuído e ensinado, independente de participar ou não deste estudo, mas que os pesquisadores esperam aprender mais sobre quão eficiente é a utilização da abordagem ArchToDSSA para a criação de uma arquitetura de referencia no domínio.

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

Guilherme Zanetti Kummel

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:**_____

Anexo II: Leitura Introdutória

Na atividade de projeto de qualquer sistema, a especificação de uma arquitetura de software representa um dos primeiros passos. Segundo a literatura, um das definições para arquitetura de software é:

> *"A arquitetura de um sistema consiste da(s) estrutura(s) de suas partes (incluindo as partes de software e hardware envolvidas no tempo de execução, projeto e teste), da natureza e das propriedades relevantes que são externamente visíveis destas partes (módulos com interfaces, unidades de hardware, objetos), e dos relacionamentos e restrições entre elas"* (D'SOUZA & WILLS, 1999).

No contexto da reutilização de software, quando as arquiteturas são especificadas através de elementos especializados no Domínio, generalizados para uso efetivo dentro do domínio e compostos em um padrão de estrutura eficaz para a construção de uma família de aplicações, elas são definidas como a **Arquitetura de Referência** do domínio ou **DSSA** (Domain Specific Software Architecture).

A criação de uma DSSA é importante nas abordagens de reutilização como Engenharia de Domínio e Linha de Produtos de Software, pois ela é a base para a instanciação de aplicações em um domínio de aplicação específico. Um de seus principais objetivos é propor uma solução estruturada para a construção de uma família de aplicações capazes de atender aos requisitos de um determinado domínio. Através da reutilização de arquiteturas, o compartilhamento de idéias, métodos, componentes e conhecimento dentro de um conjunto de aplicações similares, tornam-se mais efetivos para se obter níveis superiores de qualidade e produtividade durante o processo de construção de sistemas.

Uma maneira de se especificar uma arquitetura de referência é examinando-se e comparando-se várias arquiteturas existentes para um mesmo domínio. Para tanto, os sistemas legados constituem uma das maiores fontes.

No estudo de observação que você irá participar, é utilizada uma abordagem, isto é, um método para a especificação de arquitetura de referência (DSSA) a partir da comparação de arquiteturas existentes. Tal abordagem é denominada *ArchToDSSA* e tem o objetivo de apoiar o Engenheiro de domínio na especificação da DSSA, e conseqüentemente na reutilização de software.

A abordagem ArchToDSSA propõe uma série de passos para se chegar à DSSA, tal com descrito a seguir:

1. Identificação de equivalências e opcionalidade.

O primeiro passo é identificação de elementos equivalentes nas diferentes arquiteturas, aqui denominados equivalências. Uma equivalência é identificada para se unificar as nomenclaturas de elementos arquiteturais, que na maioria das vezes não são especificados pela mesma equipe ou empresa. Assim, um match possibilita, por exemplo, que um elemento que em uma arquitetura é denominado "Caixa" seja equivalente a um elemento que foi denominado "Cx." em outra arquitetura.

Identificados as equivalências é possível detectar se cada elemento na arquitetura será opcional ou mandatório na DSSA. A definição de elemento opcional e mandatório em um domínio pode ser encontrada em (OLIVEIRA, 2006), como segue:

Elementos Opcionais: elementos que podem ou não estar presentes em aplicações instanciadas a partir de um domínio.

Elementos Mandatórios: elementos que devem obrigatoriamente estar presentes em aplicações instanciadas a partir de um domínio.

Se para cada elemento da arquitetura, existir um elemento equivalente em todas as outras arquiteturas, esse elemento é considerado mandatório. Caso contrário, é considerado opcional na DSSA.

2. Identificação dos Pontos de Variação e Variantes

A definição de ponto de variação (VP) e variantes pode também ser encontrada em (OLIVEIRA, 2006). Resumindo:

Pontos de Variação: pontos que refletem a parametrização (possibilidade de configuração) no domínio de uma maneira abstrata e são configuráveis através das variantes.

Variantes: alternativas de implementação disponíveis, elementos necessariamente ligados a um ponto de variação, que atuam como alternativas para se configurar aquele ponto de variação.

Para uma DSSA é fundamental especificar quais elementos arquiteturais deverão estar obrigatoriamente em todas as aplicações derivadas e quais podem ser suprimidos. Para tanto, a abordagem ArchToDSSA sugere a identificação de tais elementos da seguinte forma:

Pontos de Variação	Variantes
Interfaces	Elementos que implementam a Interface
Generalizações (Superclasses)	Especializações (Subclasses)

Assim, se, por exemplo, uma Interface for selecionada como parte de uma DSSA, provavelmente ela será um ponto de variação nesta arquitetura e as classes que a implementam devem também ser selecionadas como suas variantes.

1. Criação da DSSA

Uma vez identificados todos os elementos opcionais e mandatórios, pontos de variação e suas variantes e os elementos equivalentes nas arquiteturas, o próximo passo é especificar a DSSA. Nesse ponto, sugere-se o seguinte processo:

- Escolher uma das arquiteturas comparadas para servir como base. Isto é necessário para se manter os relacionamentos existentes entre os elementos arquiteturais de maneira consistente.
- Levar para a DSSA todos os elementos mandatórios de todas as arquiteturas comparadas. No entanto, quando os elementos possuem equivalência com outros elementos, apenas um deles deve ser levado.
- Levar para a DSSA todos os pontos de variação e suas variantes. No entanto, se um ponto de variação possui pontos de variação equivalentes em outras arquiteturas, mas possui variantes diferentes, todas as variantes devem ser levadas, mantendo-se seu atributo de opcionalidade.

Exemplo de Utilização da Abordagem

Como exemplo de utilização da abordagem, serão apresentadas partes de duas arquiteturas do domínio de MP3Players, que serão comparadas no intuito de originar uma DSSA.

.

Arquitetura 1	Arquitetura 2
• MP3Player_1	• MP3Player_2
○ Diversos	○ Diversos
▪ Equalizer	▪ Equalizador
▪ GerenteMemoAM/FM	▪ GerenteMemoriaRadio
• MemoAM	• MemoriaAM
• MemoFM	• MemoriaFM
▪ GerenteSom	▪ Display
• MegaBassDigital	• I3DTag
• AVLS	• Simples
▪ Microfone	○ Principal
○ Principal	▪ MP3Player
▪ MP3Player	○ GerenteConexao
○ GerenteConexao	▪ ConexaoUSB
▪ ConexaoUSB	• HiSpeedUSB
• HiSpeedUSB	• DirectUSB
• DirectUSB	○ GerenteReproducao
○ GerenteReproducao	▪ FormatoReproducao
▪ Reproducao	• MP3
• WMA	• WMA
• Mp3	• Atrac3Plus

No caso das duas arquiteturas, pode-se identificar alguns elementos que, embora tenham nomes diferentes, possuem o mesmo significado, tais como **Equalizador = Equalizer**, **FormatoReprodução = Reprodução** e **GerenteMemoriaRadio = GerenteMemoAM/FM**. Assim, essas equivalências poderiam ser representadas por *matches*, denominados aqui **Equalizador**, **Reprodução** e **MemoriaRadio**, por exemplo.

Para identificar a opcionalidade dos elementos, deve-se observar se cada elemento possui equivalência com todas as outras arquiteturas. Nesse caso, os elementos participantes dos *matches* acima citados seriam mandatórios, enquanto outros elementos, tais como **Microfone** e **AVLS** (Sistema de limitação automática de volume), na Arquitetura 1, são elementos opcionais.

Como exemplo de pontos de variação nessas arquiteturas, tem-se o **FormatoReproducao**, que possui as variantes **MP3**, **WMA** e **Atrac3Plus**, **GerenteMemoriaRadio**, com as variantes **MemoriaAM** e **MemoriaFM** e e **ConexaoUSB**, com as variantes **HiSpeedUSB** e **DirectUSB**.

Tomando a Arquitetura 1 como arquitetura-base, a DSSA sugerida, segundo a abordagem seria a seguinte:

- **DSSA**
  - **Diversos**
    - **Equalizer (M)**
    - **GerenteMemoAM/FM (M)**
      - **MemAM (M)**
      - **MemoFM (M)**
    - **GerenteSom (O)**
      - **MegaBassDigital (O)**
      - **AVLS (O)**
    - **Microfone (O)**
  - **Principal**
    - **MP3Player (M)**
  - **GerenteConexao**
    - **ConexaoUSB (M)**
      - **HiSpeedUSB (M)**
      - **DirectUSB (M)**
  - **GerenteReproducao (M)**
    - **Reproducao (M)**
      - **MP3 (M)**
      - **WMA (M)**
      - **Atrac3Plus (O)**
  - **Display (O)**
    - **I3Dtag (O)**
    - **Simples (O)**

No exemplo acima, os elementos em preto fazem parte da Arquitetura 1, arquitetura-base, os elementos em azul foram trazidos da Arquitetura 2. Os sinais (M) e (O) significam, respectivamente, mandatório e opcional. Note que es elementos considerados mandatórios são aqueles que possuem equivalências nas duas arquiteturas. Os demais são considerados opcionais.

Anexo III: Instruções para Execução dos Estudos de Observação

Avaliação da Abordagem ArchToDSSA

Primeiro Estudo de Observação – Instruções

Você participará de um estudo de observação que visa avaliar a utilização da abordagem *ArchToDSSA* para apoio à especificação de uma arquitetura de referência no domínio. Tal estudo será dividido em duas etapas, de acordo com as instruções contidas neste documento.

Você deve ter recebido juntamente com esta folha de instruções os seguintes documentos:

- Leitura Introdutória
- Representação da Arquitetura de Sistema Legado, Arquiteturas A, B e C.
- Formulário de Registro de Equivalências Identificadas
- Formulário de Registro de Pontos de Variação Identificados
- Formulário de Registro de Variantes Identificadas

IMPORTANTE: Para a realização de ambas as etapas, a leitura introdutória já deve ter sido realizada.

1ª Etapa: Criação de uma DSSA em um primeiro domínio

Nesta etapa do estudo, você deverá seguir os passos sugeridos pela abordagem e criar uma DSSA. Para isso, você deverá:

1. Examinar as arquiteturas A, B e C e identificar elementos equivalentes. Só deverão ser considerados equivalências os elementos que tiverem equivalentes nas três arquiteturas. Nomear as equivalências e identificar os elementos relacionados pelo nome da arquitetura e a linha do elemento na arquitetura. Ex. Telefone = A1, B4, C25
2. Para cada arquitetura, identificar pontos de variação e anotar no formulário apropriado.
3. Para cada arquitetura, identificar variantes e anotar no formulário apropriado.
4. Escolher uma das arquiteturas como base e montar sobre ela a DSSA resultante, escrevendo os elementos das outras arquiteturas que você julgar que devem compor a DSSA.

2ª Etapa: Criação de uma DSSA em um segundo domínio

Nesta etapa do estudo, você deverá repetir os procedimentos realizados na primeira etapa, porém trabalhando em um domínio diferente do que foi apresentado anteriormente.

Obrigado pela sua participação!

Avaliação da Abordagem ArchToDSSA
Segundo Estudo de Observação – Instruções

Você participará de um estudo de observação que visa avaliar a utilização da abordagem *ArchToDSSA e sua ferramenta de apoio, ArchToDSSATool,* para apoio à especificação de uma arquitetura de referência no domínio. Tal estudo será dividido em duas etapas, de acordo com as instruções contidas neste documento.

Você deve ter recebido juntamente com esta folha de instruções os seguintes documentos:

- Leitura Introdutória
- Arquivos no formato XMI, representando arquiteturas de sistemas legados.

IMPORTANTE: Para a realização de ambas as etapas, a leitura introdutória e o treinamento na ferramenta já devem ter sido realizados.

1ª Etapa: Criação de uma DSSA em um primeiro domínio

Nesta etapa do estudo, você deverá seguir os passos sugeridos pela abordagem e criar uma DSSA com o auxílio da ferramenta. Para isso, você deverá:

1. Identificar e validar as equivalências. Só deverão ser considerados equivalências os elementos que tiverem equivalentes nas três arquiteturas;
2. Identificar os pontos de variação;
3. Identificar as variantes;
4. Escolher a arquitetura base e selecionar os elementos das outras arquiteturas que você julgar que devem compor a DSSA.
5. Exportar a DSSA criada para um arquivo XMI.

2ª Etapa: Criação de uma DSSA em um segundo domínio

Nesta etapa do estudo, você deverá repetir os procedimentos realizados na primeira etapa, porém trabalhando em um domínio diferente do que foi apresentado anteriormente.

Obrigado pela sua participação!

Anexo IV: Formulários de Registro de Resultados

Formulário de Registro de Equivalências Identificadas

Participante:_____Domínio:_____

Nome da equivalência	Elementos Participantes

Formulário de Registro de Pontos de Variação Identificados

Participante:_____Domínio:_____

Arquiteturas	Pontos de Variação
Arquitetura A	
Arquitetura B	
Arquitetura C	

Formulário de Registro de Variantes Identificadas

Participante:_____Domínio:_____

Arquiteturas	Variantes
Arquitetura A	
Arquitetura B	
Arquitetura C	

Anexo V: Representação das Arquiteturas de Sistemas Legados

Representação da Arquitetura de Sistema Legado

Domínio: Telefonia Celular

Arquitetura A

1.	• **Telefonia_Celular_1**
2.	○ **Diversos**
3.	▪ **Idioma**
4.	▪ **Contraste**
5.	▪ **PapelPArede**
6.	▪ **Mensagem**
7.	▪ ***CaixaPostal***
8.	• **Simplificada**
9.	• **Completa**
10.	▪ **Calculadora**
11.	○ **Principal**
12.	▪ **TelefoneCelular**
13.	○ **GerenteSom**
14.	▪ ***ToquesMusicais***
15.	• **ToquesSimples**
16.	• **ToquesPolifônicos**
17.	▪ **Campainha**
18.	▪ **Alarme**
19.	○ **GerenteJogos**
20.	▪ ***Jogos***
21.	• **Snake**
22.	• **CarRacer**
23.	○ **GerenteAgenda**
24.	▪ **GrupoUsuario**
25.	▪ **Usuario**
26.	▪ **Agenda**

Legenda:

Palavras em **Negrito** representam Pacotes

Palavras em *Itálico* representam Superclasses e/ou Interfaces

Representação da Arquitetura de Sistema Legado

Domínio: Telefonia Celular

Arquitetura B

1.	**Telefonia_Celular_2**
2.	○ **GerentePlanoFundo**
3.	▪ **PlanoFundo**
4.	▪ **Lingua**
5.	○ **GerenteCaixaPostal**
6.	▪ **Mensagem**
7.	▪ ***CaixaPostal***
8.	• **Texto**
9.	• **Voz**
10.	○ **GerenteAlarme**
11.	▪ **Alarme**
12.	○ **Principal**
13.	▪ **TelefoneCelular**
14.	○ **GerenteCampainha**
15.	▪ **ToquesMusicais**
16.	▪ **Campainha**
17.	○ **GerenteEntretenimento**
18.	▪ **MP3**
19.	▪ ***Jogos***
20.	• **Paciencia**
21.	• **BlackJack**
22.	○ **GerenteUsuario**
23.	• **GrupoUsuario**
24.	• **Usuario**
25.	• **Agenda**

Legenda:

Palavras em **Negrito** representam Pacotes

Palavras em *Itálico* representam Superclasses e/ou Interfaces

Representação da Arquitetura de Sistema Legado

Domínio: Telefonia Celular

Arquitetura C

1.	**Telefonia_Celular_3**
2.	○ **Agenda**
3.	▪ **Agenda**
4.	▪ ***Usuario***
5.	• **UsuarioComum**
6.	• **UsuarioEspecial**
7.	▪ **GrupoUsuario**
8.	○ **JogosCelular**
9.	• ***Jogos***
10.	○ **Snake**
11.	○ **Pinball**
12.	○ **BlackJack**
13.	○ **Som**
14.	▪ **Despertador**
15.	▪ **Campainha**
16.	▪ **ToquesMusicais**
17.	○ **Principal**
18.	▪ **Telefone**
19.	○ **Diversos**
20.	▪ **Calculadora**
21.	▪ ***CaixaPostal***
22.	• **Simplificada**
23.	• **Completa**
24.	▪ **Mensagem**
25.	▪ **PapelParede**
26.	▪ **Contraste**
27.	▪ **Idioma**
28.	▪ **Bluetooth**

Legenda:

Palavras em **Negrito** representam Pacotes

Palavras em *Itálico* representam Superclasses e/ou Interfaces

Representação da Arquitetura de Sistema Legado

Domínio: Escolar

Arquitetura A

1. • Escola_1
2. ○ Pessoas
3. ▪ *Pessoa*
4. • *Funcionário*
5. ○ Professor
6. ○ Coordenador
7. ○ Zelador
8. • Responsável
9. • Aluno
10. ▪ Carreira
11. ▪ Área
12. ○ Estrutura
13. ▪ *Conteúdo*
14. • Primeiro Grau
15. • *Segundo Grau*
16. ○ Convencional
17. ○ Pism
18. • *Cursinho*
19. ○ Extensivo
20. ○ Intensivo
21. ○ Recordação
22. ▪ Turma
23. ▪ Sala
24. ▪ Matéria
25. ○ Provas
26. ▪ Provas Abertas
27. ▪ Provas Fechadas
28. ▪ *Modalidades de Prova*
29. • *Provas Convencionais*
30. ○ Pism
31. ○ Bimestrais
32. • Simulados
33. ○ Matrícula
34. ▪ *Matrícula*
35. • Matrícula Primeiro Grau e Segundo Grau
36. • Matrícula Simulado
37. • Matrícula Cursinho
38. ○ Aulas
39. ▪ *Aula*
40. • Aulas primeiro grau e segundo grau
41. • Aulas Cursinho
42. • Aulas Especiais

Legenda:

Palavras em **Negrito** representam Pacotes

Palavras em *Itálico* representam Superclasses e/ou Interfaces

Representação da Arquitetura de Sistema Legado

Domínio: Escolar

Arquitetura B

1. • **Escola_2**
2. ○ **Pessoas**
3. ▪ ***Pessoa***
4. • ***Pessoa Física***
5. ○ **Supervisor**
6. ○ **Docente**
7. ○ **Aluno**
8. • **Pessoa Jurídica**
9. ▪ **Carreira**
10. ▪ **Área**
11. ○ **Estrutura**
12. ▪ ***Programação***
13. • ***Pré-Vestibular***
14. ○ **Reforço**
15. ○ **Intenso**
16. • **Fundamental**
17. • ***Médio***
18. ○ **Padrão**
19. ○ **Programa Ingresso seletivo Misto**
20. ▪ **Disciplinas**
21. ○ **Provas**
22. ▪ **Provas Dissertativas**
23. ▪ **Provas Múltipla Escolha**
24. ▪ ***Avaliações***
25. • ***Prova Padrão***
26. ○ **Provas do Programa de Ingresso Seletivo misto**
27. ○ **Provas Bimestrais**
28. • **Provas de Simulação**
29. ○ **Matrícula**
30. ▪ ***Inscrição***
31. • **Inscrição Pré-Vestibular**
32. • **Inscrição Fundamental e Médio**
33. • ***Inscrição Simulado***
34. ○ **Simulado Aberto**
35. ○ **Simulado Fechado**
36. ○ **Aulas**
37. ▪ ***Aula***
38. • **Aulas fundamental e médio**
39. • **Aulas Pré-Vestibular**
40. ▪ **Sala**

Legenda:

Palavras em **Negrito** representam Pacotes

Palavras em *Itálico* representam Superclasses e/ou Interfaces

Domínio: Escolar

Arquitetura C

1. • Escola_3
2. ○ Pessoas
3. ▪ *Pessoa*
4. • *Funcionário*
5. ○ Educadores
6. ○ Zelador
7. ○ Diretor
8. • Responsável Estudante
9. • Estudante
10. ○ Estrutura
11. ▪ *Conteúdo Programático*
12. • Ensino Fundamental
13. • Ensino Médio
14. • *Vestibular*
15. ○ Completo
16. ○ *Compacto*
17. ○ Recordação
18. • Cursos de Férias
19. ▪ Matéria
20. ○ Provas
21. ▪ *Provas*
22. • *Provas para fundamental e médio*
23. ○ Provas Pism
24. ○ Provas Bimestrais
25. • *Provas Pré-Vestibular*
26. ○ Dissertativas
27. ○ Fechadas
28. ○ Matrícula
29. ▪ *Matrícula*
30. • Matrícula primeiro e segundo grau
31. • Matrícula Pré-Vestibular
32. ▪ *Matrícula Prova Simulado*
33. ○ Matrícula Simulado Fechado
34. ○ Matrícula Simulado Aberto
35. ○ Aulas
36. ▪ *Aula*
37. • Aulas do ensino fundamental e médio
38. • Aulas para Vestibular
39. • Aulas recordação

Legenda:

Palavras em **Negrito** representam Pacotes

Palavras em *Itálico* representam Superclasses e/ou Interfaces

Anexo VI: Dicionário de Sinônimos do Domínio

Domínio Telefonia Celular

- Idioma = Língua
- TelefoneCelular = Telefone
- Alarme = Despertador
- PapelParede = PlanoFundo
- Caixa = Cx
- Jogo = Game
- Tela = Display
- Conexão = Ligação
- Toque = RingTone

Palavras ignoradas

- Gerente
- View

Domínio Escolar

- Professor = Docente = Educadores
- Turma = Sala
- Aulas primeiro grau e segundo grau = Aulas fundamental e médio = Aulas do ensino fundamental e médio
- Aulas cursinho = Aulas pré-vestibular = Aulas para vestibular
- Aulas especiais = Aulas de reforço = Aulas recordação
- Matéria = Disciplinas
- Conteúdo = Programação = Conteúdo programático
- Primeiro grau = Fundamental = Ensino fundamental
- Segundo grau = Médio = Ensino médio
- Convencional = Padrão = Ensino Convencional
- Pism = Programa de ingresso seletivo misto = Ensino Pism
- Cursinho = Pré-vestibular = Vestibular
- Especial = Reforço = Recordação
- Extensivo = Estendido = Completo
- Intensivo = Intenso = Compacto

- Matrícula = Inscrição
- Matrícula simulado = Inscrição simulado = Matrícula prova simulado
- Matricula primeiro grau e segundo grau = Inscrição fundamental e médio = Matricula primeiro e segundo grau
- Matrícula cursinho = Inscrição pré-vestibular = Matrícula pré-vestibular
- Funcionário = Pessoa física
- Coordenador = Supervisor = Diretor
- Zelador = Servente
- Aluno = Estudante
- Responsável = Responsável estudante
- Modalidades de prova = Avaliações = Provas
- Provas convencionais = Prova padrão = Provas para fundamental e médio
- Bimestrais = Provas bimestrais
- Prova Pism = Provas do Programa de ingresso seletivo misto
- Prova fechada = Provas múltipla escolha = Fechada
- Provas abertas = Provas dissertativas = Dissertativas
- Simulados = Prova de simulação = Prova pré-vestibular
- Simulado fechado = Matrícula simulado fechado
- Simulado aberto = Matrícula simulado aberto
- Prova = Provas

Palavras ignoradas

- Aulas

Anexo VII: Formulário de Caracterização dos Participantes

Caracterização do Participante

Nome _____

Nível (Graduação Mestrado, Doutorado): _____

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 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

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 de Escolas
 Experiência no domínio de Telefonia Celular

Anexo VIII: Avaliação Subjetiva

Avaliação do Primeiro Estudo de Observação

1) Os conceitos de Arquiteturas e DSSA já eram de seu conhecimento antes do estudo realizado? E depois?

2) Qual o grau de dificuldade ou facilidade em realizar as etapas sugeridas? Explique.

3) A abordagem de alguma forma o ajudou na comparação de arquiteturas e criação da DSSA?

4) Como a abordagem poderia ser melhorada?

5) Acredita que uma ferramenta iria ajudá-lo na execução das tarefas? De que forma?

Avaliação do Segundo Estudo de Observação

1) Os conceitos de Arquiteturas e DSSA já eram de seu conhecimento antes do estudo realizado? E depois?

2) Qual o grau de dificuldade ou facilidade em realizar as etapas sugeridas? Explique.

3) A Ferramenta de alguma forma o ajudou na comparação de arquiteturas e criação da DSSA?

4) Como a ferramenta poderia ser melhorada?

5) Como seria este processo sem a ferramenta? Fácil, Difícil, Rápido, Demorado... Explique

6) A ferramenta apresentada pode ser utilizada em alguma etapa do desenvolvimento de software da sua área de trabalho?