UNIVERSIDADE FEDERAL DE UBERLÂNDIA
FACULDADE DE COMPUTAÇÃO
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

ASPECTOS INICIAIS MODELADOS COM UMA EXTENSÃO DA SYSML

KÊNIA SANTOS DE OLIVEIRA

Uberlândia - Minas Gerais
2013

UNIVERSIDADE FEDERAL DE UBERLÂNDIA
FACULDADE DE COMPUTAÇÃO
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

KÊNIA SANTOS DE OLIVEIRA

ASPECTOS INICIAIS MODELADOS COM UMA EXTENSÃO DA SYSML

Dissertação de Mestrado apresentada à Faculdade de Computação da Universidade Federal de Uberlândia, Minas Gerais, como parte dos requisitos exigidos para obtenção do título de Mestre em Ciência da Computação.

Área de concentração: Engenharia de Software.

Orientador:
Prof. Dr. Michel dos Santos Soares

Uberlândia, Minas Gerais
2013

UNIVERSIDADE FEDERAL DE UBERLÂNDIA
FACULDADE DE COMPUTAÇÃO
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Os abaixo assinados, por meio deste, certificam que leram e recomendam para a Faculdade de Computação a aceitação da dissertação intitulada "**Aspectos Iniciais Modelados com uma Extensão da SysML**" por **Kênia Santos de Oliveira** como parte dos requisitos exigidos para a obtenção do título de **Mestre em Ciência da Computação**.

Uberlândia, 19 de Fevereiro de 2013

Orientador: _____

Prof. Dr. Michel dos Santos Soares
Universidade Federal de Uberlândia

Banca Examinadora:

_____

Prof. Dr. Marcelo de Almeida Maia
Universidade Federal de Uberlândia

_____

Prof. Dr. Marco Tulio de Oliveira Valente
Universidade Federal de Minas Gerais

UNIVERSIDADE FEDERAL DE UBERLÂNDIA
FACULDADE DE COMPUTAÇÃO
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Data: Fevereiro de 2013

Autor:	**Kênia Santos de Oliveira**
Título:	**Aspectos Iniciais Modelados com uma Extensão da SysML**
Faculdade:	**Faculdade de Computação**
Grau:	**Mestrado**

_____

Autor

Agradecimentos

Agradeço

A Deus pela força, proteção e amparo em todos os momentos.

A minha família, especialmente aos meus pais Dozinete de Oliveira e Maria da Luz Santos Rezende pela dedicação e confiança.

Aos meus irmãos Clayton e Cássio, minhas cunhadas Zélia e Sirlene, e meus sobrinhos Adriell, Raquell e Emanuelly pelo carinho.

Ao meu namorado José Ricardo de Oliveira pelo amor e compreensão.

Aos meus amigos da pós-graduação Fabíola, Joyce, Leiliane, Lídia e Cícero pelo companheirismo e incentivo. A todos os outros colegas da pós-graduação que de uma forma ou de outra compartilharam diferentes momentos.

Ao professor Michel dos Santos Soares pelo profissionalismo, apoio e orientação em todos os momentos da realização dessa pesquisa.

A todos os professores da FACOM que colaboraram para minha formação.

Ao técnico administrativo da pós-graduação Erisvaldo Araujo Fialho por sua sempre cordialidade.

A Universidade Federal de Goiás - Campus Catalão, especialmente a equipe do Departamento de Química, por ter concedido meu afastamento para a conclusão do mestrado. Este apoio foi fundamental para a realização deste trabalho.

Resumo

A Programação Orientada a Aspectos foi proposta com o objetivo de manipular interesses transversais de uma maneira eficiente. Propostas iniciais nesta área foram aplicadas no código fonte. Posteriormente, aspectos foram considerados para serem aplicados em outras fases do desenvolvimento de software tais como Engenharia de Requisitos e Arquitetura de Software. Há várias vantagens em identificar aspectos no nível de requisitos e no nível arquitetural, tais como detectar inicialmente conflitos de interesses, melhorar a modularidade dos requisitos, reduzir custos de manutenção de software e preservar a noção de aspectos no processo de desenvolvimento de software garantindo rastreabilidade. Portanto, o propósito desse trabalho é desenvolver um modelo para representar aspectos no nível de requisitos e no nível arquitetural. O modelo de requisitos define as atividades de identificação de requisitos aspectuais tanto de origem funcional quanto não-funcional, separação e composição de requisitos e requisitos aspectuais e identificação de conflitos entre requisitos aspectuais. Uma vez que diferentes *stakeholders* necessitam visualizar o sistema a partir de diferentes perspectivas, o modelo de arquitetura permite representar diferentes visões considerando a representação com aspectos. As visões propostas são a estrutural, a de casos de uso + requisitos, e a de desenvolvimento. Em comparação com outras abordagens analisadas, os modelos propostos nesse trabalho cobrem importantes características que os outros modelos não cobrem, como por exemplo, manter a rastreabilidade de aspectos entre os níveis de requisitos e de arquitetura. Para representar os modelos, extensões da linguagem de modelagem SysML foram propostas.

Palavras chave: Aspectos, Engenharia de Requisitos, Arquitetura de Software, SysML.

Abstract

Aspect Oriented Programming has been proposed in order to handle crosscutting concerns in an efficient manner. Initial proposals in this area have been applied to the source code. Subsequently, aspects were considered to be implemented in other phases of software development such as Requirements Engineering and Software Architecture. There are several advantages in identifying aspects at the requirements level and architecture level such as detecting conflicts of interest, improving the requirements modularity, reducing costs of software maintenance and preserving the notion of aspects in software development process ensuring traceability. Therefore, the purpose of this work is to develop a model to represent aspects at the requirements level and the architecture level. The requirements model defines the activities of identification of aspect requirements, both functional and non-functional, separation and composition of aspect requirements and identification of conflict between aspect requirements. Since different stakeholders need to view the system from different perspectives, the architecture model allows to represent different views considering the representation with aspects. The proposed views are structural, use case + requirements and development. Compared to other analysed approaches, the proposed models in this work represent important characteristics that others models do not represent, such as maintaining traceability of aspects between requirements and the architecture level. In order to represent the models, extensions to the SysML modeling language were proposed.

Keywords: Aspect, Requirements Engineering, Software Architecture, SysML.

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Abreviaturas e Siglas

AORE	Aspect Oriented Requirements Engineering
AOSD	Aspect Oriented Software Development
BR	Brasil
HW	Health Watcher
MG	Minas Gerais

Capítulo 1

Introdução

Neste primeiro capítulo serão abordados assuntos introdutórios e relevantes para a contextualização do trabalho. Na Seção 1.1 é realizada a contextualização sobre o desenvolvimento de software orientado a aspectos. Na Seção 1.2 os objetivos do trabalho são descritos. Na Seção 1.3 é realizada uma revisão correlata da literatura. Na Seção 1.4 é apresentada a metodologia utilizada para a realização do trabalho. Na Seção 1.5 a visão geral da dissertação é apresentada.

1.1 Contextualização sobre o Desenvolvimento de Software Orientado a Aspectos

Para criar softwares que são mais fáceis de implementar, entender, verificar e evoluir deve-se decompor o mesmo em módulos de tal forma que cada módulo esconde um interesse (comportamento particular) do software [Parnas 1972]. A atividade de localizar interesses em módulos bem separados é conhecida como separação de interesses (*separation of concerns*) [Reade 1989], [Dijkstra 1997]. Essa decomposição leva a módulos que são fracamente acoplados e podem ser mais eficientemente implementados, entendidos e verificados.

Com o aumento da complexidade das aplicações de software é necessário que os mesmos tratem uma variedade de interesses computacionais. Desse modo, a separação de interesses tem sido identificada como um dos princípios fundamentais da Engenharia de Software [Tarr et al. 1999]. A separação de interesses pode fornecer muitos benefícios para a engenharia de software, incluindo redução da complexidade, melhora na reutilização, e evoluções mais simples.

Com o propósito de obter especificações de software modularizadas e com o correto particionamento dos seus interesses, surgiu um novo paradigma de desenvolvimento de software denominado Desenvolvimento de Software Orientado a Aspectos (*Aspect Oriented Software Development - AOSD*) [Filman et al. 2004], originado a partir da Programação

Orientada a Aspectos (*Aspect Oriented Programming*) [Kiczales et al. 1997]. O AOSD utiliza-se do princípio da separação de interesses em todos os estágios do ciclo de vida do software aumentando sua interdependência [Pinto et al. 2011].

A motivação para a Programação Orientada a Aspectos é derivada dos problemas causados por espalhamento (*scattering*) e entrelaçamento (*tangling*) de código [Kiczales et al. 1997]. Esses problemas comprometem o desenvolvimento de software, como por exemplo, a compreensão e a evolução [Tarr et al. 1999]. A separação de interesses ajuda a eliminar o problema de espalhamento e entrelaçamento por manter características distintas em unidades separadas.

Inicialmente pesquisas em AOSD focaram principalmente no nível de implementação, porém o conceito tem sido aplicado nos estágios iniciais do desenvolvimento de software como Engenharia de Requisitos e Arquitetura de Software [Rashid et al. 2006a], [Sánchez et al. 2010].

A Engenharia de Requisitos [Juristo et al. 2002], [Damian et al. 2004] e a Arquitetura de Software [Nakagama et al. 2011] são consideradas cruciais no desenvolvimento de software. A Engenharia de Requisitos é uma fase fundamental da Engenharia de Software [Yu 1997]. Ela atua como um ponto de partida para projetar a Arquitetura de Software. O projeto de arquitetura é útil para modularizar o software assim como estabelecer os principais componentes e suas interfaces. Ter uma arquitetura de software bem definida é fundamental para desenvolver, compreender e evoluir o software [Vliet 2008].

A Engenharia de Requisitos e a Arquitetura de Software ajudam os desenvolvedores a avaliar a complexidade do software com o objetivo de obter um documento de requisitos e uma configuração arquitetural que facilitam a descoberta de eventuais problemas ainda nas fases iniciais do desenvolvimento, facilitando também a evolução e o gerenciamento da manutenção do software. A Engenharia de Requisitos e a Arquitetura de Software não são fases independentes, ao invés disso são fases fortemente interligadas [Avgeriou et al. 2011], [Sommerville 2010].

Algumas abordagens de aspectos iniciais [Baniassad et al. 2006], [Conejero et al. 2010] tem sido propostas, porém muitas pesquisas ainda podem ser realizadas nessa área, como por exemplo incluir aspectos no nível de requisitos, incluir aspectos no nível de arquitetura e realizar o relacionamento entre essas duas fases com aspectos.

Existem diversas linguagens de modelagem que permitem representar requisitos e arquitetura de software. Uma dessas linguagens é a SysML [OMG-SysML 2010]. A SysML permite uma modelagem de propósito geral para aplicação em engenharia de sistemas, suportando a especificação, análise, projeto, verificação e validação de um amplo espectro de sistemas e sistemas de sistemas [Friedenthal et al. 2012]. A SysML possui um diagrama específico para modelar requisitos individuais bem como seus relacionamentos, o que não acontece em outras linguagens de modelagem. Por exemplo, para modelar requisitos na UML (*Unified Modeling Language*) [OMG-UML 2011] são utilizados Casos de Uso. A

principal questão com esta abordagem é que Casos de Uso são específicos para modelar cenários de requisitos e não requisitos individuais. Os diagramas de Requisitos da SysML também podem aparecer em outros modelos com o objetivo de representar rastreabilidade entre requisitos e o projeto de software. Essa abordagem facilita a representação do relacionamento entre requisitos e arquitetura de software. A SysML ainda não possui extensões para a modelagem de aspectos, porém as mesmas podem ser criadas uma vez que a SysML é uma linguagem de modelagem extensível.

1.2 Objetivos do Trabalho

Diante da importância em considerar aspectos nos estágios iniciais do desenvolvimento de software, esse trabalho tem como objetivo geral estabelecer uma relação da Engenharia de Requisitos com a Arquitetura de Software considerando a representação com aspectos, e estender a linguagem de modelagem SysML para representá-los. Desse modo, é definido um metamodelo para a especificação dos aspectos no nível de requisitos e no nível arquitetural.

Assim, objetiva-se obter soluções para as seguintes questões de pesquisa:

- Q1 - Quais benefícios a introdução de aspectos traz no projeto de arquitetura de software, considerando requisitos não-funcionais?

  A resposta para essa questão é crucial para saber se a introdução de aspectos no projeto de arquitetura de software para gerenciar interesses transversais é vantajosa. No Capítulo 3 é apresentada uma resposta para essa questão. Por meio de uma revisão sistemática da literatura, são avaliados quais são os interesses mais comuns quando é projetada uma arquitetura de software utilizando aspectos. Também é avaliado se a introdução de aspectos é útil na manutenção de software.

- Q2 - Como especificar aspectos na modelagem de requisitos utilizando o diagrama de Requisitos da SysML?

  Na Engenharia de Requisitos também é necessário identificar e implementar diferentes interesses que podem estar espalhados pelos artefatos de Engenharia de Requisitos. Assim, é fundamental conhecer como estes interesses serão representados na modelagem. Outra questão importante é permitir que requisitos sejam rastreáveis para a Arquitetura de Software. Utilizando extensões no diagrama de Requisitos da SysML, uma solução para estas questões é apresentada no Capítulo 4.

- Q3 - Como especificar uma arquitetura de software em múltiplas visões que permite representar aspectos utilizando a SysML como linguagem de modelagem?

O arquiteto de software necessita de diferentes visões da arquitetura de software para usos e usuários diferentes [Perry e Wolf 1992]. Desse modo, a utilização de uma arquitetura em múltiplas visões é fundamental. O Capítulo 5 apresenta uma arquitetura que permite a especificação de aspectos em diferentes visões. A representação é realizada utilizando extensões da SysML.

Desse modo, com o propósito de desenvolver um modelo que represente aspectos considerando as questões citadas, os objetivos específicos deste trabalho são:

- incluir aspectos na modelagem de requisitos de software em diferentes níveis de abstração;
- estabelecer a relação dos requisitos transversais na Arquitetura de Software;
- utilizar a SysML como linguagem de modelagem em nível de requisitos e de arquitetura;
- avaliar a aplicação dos conceitos de aspectos em um estudo de caso aplicando o modelo de representação de aspectos obtido.
- estender a ferramenta ArgoUML para representar os modelos obtidos.

1.3 Revisão da Literatura Correlata

O AOSD pode ser considerado um novo paradigma de desenvolvimento de software que complementa os outros paradigmas, principalmente o de orientação a objetos, com o propósito de melhorar o processo de desenvolvimento de software através de melhor separação de interesses. De acordo com Dijkstra [Dijkstra 1997], no princípio da separação de interesses, cada interesse que é relevante para uma aplicação é melhor tratado separadamente de outro interesse. Essa divisão fornece melhores resultados e oferece vantagens como redução da complexidade e melhora no reuso dos artefatos de software.

Abordagens de aspectos iniciais foram propostas, incluindo Engenharia de Requisitos Orientada a Aspectos (*Aspect Oriented Requirements Engineering* - AORE) [Grundy 1999], [Moreira et al. 2002] bem como Arquitetura de Software Orientada a Aspectos (*Software Architecture Aspect Oriented* - SAAO) [Perez et al. 2006], [Navasa et al. 2009]. São várias as vantagens de se identificar e gerenciar aspectos iniciais, como por exemplo, melhorar a modularidade nos requisitos e no projeto de arquitetura, detectar inicialmente conflitos de interesses, reduzir custos de manutenção do software e preservar a noção de aspectos no processo de desenvolvimento garantindo a rastreabilidade [Baniassad et al. 2006], [Conejero et al. 2012]. Segundo [Baniassad et al. 2006], conhecer aspectos no nível de requisitos ajuda o arquiteto de software a projetar um sistema melhor, e conhecer aspectos no nível arquitetural ajuda a produzir uma implementação mais robusta. Ainda segundo o mesmo autor, identificar e gerenciar aspectos iniciais pode:

- aumentar a consistência dos requisitos com o projeto de arquitetura e destes com a implementação;
- fornecer uma análise lógica e rastreabilidade para requisitos através das atividades do ciclo de vida;
- ajudar a garantir que interesses transversais, evidentes em um domínio de problema do sistema ou em um espaço de soluções, sejam capturados como aspectos na implementação.

Diante da importância de considerar aspectos nos estágios iniciais há alguns trabalhos apresentados na área. Diferentes abordagens para modelar aspectos nos níveis de requisitos [Rashid et al. 2006b], [Kienzle et al. 2009] e de arquitetura [Woodside et al. 2009], [Ali et al. 2010b], [Pinto et al. 2012] foram propostas nos últimos anos.

Para [Sampaio et al. 2007] a AORE visa trazer os benefícios alcançados pela separação de interesses para a Engenharia de Requisitos. Segundo os autores, ferramentas de suporte são fundamentais para reduzir a sobrecarga das tarefas realizadas na AORE. Desse modo, os autores descrevem como uma abordagem baseada em ferramenta, denominada EA-Miner, oferece suporte automatizado para mineração de vários tipos de interesses a partir de uma variedade de documentos de requisitos no estágio inicial, e como esses conceitos podem ser estruturados em um modelo específico de requisitos orientado a aspectos. A ferramenta é baseada em técnicas de processamento de linguagem natural e apenas realiza a identificação dos interesses transversais estruturando-os na linguagem XML. Essa identificação é realizada somente nos requisitos sem considerar outra atividade de desenvolvimento.

Em [Chitchyan et al. 2007b] é apresentada uma linguagem de descrição de requisitos que, segundo os autores, enriquece as especificações de requisitos existentes em linguagem natural com informações semânticas. Especificações de composição são escritas baseadas nessa semântica ao invés da sintaxe de requisitos, fornecendo melhores meios de expressar a intencionalidade da composição e facilitando o raciocínio baseado em semântica sobre a influência de aspectos e *trade-offs* (conflito de escolha).

De acordo com [Kienzle et al. 2009], as abordagens existentes de modelagem orientada a aspectos tem fundamentalmente focado na separação e composição de modelos usando a mesma notação de modelagem. Assim, é apresentada em [Kienzle et al. 2009] uma abordagem de modelagem orientada a aspectos, RAM (*Reusable Aspect Models*), que descreve a modelagem em múltiplas visões. Segundo os autores, RAM permite definir modelos independentes de aspectos reusáveis utilizando três notações de modelagem. As notações são diagramas de Classe da UML para modelagem da estrutura e diagramas de Sequência e de Estado da UML para modelagem do comportamento. Com esses diagramas obtém-se a arquitetura do software modelada, mas não há nenhum tratamento para os aspectos no nível de requisitos.

Distribuição e mobilidade são interesses transversais cuja implementação em aspectos aumenta a reusabilidade e diminui custos de manutenção [Lopes 1997]. Desse modo, [Ali et al. 2010b] apresentam uma abordagem denominada Ambient-PRISMA para modelagem e desenvolvimento de aplicações móveis e distribuídas. O trabalho apresenta um metamodelo que introduz ambientes para projetar modelos arquiteturais de software orientados a aspectos para sistemas móveis. Neste trabalho é apresentado um *middleware*, denominado Ambient-PRISMANET, que mapeia o metamodelo para a tecnologia .NET e suporta ambientes de execução distribuída necessários para executar aplicações móveis. Ambient-PRISMA estende a abordagem PRISMA apresentada em [Perez et al. 2005] e [Perez et al. 2006]. A abordagem PRISMA inclui um metamodelo, uma Linguagem de Descrição de Arquitetura Orientada a Aspectos e uma ferramenta para desenvolvimento de sistemas de software a partir de suas arquiteturas de software orientada a aspectos. Além disso, PRISMA integra o AOSD e o Desenvolvimento de Software Baseado em Componentes para a definição de arquiteturas de software. Em Ambient-PRISMA não há representação dos requisitos do software a ser modelado. A representação é feita somente arquiteturalmente e posteriormente no código fonte.

No trabalho [Navarro et al. 2007] é proposta uma metodologia denominada ATRIUM (*Architecture generaTed from RequIrements applying a Unified Methodology*) que realiza a rastreabilidade entre requisitos e arquitetura de software. Os modelos de arquitetura são especificados com PRISMA. ATRIUM suporta um processo iterativo e incremental envolvendo um conjunto de passos que guiam um conjunto informal de requisitos para um modelo PRISMA instanciado.

No trabalho [Fuentes et al. 2008] é apresentada uma arquitetura orientada a aspectos, denominada CAM (modelos de aspectos e componentes), que usa o Desenvolvimento Dirigido por Modelos. O objetivo é fornecer apoio para definir e especificar arquiteturas orientadas a aspectos utilizando arquiteturas não orientadas a aspectos como fonte. A abordagem de Desenvolvimento Dirigido por Modelos é utilizada para transformar um modelo de arquitetura baseada em componentes em um modelo de arquitetura orientada a aspectos. De acordo com os autores não é necessário que o arquiteto aprenda uma nova linguagem de arquitetura orientada a aspectos, pois foi automatizada a etapa de transformar modelos UML marcados com aspectos para uma especificação DAOP-ADL (*Dynamic Aspect-Oriented Platform - Architecture Description Language*). Assim, o arquiteto de software somente trabalha com a UML 2.0. Os autores afirmam que a rastreabilidade de aspectos a partir dos requisitos pode ser explorada, porém essa possibilidade não é tratada no trabalho.

Em [Nakagama et al. 2011] uma arquitetura de referência para Ambientes de Engenharia de Software (AES) denominada RefASSET (*Reference Architecture for Software Engineering Tools*) é apresentada, envolvendo conhecimento e experiência do desenvolvimento de AES bem como aspectos no nível arquitetural. Nesse trabalho é apresentada

uma especialização de RefASSET para o domínio de teste de software. Os autores concluem que aspectos inseridos em arquiteturas de referência de AES podem contribuir para o desenvolvimento de AES de fácil evolução e ferramentas relacionadas. Há uma diversidade de AES e ferramentas disponíveis em variadas plataformas, e arquiteturas de suporte de diferentes atividades, processos, métodos e técnicas. A abordagem do trabalho não cobre todos os casos, mas é uma direção, com o objetivo de estabelecer uma arquitetura de referência para o domínio de engenharia de software.

Uma linguagem de descrição de arquitetura orientada a aspectos (LDA-OA) baseada em XML é apresentada em [Pinto et al. 2011]. De acordo com os autores, a LDA-OA oferece um conjunto de ferramentas que auxilia o desenvolvedor a especificar uma arquitetura orientada a aspectos impondo o (re)uso de componentes e conectores armazenados em um repositório. A LDA-OA também conecta a arquitetura de software com a fase de projeto e implementação desde que o conjunto de ferramentas da LDA-OA inclua transformações dirigidas a modelos para gerar automaticamente ou um projeto detalhado ou um código orientado a aspectos. Não é citado no trabalho uma relação com a fase de requisitos.

Em [Navasa et al. 2009] uma linguagem de descrição de arquitetura - AspectLEDA - é formalmente descrita com o objetivo de manipular os conceitos de orientação a aspectos na fase de arquitetura de software. Os autores também apresentam um modelo para a inserção de aspectos no nível arquitetural. Esse modelo é representado em duas camadas sendo a primeira camada o sistema base e a segunda camada o nível de aspectos. A primeira etapa do modelo é construir o sistema base, em que se consideram as especificações de requisitos e de projeto em alto nível de abstração sem ainda considerar os aspectos. As especificações são feitas por meio de diagramas da UML, particularmente por diagramas de Casos de Uso e diagramas de Sequência. Após a definição do sistema base, aspectos podem ser adicionados para obter o sistema estendido. Como consequência, quando o diagrama de Casos de Uso é estendido o diagrama de Sequência correspondente também é modificado. Assim que os aspectos são adicionados, a próxima etapa do método é descrever o sistema estendido em termos da linguagem de descrição de arquitetura orientada a aspectos, o que é feito por meio da linguagem AspectLEDA.

Em [Costa-Soria et al. 2009] é apresentada uma abordagem para dinamicamente reconfigurar arquiteturas de software tendo os benefícios de técnicas orientadas a aspectos. As técnicas orientadas a aspectos são usadas para evitar o entrelaçamento entre as descrições de reconfiguração e funcionalidades do sistema. A abordagem apresentada é uma alternativa independente de plataforma enquanto aumenta o reuso e a redução do esforço de manutenção.

Apesar da existência de diferentes abordagens para modelar aspectos nos níveis de requisitos e de arquitetura, ainda há muito o que fazer nessa área. Os modelos para representação de aspectos nos estágios iniciais do ciclo de desenvolvimento de software

apresentados na literatura abordam diferentes domínios de aplicação assim como diferentes linguagens de modelagem. Nos estágios iniciais de desenvolvimento seria muito útil definir algum tipo de mapeamento entre modelos de requisitos orientados a aspectos e arquitetura orientada a aspectos [Sánchez et al. 2010]. Em muitos trabalhos isso não é considerado e o arquiteto de software não tem nenhuma ajuda para identificar interesses transversais no nível arquitetural a partir das especificações de requisitos [Pinto et al. 2011]. Contudo, ainda há ausência de técnicas e ferramentas para lidar com essa questão, principalmente no estabelecimento de uma relação da Engenharia de Requisitos com a Arquitetura de Software, o que pode dificultar a representação dos aspectos iniciais e consequentemente a identificação dos aspectos nas outras fases do desenvolvimento de software. Além disso, poucas aplicações práticas foram propostas.

Desse modo, ainda são relevantes pesquisas na área de aspectos iniciais que apresentem modelos para a representação de aspectos no nível de requisitos e de arquitetura, bem como o estabelecimento da relação dessas atividades de desenvolvimento de software.

1.4 Metodologia

A maioria das pesquisas começa com algum tipo de revisão da literatura. No entanto, a menos que uma revisão ampla da literatura seja feita, ela é de pouco valor científico. Uma revisão sistemática na literatura é um meio de avaliar e interpretar todas pesquisas relevantes disponíveis para uma questão de pesquisa particular, área de tópico ou fenômeno de interesse, sintetizando o trabalho existente [Kitchenham e Charters 2007]. Por exemplo, revisões sistemáticas devem ser realizadas de acordo com uma estratégia de busca pré-definida. A estratégia de busca deve permitir a integridade da pesquisa a ser avaliada. Assim, para entender os benefícios de introduzir aspectos no desenvolvimento de software foi realizada uma revisão sistemática da literatura existente em relação aos modelos propostos para representar aspectos no nível arquitetural. A revisão sistemática é apresentada no Capítulo 3.

Foi também realizado um estudo aprofundado da SysML, uma vez que o modelo obtido é baseado nessa linguagem, principalmente em relação aos diagramas de Requisitos, de Blocos e de Casos de Uso da linguagem.

Um estudo de caso foi utilizado como instrumento da pesquisa. Estudo de caso é uma pesquisa empírica que investiga em profundidade um fenômeno contemporâneo dentro de seu contexto da vida real [Yin 2009]. Segundo [Runeson e Host 2009], estudo de caso é uma metodologia de pesquisa adequada para muitos tipos de pesquisas em engenharia de software. Estudos de caso, em sua verdadeira essência, exploram e investigam fenômenos da vida real através da análise contextual detalhada de um número limitado de eventos ou condições e suas relações.

Conforme as técnicas de validação em Engenharia de Software apresentadas por [Shaw

2002], este trabalho foi validado por meio de *experience* (experiência) em que as validações são baseadas no uso real do resultado. A experiência foi realizada aplicando os modelos desenvolvidos no estudo de caso do sistema Health Watcher. O documento original dos requisitos do sistema Health Watcher foi utilizado como entrada para o modelo de requisitos. Em seguida, a partir das especificações dos requisitos a arquitetura foi proposta.

Este trabalho também foi validado por meio de *evaluation* (avaliação) em que a validação é realizada utilizando um conjunto de critérios. Os critérios selecionados para avaliar o modelo de requisitos foram: processo para identificar requisitos transversais, identificação de requisitos transversais de origem funcional e não funcional, composição de aspectos com requisitos, identificação de conflitos, resolução de conflitos, modelagem gráfica, diagrama de requisitos específico, relacionamento entre requisitos, relação com a UML, extensibilidade do modelo, relacionamento com a arquitetura, rastreabilidade com o projeto, e suporte a automatização. Os critérios selecionados para avaliar o modelo de arquitetura foram: múltiplas visões, quais visões, rastreabilidade com aspectos no nível de requisitos, abordagem simétrica ou assimétrica, suporte a automatização, representação gráfica, composição de aspectos com arquitetura, linguagem de modelagem.

A avaliação foi realizada comparando os modelos apresentados neste trabalho com outros cinco trabalhos de Engenharia de Requisitos Orientada a Aspectos e outros cinco trabalhos de Arquitetura de Software Orientada a Aspectos, utilizando os critérios citados. A seleção destes trabalhos foi a partir da revisão bibliográfica (para requisitos) e da revisão sistemática (para arquitetura), sendo selecionados os cinco trabalhos mais citados de acordo com o número de citações recuperados a partir da ferramenta de busca Google.

1.5 Visão Geral da Dissertação

Este trabalho propõe modelos para representar aspectos no nível de requisitos e no nível arquitetural utilizando a linguagem de modelagem SysML para representar os modelos propostos, contribuindo assim com a pesquisa e a prática em Engenharia de Software. O trabalho está organizado em sete capítulos incluindo este de introdução.

No Capítulo 2 é realizada uma breve revisão de conceitos, onde são apresentados conceitos de Engenharia de Software, Arquitetura de Software, Desenvolvimento de Software Orientado a Aspectos e sobre a linguagem de modelagem SysML.

No Capítulo 3 é descrita uma revisão sistemática em relação a Arquitetura de Software Orientada a Aspectos. É apresentado todo o processo para a realização da revisão assim como a análise dos resultados encontrados.

No Capítulo 4, primeiramente é realizada uma visão geral sobre Engenharia de Requisitos Orientada a Aspectos. Os principais conceitos sobre o diagrama de Requisitos da SysML também são apresentados. O objetivo principal do capítulo é propor um modelo para representar aspectos no nível de requisitos. Neste capítulo também são realizadas

as extensões da linguagem de modelagem SysML para representar aspectos no nível de requisitos.

No Capítulo 5 é apresentada uma visão geral sobre Arquitetura de Software Orientada a Aspectos, diagrama de Blocos e Casos de Uso da linguagem de modelagem SysML. Neste capítulo é proposta uma Arquitetura de Software Orientada a Aspectos com Múltiplas Visões e também as extensões da SysML para representar a arquitetura.

No Capítulo 6 são aplicados os modelos propostos em um estudo de caso. Este capítulo tem como objetivo detalhar a aplicação do modelo. Também é apresentada nesse capítulo uma visão geral das modificações realizadas na ferramenta ArgoUML para representar os modelos de requisitos e a arquitetura de software com aspectos.

Finalmente, no Capítulo 7, são apresentadas as conclusões referentes ao trabalho, algumas contribuições e também possíveis trabalhos futuros que podem ser realizados para dar continuidade a esta pesquisa.

Capítulo 2

Referencial Teórico

Neste capítulo serão abordados conceitos importantes para o melhor entendimento das contribuições propostas nos próximos capítulos. Na Seção 2.1 é apresentada uma visão geral sobre os conceitos de desenvolvimento orientado a aspectos. Na Seção 2.2 é realizada uma introdução sobre a Engenharia de Requisitos. Na Seção 2.3 é realizada uma introdução sobre Arquitetura de Software. Na Seção 2.4 é apresentada a a linguagem de modelagem SysML com suas principais características.

2.1 Desenvolvimento Orientado a Aspectos

O conceito de aspectos foi apresentado por Gregor Kiczales e sua equipe da *Xerox Palo Alto Research Center* no trabalho "*Aspect-Oriented Programming*" [Kiczales et al. 1997]. Os autores encontraram problemas de programação que nas técnicas de orientação a objetos ou procedimental forçam a implementação de decisões de projeto serem espalhadas por todo o código, resultando em código desordenado que é excessivamente difícil de desenvolver e manter. Assim, a motivação para a abordagem de Programação Orientada a Aspectos é derivada dos problemas causados por espalhamento (*scattering*) e entrelaçamento (*tangling*) de código [Kiczales et al. 1997]. A implementação de um interesse é espalhada se o seu código está disperso por vários módulos. O interesse afeta a implementação de múltiplos módulos. A implementação de um interesse é entrelaçada se o seu código é misturado com o código que implementa outros interesses. O módulo no qual ocorre o entrelaçamento não é coeso.

O Desenvolvimento de Software Orientado a Aspectos (AOSD) foi originado a partir da Programação Orientada a Aspectos. O AOSD é uma tecnologia de desenvolvimento de software ajuda a alcançar melhor separação de interesses fornecendo mecanismos para localizar interesses transversais em artefatos de software através do processo de desenvolvimento de software [Shaker e Peters 2005]. Na separação de interesses cada módulo realiza somente uma tarefa, o que resulta em artefatos mais coesos. O termo interesses transversais (*crosscutting concerns*) refere-se a fatores de qualidade ou funcionalidades

de software que não podem ser efetivamente modularizadas usando técnicas de desenvolvimento de software existentes como a de orientação a objetos, pois seguem diferentes regras de decomposição. As técnicas de AOSD fornecem um meio sistemático para identificação, modularização, representação e composição de interesses transversais [Rashid et al. 2003]. O AOSD permite que múltiplos interesses sejam expressados separadamente e automaticamente unificados em sistemas de trabalho [Filman et al. 2004]. Interesses transversais que são encapsulados em módulos separados são conhecidos como aspectos, de modo que a localização desses interesses pode ser realizada [Rashid et al. 2003].

Alguns conceitos importantes da Programação Orientada a Aspectos mas que também são utilizados no AOSD são:

- *joinpoint* - os *joinpoints* indicam a localização no programa principal onde o aspecto será inserido [Kiczales et al. 1997];
- *pointcut* - os *pointcuts* são expressões que se referem a um conjunto de *joinpoints* [Pinto et al. 2009a];
- *advice* - os *advices* são o comportamento adicional (ou ações) que serão inseridos no programa principal [Shaker e Peters 2005];
- tipo de *advice* - os *advices* possuem tipos que especificam como o *advice* será operado. Os tipos mais citados são *before*, em que o *advice* é executado antes do *joinpoint*, *after*, em que o *advice* é executado depois do *joinpoint*, e *around*, que executa "em volta" do *joinpoint* (esse tipo de *advice* serve para substituir a execução do *joinpoint* pela execução do *advice*, ou executar parte do *advice* antes do *joinpoint* e outra parte depois) [Bennouar et al. 2010];
- *weaver* - a inserção do *advice* é alcançada por meio de um mecanismo especial chamado *weaver*. O mecanismo *weaver* é responsável por compor o programa principal com os aspectos [Kiczales et al. 1997].

2.2 Importância da Engenharia de Requisitos

A Engenharia de Requisitos é considerada uma parte fundamental do ciclo de vida do software [Juristo et al. 2002], [Ian e Stevens 2002], [Sommerville 2010] uma vez que requisitos mal especificados podem ser um fator crítico no fracasso de um projeto. Para [Vliet 2008], a fase de Engenharia de Requisitos é o primeiro grande passo para a solução de um problema. Durante esta fase, as exigências do usuário com respeito a um sistema futuro são cuidadosamente identificadas e documentadas.

Para [Zave 1997], a Engenharia de Requisitos é uma parte da Engenharia de Software preocupada com os objetivos do mundo real para funções e restrições dos sistemas de software. Está preocupada também com a relação destes fatores para especificações precisas do comportamento do software, e para a sua evolução ao longo do tempo.

Para [Nuseibeh e Easterbrook 2000], a Engenharia de Requisitos é o processo de descobrir qual o propósito do software, pela identificação dos *stakeholders* e suas necessidades, e documentar isso de uma forma que seja passível de análise, comunicação e subsequente implementação. A definição tradicional de *stakeholder* é qualquer grupo ou indivíduo que pode afetar ou ser afetado pela realização dos objetivos de uma organização [Freeman 2010]. Durante a Engenharia de Requisitos, diferentes tipos de *stakeholders* podem ser a fonte de diferentes tipos de requisitos. Os *stakeholders* estabelecem as funções, as restrições e os objetivos do sistema.

Segundo [Vliet 2008], o aspecto mais difícil e desafiador da Engenharia de Requisitos é obter uma descrição completa do problema a ser resolvido. Assim, geralmente a Engenharia de Requisitos é dividida em quatro atividades [Sommerville 2005]: elicitação de requisitos, especificação de requisitos, validação e verificação de requisitos, e negociação de requisitos.

Na negociação de requisitos pode ser utilizado um meio de priorização de requisitos conhecido como MoSCoW [Vliet 2008]. Nessa técnica os requisitos são diferenciados em quatro tipos:

- *must* (deve) - esses são requisitos no topo da prioridade, eles são obrigatórios para que o sistema seja aceito pelo cliente.
- *should* (deveria)- esses requisitos não são estritamente obrigatórios, mas são altamente desejáveis;
- *could* (poderia)- se o tempo permitir esses requisitos serão realizados, na prática geralmente não são;
- *won't* (não irá)- esses requisitos não serão realizados na presente versão, eles serão considerados em alguma próxima versão do sistema.

Outra questão que pode ocorrer são conflitos de escolhas entre *stakeholders*. Assim, é necessário realizar negociação entre os *stakeholders*.

Requisitos podem ser classificados em várias maneiras. Uma classificação apresentada em [Sommerville 2010] é em relação ao nível de detalhes. Nessa classificação são utilizados os termos: requisitos do usuário, que são declarações em alto nível sobre as funções que o sistema deve fornecer e as restrições sob as quais deve operar, geralmente estas declarações são em linguagem natural; requisitos do sistema, que são derivados a partir dos requisitos de usuário mas com uma descrição detalhada das funções e as restrições de sistema e; especificação de projeto de software, que é uma descrição abstrata do projeto de software, sendo este uma base para o projeto e a implementação mais detalhados.

No final do processo o resultado da Engenharia de Requisitos é documentado na especificação de requisitos que serve como um ponto de início para o desenvolvimento da arquitetura de software. A especificação de requisitos reflete a compreensão mútua do

problema a ser resolvido entre o analista e o cliente. A Engenharia de Requisitos e o projeto da arquitetura não podem ser estritamente separados. Geralmente um projeto preliminar é feito depois que um conjunto inicial de requisitos forem determinados. A especificação de requisitos pode ser mudada e refinada. Lidar com mudanças nos requisitos é considerado um problema de Engenharia de Software [Berry 2004]. Assim, conhecer o que construir, o que inclui elicitação de requisitos e técnicas de especificação, é a fase mais difícil no projeto de software [Brooks 1987].

2.3 Importância da Arquitetura de Software

Projetar uma Arquitetura de software é uma atividade que ajuda os desenvolvedores não somente a controlar a complexidade de um software mas também a definir sua estrutura com o objetivo de obter fácil manutenção e evolução [Shaw e Garlan 1996], [Navasa et al. 2009].

Para o sucesso de um projeto de software é importante ter uma arquitetura bem projetada. A arquitetura permite capturar decisões iniciais de projeto o que levará em softwares mais confiáveis [Bass et al. 1998], [Bosch 2004].

A arquitetura de software descreve sua estrutura em termos de componentes (unidades computacionais), conectores (unidades de coordenação) e suas configurações arquiteturais (conexão de componentes e conectores) [Medvidovic e Taylor 2000]. Essa estrutura colabora com a compreensão do software mesmo antes de sua implementação.

Uma arquitetura de software serve como um meio de comunicação entre os *stakeholders*. Todos os *stakeholders* tem interesses que podem ser diferentes. Por exemplo, usuários finais estão interessados em ver como o sistema fornecerá as funcionalidades solicitadas e os desenvolvedores de software estão interessados em saber onde e como implementar as funcionalidades. Por isso é que se tem utilizado bastante a prática de visões arquiteturais.

O termo visão significa a representação de um sistema completo a partir da perspectiva de um conjunto relacionado de interesses [Hilliard 2000].

O padrão IEEE 1471 recomenda uma prática para descrição de arquitetura em múltiplas visões para *software-intensive systems* [Hilliard 2000]. Segundo este modelo, uma descrição de arquitetura é organizada em uma ou mais visões (*view*) arquiteturais. Na Figura 2.1 é mostrado o modelo conceitual para a descrição da arquitetura do padrão IEEE 1471.

Há vários modelos de arquitetura em múltiplas visões como: as visões da OMT (*Object Modeling Technique*) [Rumbaugh et al. 1991], as visões do RM-ODP (*Reference Model of Open Distributed Processing*) [Farooquia et al. 1995], as visões de Booch [Booch 1994], as visões de Leist e Zellner [Leist e Zellner 2006] e o modelo em múltiplas visões "4+1" [Kruchten 1995].

O modelo "4+1" é considerado uns dos principais modelos para representação de ar-

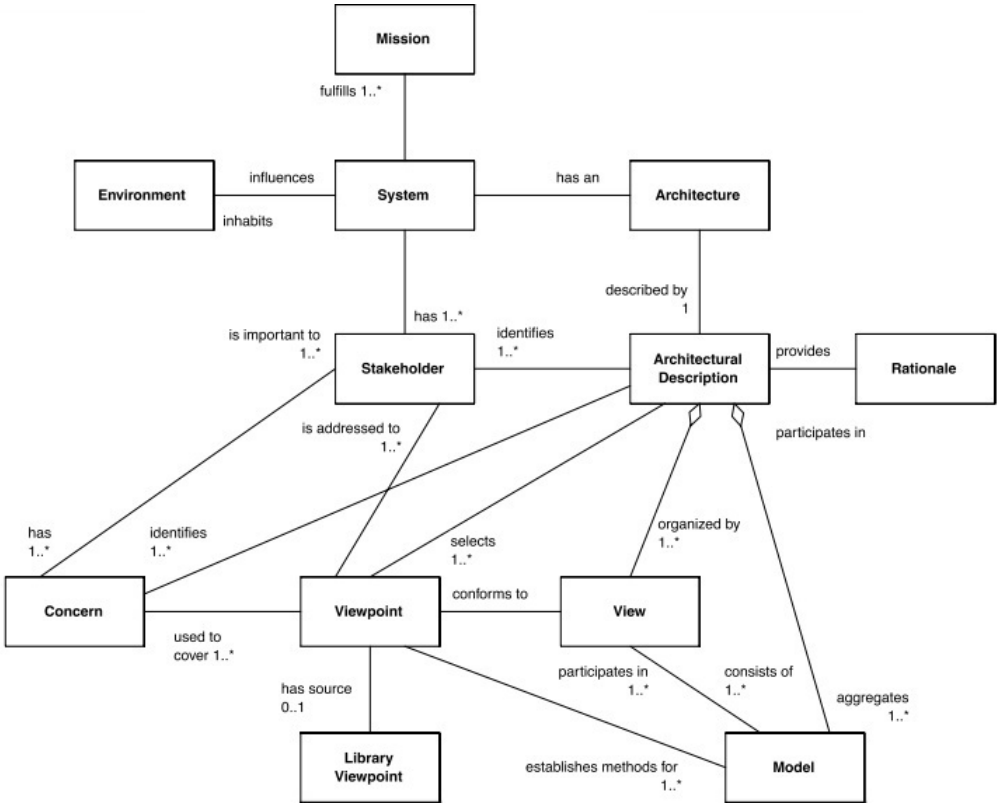

Figura 2.1: Modelo conceitual para descrição de arquitetura do padrão IEEE 1471 [Hilliard 2000].

quitetura [Garlan et al. 2010]. Como pode ser observado na Figura 2.2, as visões consideradas são a lógica (*logical*), de processo (*process*), a física (*physical*), de desenvolvimento (*development*) e cenários (*scenarios*). A visão lógica suporta principalmente os requisitos funcionais, ou seja, o que o sistema deve fornecer em termos de serviços para seus usuários. A visão de processo foca no comportamento do sistema. Um processo é um agrupamento de tarefas que formam uma unidade executável. A visão física descreve o mapeamento do software em hardware e reflete seus aspectos distribuídos. A visão de desenvolvimento descreve a organização estática do software em seu ambiente de desenvolvimento. Nessa visão o sistema é descrito do ponto de vista do programador. Os cenários são instâncias de casos de uso mais gerais. Cenários são, em certo sentido, abstrações dos requisitos mais importantes.

O modelo de visões "4+1" é genérico. Várias notações, ferramentas e métodos de projetos podem ser usados. As várias visões não são totalmente ortogonais ou independentes. Elementos de uma visão são conectados com elementos em outras visões, seguindo certas regras de projeto e heurísticas.

Para descrever uma arquitetura de software podem ser utilizadas linguagens de modelagem de software e também linguagens de descrição de arquitetura chamadas ADLs (*Architecture Description Languages*). As ADLs são linguagens formais que suportam o desenvolvimento baseado em arquitetura [Medvidovic e Taylor 2000]. As linguagens de

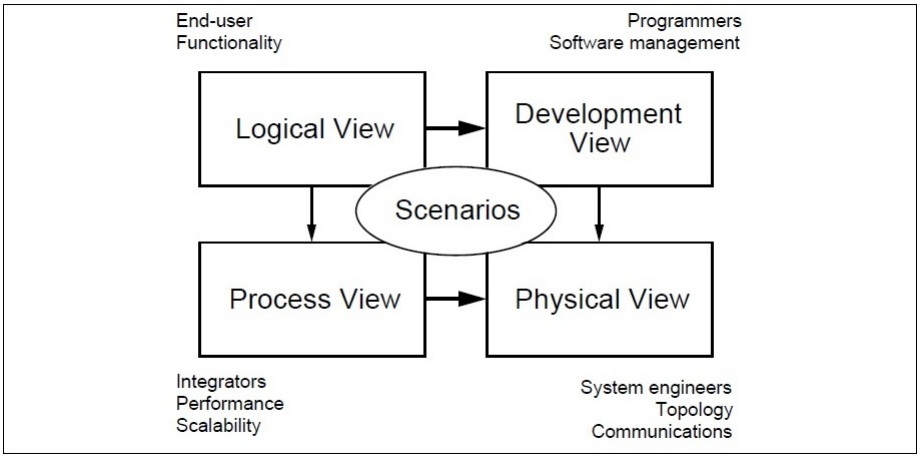

Figura 2.2: O modelo de visões "4+1" [Kruchten 1995].

modelagem de software estão mais preocupadas com o comportamento como um todo e não das partes, enquanto que as ADLs concentram-se na representação dos componentes, mas na prática muitas linguagens de modelagem permitem a representação de componentes e podem representar arquiteturas razoavelmente bem [Clements 1996].

Segundo [Medvidovic e Taylor 2000], várias ADLs tem sido propostas para modelagem de arquitetura, tanto para um domínio particular quanto para propósito geral. Ainda segundo estes autores, uma ADL deve explicitamente modelar componentes, conectores e suas configurações arquiteturais. Além disso, deve fornecer ferramentas de suporte para a evolução e o desenvolvimento baseado em arquitetura.

2.4 SysML

A SysML é uma linguagem de modelagem de propósito geral especificada pelo *Object Management Group* (OMG). Ela suporta a especificação, análise, projeto, verificação e validação de uma variedade de sistemas complexos [OMG-SysML 2010]. Esses sistemas podem incluir hardware, software, informação, processos e pessoal.

A linguagem SysML é derivada a partir da linguagem UML. Isso facilita a comunicação entre equipes heterogêneas que trabalham juntas para desenvolver um sistema intensivo de software [Soares 2010]. A SysML reusa muitos dos principais diagramas da UML. Em alguns casos os diagramas são estritamente reusados e em outros casos eles são modificados como pode ser observado na Figura 2.3. A SysML não reusa diagramas da UML relativos a software, como Diagramas de Componentes e de Desenvolvimento. O Diagrama de Blocos é utilizado em substituição a esses diagramas, uma vez que é um diagrama geral que pode representar elementos em vários níveis de abstração.

O diagrama de Requisitos é um novo tipo de diagrama da SysML. O diagrama de Requisitos fornece um construtor de modelagem de requisitos baseados em texto e o relacionamento entre requisitos e outros elementos do modelo que os satisfazem ou que os

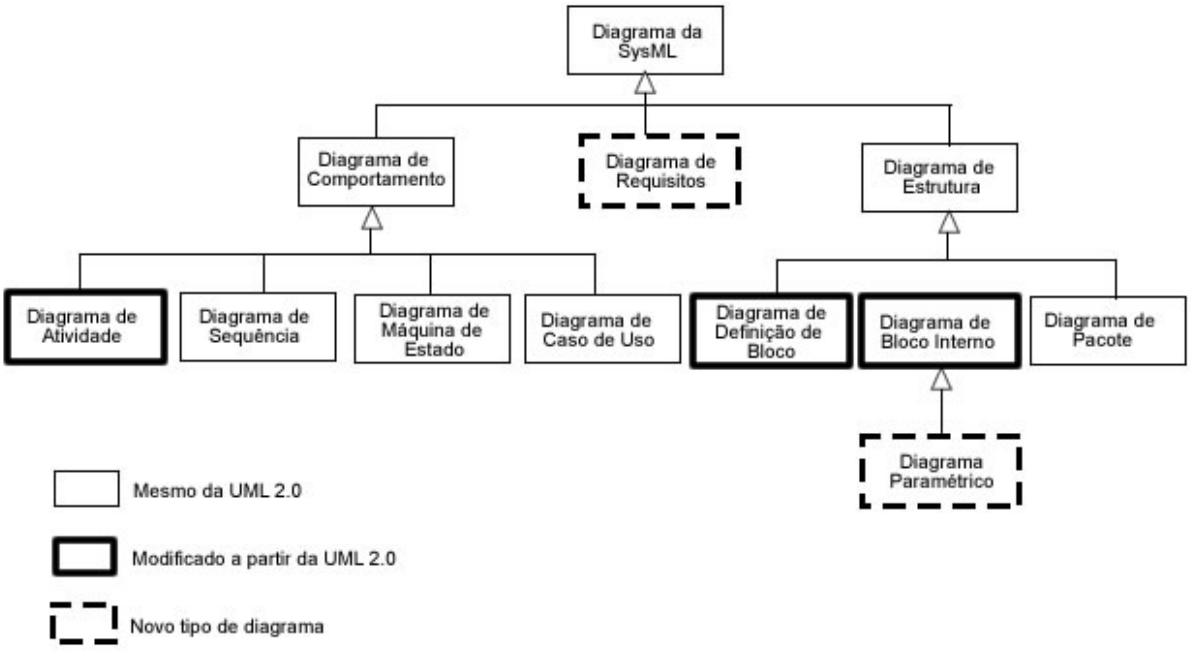

Figura 2.3: Diagramas da SysML [OMG-SysML 2010].

verificam. Outro novo tipo de diagrama da SysML é o diagrama Paramétrico que descreve as restrições entre as propriedades associadas com os blocos. Esse diagrama é usado para integrar modelos de comportamento e estrutura com modelos de análise de engenharia, tais como desempenho e confiabilidade.

Capítulo 3

Revisão Sistemática da Literatura

Este capítulo foi baseado no artigo *"A Systematic Review on Aspects in Software Architecture Design"* apresentado na *XXXI International Conference of the Chilean Computer Science Society*. O objetivo do capítulo é apresentar uma revisão sistemática em relação à Arquitetura de Software Orientada a Aspectos. Na Seção 3.1 é apresentado de modo geral o motivo de se realizar a revisão sistemática. Na Seção 3.2 é apresentado o método de pesquisa utilizado. Na Seção 3.3 são mostrados os critérios utilizados na avaliação dos artigos selecionados na revisão sistemática. Na Seção 3.4 são realizadas as discussões dos resultados encontrados. Na Seção 3.5 são feitas as conclusões da revisão realizada.

3.1 Objetivo de Realizar a Revisão Sistemática

Os reais benefícios de introduzir aspectos em arquitetura de software ainda são debatidos na literatura [Ali et al. 2010a]. Por exemplo, Madeyski e Szala [Madeyski e Szala 2007] chegaram à conclusão que o impacto da programação orientada a aspectos na eficiência e qualidade do projeto de desenvolvimento de software não foi confirmado uma vez que a utilização de programação orientada a aspectos não afeta significativamente métricas de qualidade do projeto de software. Os autores recomendam a realização de mais pesquisas. Entretanto, Lobato et al. [Lobato et al. 2008] apresentaram uma arquitetura de software orientada a aspectos, chamada ArchM, e afirmaram que a arquitetura garantiu melhor modularização, melhor separação de interesses, baixo acoplamento, alta coesão e melhores métricas de complexidade quando comparados com a arquitetura original dos estudos de casos realizados.

O propósito da pesquisa sistemática é avaliar quais são os interesses mais comuns quando é projetada uma arquitetura de software utilizando aspectos e se a introdução de aspectos é útil na manutenção de software. O objetivo de uma revisão sistemática na literatura é sintetizar evidências existentes relativas a uma tecnologia. Quando bem projetada, a revisão sistemática suporta o desenvolvimento de um guia baseado em evidências para profissionais [Kitchenham et al. 2009]. Revisões sistemáticas requerem considera-

velmente mais esforço do que revisões tradicionais. Sua maior vantagem é que fornece informações sobre o efeito de algum fenômeno através de uma grande variedade de definições e métodos empíricos [Kitchenham e Charters 2007]. Ainda segundo [Kitchenham e Charters 2007] algumas das características que diferem uma revisão sistemática de uma revisão convencional são:

- Revisões sistemáticas iniciam por definir um protocolo de revisão que especifica as questões de pesquisa que serão abordadas e os métodos que serão usados para realizar a revisão.
- Revisões sistemáticas são baseadas na estratégia de pesquisa definida que objetiva detectar o máximo de literatura relevante quanto possível.
- Revisões sistemáticas documentam sua estratégia de pesquisa de modo que os leitores possam avaliar o rigor, a completude e a repetibilidade do processo.
- Revisões sistemáticas requerem critérios explícitos de inclusão e exclusão para avaliar cada potencial estudo primário.
- Uma revisão sistemática é um pré-requisito para análises quantitativas da literatura.

3.2 Método da Pesquisa

Com o objetivo de conduzir a revisão sistemática, um protocolo foi definido. O protocolo define as estratégias de pesquisa, incluindo questões de pesquisa, *string* de busca, seleção de veículos de publicação, período de publicação e critérios de inclusão e exclusão dos artigos. As próximas subseções discutem o protocolo em mais detalhes.

3.2.1 Questão de Pesquisa

A questão de pesquisa que motivou esse estudo é:

"Quais benefícios a introdução de aspectos traz no projeto de arquitetura de software, considerando requisitos não-funcionais?"

A resposta para essa questão é crucial para saber se a introdução de aspectos no projeto de arquitetura de software para gerenciar interesses transversais é vantajosa. Com o objetivo de responder a essa questão, a proposta é iniciar a pesquisa por realizar uma revisão sistemática com o propósito de identificar artigos de pesquisa publicados nos quais aspectos foram introduzidos no projeto arquitetural.

3.2.2 Veículos e Período de Publicação

A revisão sistemática foi iniciada com a busca em um número de conferências e revistas científicas de engenharia de software.

O período de publicação foi a partir de janeiro de 2006 até setembro de 2011.
As conferências escolhidas foram:

- ECOOP - *European Conference on Object-Oriented Programming*;
- OOPSLA - *International Conference on Object Oriented Programming*;
- AOSD - *International Conference on Aspect-Oriented Software Development*;
- CSMR - *European Conference on Software Maintenance and Reengineering*;
- ICSE - *International Conference on Software Engineering*;
- ECSA - *European Conference on Software Architecture* e;
- WICSA - *Working IEEE/IFIP Conference on Software Architecture.*

As revistas científicas escolhidas foram:

- SCP - *Science of Computer Programming*;
- JSS - *Journal of Systems and Software*;
- TOSEM - *ACM Transactions on Software Engineering Methodology*;
- TSE IEEE - *IEEE Transactions on Software Engineering* e;
- IST - *Information and Software Technology.*

Estas conferências e revistas científicas foram consideradas porque elas estão entre os melhores veículos para publicar artigos de engenharia de software [Wong et al. 2011].

Artigos publicados em *workshops* realizados em conjunto com estas conferências não foram considerados. Além disso, foram excluídos editoriais, palestras, painéis de discussão e artigos que não exibem uma profunda relação com a arquitetura de software e desenvolvimento de software orientado a aspectos. Na Tabela 3.1 um resumo dos resultados de busca e dos estudos selecionados é apresentado.

3.2.3 *String* de Busca

A escolha das *strings* de busca foi:
(*"aspect oriented"* AND *"software architecture"*).

Dois métodos foram aplicados para a busca: automático e manual. A busca automática refere-se em executar a *string* em uma máquina de busca de fonte de dados eletrônicos. A busca manual refere-se em buscar manualmente pelos artigos.

Os dois métodos foram usados porque a busca automática não estava disponível para as conferências ECOOP (anos de 2006, 2008, 2009 e 2011) e ECSA (anos de 2007 e 2011).

Conferências e Revistas	Artigos recuperados	Seleção inicial	Seleção final	Método de busca	Fonte de dados eletrônica
ECOOP	9	4	0	Automático e Manual	ACM
OOPSLA	21	5	1	Automático	ACM
AOSD	48	8	2	Automático	ACM
CSMR	2	2	1	Automático	IEEEXplore
ICSE	129	4	1	Automático	ACM
ECSA	64	17	0	Automático e Manual	ACM
WICSA	19	11	3	Automático	IEEEXplore
SCP	2	2	1	Automático	Science Direct
JSS	6	6	5	Automático	Science Direct
TOSEM	4	3	0	Automático	ACM
TSE IEEE	17	6	0	Automático	IEEEXplore
IST	6	6	5	Automático	Science Direct
Total	327	74	19	—	—

Tabela 3.1: Visão geral dos resultados da busca e seleção de estudos.

3.2.4 Critério para Incluir um Artigo

Para cada artigo, o título, as palavras-chave e o resumo foram lidos com o objetivo de fazer a seleção. Quando em dúvida se o artigo deveria ser adicionado, foram lidos também a introdução e a conclusão.

As palavras-chave consideradas foram relacionadas à arquitetura de software, incluindo, por exemplo, *modularity*, *pattern*, *style*, *performance*, e as bem conhecidas **ilities* [Sommerville 2010]. Atenção especial foi dada a qualidades tais como *understandability* e *modifiability*, que foram avaliadas para cada artigo. O propósito foi avaliar os interesses mais comuns considerados quando uma arquitetura de software usando aspectos é projetada e se a introdução de aspectos é útil na manutenção de software.

O processo de seleção é descrito na Figura 3.1. O número total de artigos recuperados foi 327. Após a aplicação dos critérios, 19 artigos foram totalmente lidos.

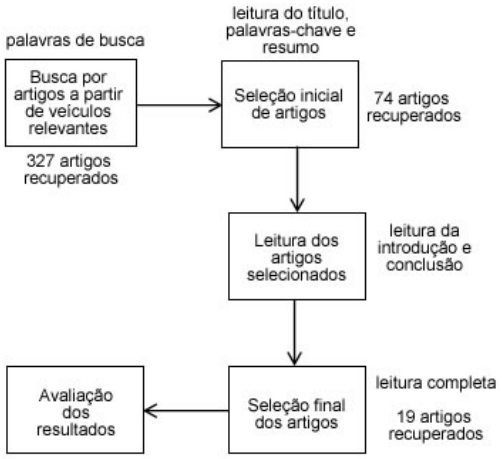

Figura 3.1: Processo de seleção de artigos.

Na Tabela 3.2 é apresentada uma visão geral dos 19 artigos que foram selecionados para leitura completa, o veículo e o ano de publicação.

Veículo/Ano	**Nome do artigo**	**Referência**
JSS 2010	Ambient-PRISMA: Ambients in Mobile Aspect-Oriented Software Architecture	[Ali et al. 2010b]
JSS 2010	A New Approach for Component's Port Modeling in Software Architecture	[Bennouar et al. 2010]
JSS 2010	Enhancing Middleware Support for Architecture-Based Development through Compositional Weaving of Styles	[Malek et al. 2010]
JSS 2011	An Aspect-Oriented Reference Architecture for Software Engineering Environments	[Nakagama et al. 2011]
JSS 2011	Deriving Detailed Design Models from an Aspect-Oriented ADL using MDD	[Pinto et al. 2012]
IST 2008	Generating CAM Aspect-Oriented Architectures using Model-Driven Development	[Fuentes et al. 2008]
IST 2008	Integrating Aspects in Software Architectures: PRISMA Applied to Robotic Tele-Operated Systems	[Perez et al. 2008]
IST 2009	An ADL Dealing with Aspects at Software Architecture Stage	[Navasa et al. 2009]
IST 2009	Malaca: A Component and Aspect-Oriented Agent Architecture	[Amor e Fuentes 2009]
IST 2011	Specifying Aspect-Oriented Architectures in AO-ADL	[Pinto et al. 2011]
JCSP 2006	Modeling and Performance Analysis for Security Aspects	[Dai e Cooper 2006]
AOSD 2009	Composing Architectural Aspects Based on Style Semantics	[Chavez et al. 2009]
AOSD 2009	Domain Driven Discovery of Stable Abstractions for Pointcut Interfaces	[Van Landuyt et al. 2009]
CSMR 2009	Handling the Dynamic Reconfiguration of Software Architectures Using Aspects	[Costa-Soria et al. 2009]
WICSA 2009	Aspect-Oriented Space Containers for Efficient Publish/Subscribe Scenarios in Intelligent Transportation Systems	[Kuhn et al. 2009]
WICSA 2009	On the Need of Architectural Patterns in AOSD for Software Evolution	[Pinto et al. 2009b]
WICSA 2007	Requirements and Scenarios: Running Aspect-Oriented Software Architectures	[Navarro et al. 2007]
OOPSLA 2007	Epi-Aspects: Aspect-Oriented Conscientious Software	[Fleissner e Baniassad 2007]
ICSE 2009	Taming Dynamically Adaptive Systems using models and aspects	[Morin et al. 2009]

Tabela 3.2: Artigos selecionados.

3.3 Critérios Usados na Avaliação

A avaliação dos artigos foi baseada em critérios como: tipo de validação, requisitos não-funcionais, visões arquiteturais e domínio de aplicação. Em seguida cada critério é descrito:

- Tipo de Validação - Há uma variedade de métodos e abordagens para validar o trabalho de pesquisa, incluindo a aplicação da pesquisa para o ambiente industrial, experimento controlado e aplicação da teoria em estudos de casos. Nessa revisão esses tipos de validação foram considerados. Foi considerado que ambiente industrial pode trazer resultados mais fortes, seguido por experimento controlado e estudos de caso [Shaw 2002].
- Requisitos não-funcionais - Na literatura, requisitos não-funcionais são frequentemente referenciados como "ilities" (por exemplo *usability*) ou "ities" (por exemplo *integrity*) [Chung e Prado Leite 2009]. Com o objetivo de avaliar as qualidades de cada abordagem, requisitos não-funcionais tais como *security*, *reusability*, *maintainability*, *scalability* e outros foram considerados. Estes estão entre os mais importantes requisitos não-funcionais para manutenção e reengenharia de software.
- Visões Arquiteturais - É consenso que uma descrição arquitetural consiste de múltiplas visões [Boucke et al. 2008]. O termo visão é usado para se referir a expressão de uma arquitetura do sistema com respeito a um ponto de vista particular usado para alcançar a separação de interesses [Hilliard 2000].
- Domínio de Aplicação - O domínio de aplicação é importante para saber qual o tipo específico de arquitetura de sistema foi desenvolvido. Embora o sistema de software possua muitos elementos comuns, o domínio específico deve ser levado em consideração, já que possui alta influência na arquitetura. Certos tipos de domínios requerem visões específicas. Por exemplo, é esperado que a visão de processo deva ser incluída na arquitetura de um sistema de tempo real distribuído.

3.4 Discussão

Na Tabela 3.3 as características identificadas para cada artigo são descritas, não considerando os requisitos não-funcionais que serão apresentados na Tabela 3.4.

A partir dos estudos analisados, houve apenas três aplicações industriais, o que demonstra que, apesar do aumento das pesquisas e interesse na área, a arquitetura de software orientada a aspectos não foi amplamente utilizada em ambiente industrial. No entanto, esse resultado deve ser considerado com cuidado uma vez que a indústria frequentemente não publica seus resultados.

Referência	**Tipo de avaliação**	**Visões Arquiteturais**	**Domínio de Aplicação**	**Linguagem de Modelagem**
[Ali et al. 2010b]	Estudo de Caso	Não definido	Sistemas Móveis e Distribuídos	AO-ADL estendida de PRISMA
[Bennouar et al. 2010]	Estudo de Caso	Comportamental e Estrutural	Middleware	ADL chamada SEAL
[Malek et al. 2010]	Estudo de Caso	Estrutural	Middleware	UML
[Pinto et al. 2012]	Estudo de Caso	Estrutural	E-commerce	XML, ADL não identificada, UML 2.0 e Theme/UML
[Nakagama et al. 2011]	Estudo de Caso	Módulo, Tempo de Execução e Desenvolvimento	Ambientes de Engenharia de Software e teste de software	UML 2.0 para representar arquitetura de referência orientada a aspectos
[Pinto et al. 2011]	Industrial	Comportamento Aspectual	Sistemas de Saúde	AO-ADL
[Navasa et al. 2009]	Estudo de Caso	Estrutural e Comportamental	Sistema de Reserva	UML e AspectLEDA
[Fuentes et al. 2008]	Estudo de Caso	Estrutural e Comportamental	Sistemas Distribuídos	UML 2.0, CAM Profile para UML 2.0 e DAOP-ADL
[Perez et al. 2008]	Industrial	Funcional	Sistemas Teleoperadores Robóticos	AO-ADL estendida de PRISMA
[Amor e Fuentes 2009]	Estudo de Caso	Estrutural e Comportamental	Software Baseado em Agente	UML e MaDL baseada em XML
[Dai e Cooper 2006]	Estudo de Caso	Estrutural	Sistemas de Tempo Real	Extensões da UML
[Chavez et al. 2009]	Estudo de Caso	Estrutural e Comportamental	Não definido	AspectualACME AO-ADL e CSP (*Communicating Sequential Processes*)
[Van Landuyt et al. 2009]	Estudo de Caso	Caso de Uso	Sistemas Financeiros	UML e AO-ADL
[Costa-Soria et al. 2009]	Não validado	Não definido	Não definido	PRISMA AO-ADL
[Kuhn et al. 2009]	Industrial	Não definido	Sistemas de Transporte Inteligente	Não definido
[Pinto et al. 2009b]	Estudo de Caso	Não definido	Aplicações para Dispositivos Móveis	AO-ADL
[Navarro et al. 2007]	Estudo de Caso	Comportamental	Dispositivos Eletrônicos	UML e PRISMA
[Fleissner e Baniassad 2007]	Estudo de Caso	Estrutural	Sistemas de Informação	Não definido
[Morin et al. 2009]	Estudo de Caso	Estrutural	Sistemas Adaptativos Dinamicamente	Não definido

Tabela 3.3: Resumo dos artigos escolhidos.

Outro fato interessante sobre o tipo de avaliação é que experimento controlado não foi aplicado em nenhum dos artigos pesquisados.

De acordo com a Tabela 3.3, as visões arquiteturais mais comuns descritas nos artigos

escolhidos são a visão estrutural e a visão comportamental. Além disso, visões importantes, tais como recursos, processos e cenários não foram mencionadas em nenhum dos 19 artigos. Isso é uma surpresa, e também uma questão importante, uma vez que é bem conhecido que arquitetura de software deve ser representada em múltiplas visões, o que é útil para melhorar a separação de interesses. Mesmo em domínios tais como sistemas distribuídos, as visões de processo e de recurso não são mencionadas. Alguns trabalhos não deixam claro quais visões foram adotadas na arquitetura.

A maioria das abordagens usam sua própria linguagem de descrição de arquitetura orientada a aspectos, o que dependendo do nível de dificuldade da linguagem pode inibir seu uso. A UML é uma linguagem utilizada em vários trabalhos, normalmente com estereótipos adicionais para representar aspectos.

A questão de pesquisa que motivou este estudo é sobre os benefícios da introdução de aspectos na arquitetura de software. Ficou claro que o requisito não-funcional mais importante nas arquiteturas apresentadas nos artigos é modularidade (mencionado em 17 artigos dos 19). Isso é esperado, uma vez que aspectos foram propostos com o objetivo de melhorar a modularidade de interesses transversais. Reusabilidade é também frequentemente mencionada (12 dos 19 artigos), o que significa que a introdução de aspectos também ajuda a introduzir melhores formas para reusar software.

De acordo com [Bartsch e Harrison 2008], o fator de manutenção deve ser dividido nos critérios de *understandability* (compreensibilidade) e *modifiability* (modificabilidade). O critério *modifiability* refere-se à facilidade que uma mudança pode ser aplicada em um sistema de software e *understandability* refere-se à facilidade para entender o sistema de software. *Modifiability* e *understandability*, que são propriedades fundamentais para manutenção de software, são mencionados em cerca da metade dos artigos. Assim, pode-se deduzir que a introdução de aspectos no nível arquitetural contribui para as atividades de manutenção de software.

Somente um artigo mencionou escalabilidade e extensibilidade. Essas são propriedades importantes para a evolução do sistema e devem ser analizadas em pesquisas futuras.

3.5 Conclusões da Revisão Realizada

Os reais benefícios da introdução de aspectos em arquitetura de software ainda são debatidos na literatura. Por isso, uma revisão sistemática da literatura em relação à arquitetura de software orientada a aspectos foi conduzida. Para a realização da revisão sistemática um protocolo foi definido, utilizando critérios de avaliação tais como tipos de validação, requisitos não-funcionais, visões arquiteturais e domínio de aplicação.

Os resultados da pesquisa mostram que há benefícios em incluir aspectos no nível de arquitetura em todos os artigos. Sobre os efeitos positivos, modularidade é enfatizada, uma vez que foi citada em quase todos os artigos. Além disso, manutenção é facilitada

Requisitos não-funcionais	**Conferências e Revistas**	**Total**
Modularity	Todos exceto [Pinto et al. 2012], [Malek et al. 2010] and [Morin et al. 2009]	17
Reusability	[Ali et al. 2010b], [Pinto et al. 2012], [Nakagama et al. 2011], [Pinto et al. 2011], [Navasa et al. 2009], [Fuentes et al. 2008], [Perez et al. 2008], [Amor e Fuentes 2009], [Dai e Cooper 2006], [Chavez et al. 2009], [Van Landuyt et al. 2009], [Costa-Soria et al. 2009]	12
Modifiability	[Ali et al. 2010b], [Pinto et al. 2011], [Amor e Fuentes 2009], [Dai e Cooper 2006], [Chavez et al. 2009], [Van Landuyt et al. 2009], [Costa-Soria et al. 2009], [Pinto et al. 2009b], [Fleissner e Baniassad 2007]	9
Understandability	[Nakagama et al. 2011], [Malek et al. 2010], [Pinto et al. 2011], [Navasa et al. 2009], [Fuentes et al. 2008], [Perez et al. 2008], [Amor e Fuentes 2009], [Dai e Cooper 2006]	8
Evolvability	[Nakagama et al. 2011], [Pinto et al. 2011], [Navasa et al. 2009], [Fuentes et al. 2008], [Perez et al. 2008], [Costa-Soria et al. 2009], [Pinto et al. 2009b]	7
Traceability	[Pinto et al. 2011], [Navasa et al. 2009], [Fuentes et al. 2008], [Amor e Fuentes 2009], [Van Landuyt et al. 2009], [Navarro et al. 2007]	6
Flexibility	[Perez et al. 2008], [Amor e Fuentes 2009], [Costa-Soria et al. 2009], [Malek et al. 2010]	4
Composability	[Pinto et al. 2011], [Fuentes et al. 2008], [Perez et al. 2008]	3
Interoperability	[Nakagama et al. 2011], [Amor e Fuentes 2009]	2
Adaptability	[Ali et al. 2010b], [Navasa et al. 2009]	2
Stability	[Chavez et al. 2009], [Van Landuyt et al. 2009]	2
Scalability	[Pinto et al. 2011], [Pinto et al. 2012]	2
Extensibility	[Amor e Fuentes 2009]	1
Reliability	[Fleissner e Baniassad 2007]	1

Tabela 3.4: Requisitos não-funcionais encontrados.

quando aspectos são considerados na arquitetura. Isso pode ser concluído com os requisitos não-funcionais *modifiability* e *understandability*, os quais são essenciais para atividades de manutenção, uma vez que foram considerados na maioria dos artigos.

Uma ameaça à validade é que alguns artigos não mostram detalhes suficientes na avaliação e nas visões arquiteturais. Assim, fica difícil verificar quais tipos de avaliações e visões arquiteturais são trabalhadas nos artigos. Outra ameaça é que os resultados foram baseados no que foi concluído pelos autores em cada artigo pesquisado. Desse modo, não houve um processo para verificar as conclusões descritas em cada artigo.

Capítulo 4

Integração de Aspectos no Diagrama de Requisitos da SysML

O objetivo deste capítulo é propor um modelo para representar aspectos no nível de requisitos. O modelo é representado utilizando a linguagem de modelagem SysML. Na Seção 4.1 é realizada uma introdução à Engenharia de Requisitos Orientada a Aspectos. Na Seção 4.2 são apresentados os conceitos sobre o diagrama de Requisitos da SysML. Na Seção 4.3 é realizada a extensão do diagrama de Requisitos da SysML para representar aspectos. Na Seção 4.4 é apresentado um processo de como representar aspectos no nível de requisitos e na Seção 4.5 é apresentada uma comparação com outros modelos de Engenharia de Requisitos Orientada a Aspectos propostos na literatura.

4.1 Introdução à Engenharia de Requisitos Orientada a Aspectos

Similar ao que foi evidenciado pela comunidade de Programação Orientada a Aspectos, requisitos podem conter espalhamento e entrelaçamento, o que necessita de tratamento especial [Sampaio e Rashid 2008].

Um requisito aspectual é um interesse do *stakeholder* que interfere em outros requisitos ou artefatos [Rashid et al. 2006a].

A identificação de aspectos nos estágios iniciais de desenvolvimento de software pode ser dividida em duas categorias: aspectos funcionais e aspectos não-funcionais [Sampaio et al. 2007]. Requisitos não-funcionais são geralmente candidatos naturais a interesses transversais no nível de engenharia de requisitos uma vez que são propriedades de amplo escopo que tendem a restringir outros requisitos [Rashid et al. 2002], [Rashid et al. 2003].

De acordo com [Araújo et al. 2005], um aspecto no nível de requisitos afeta múltiplos outros requisitos no sistema de modo que:

- o aspecto pode restringir o comportamento do requisito afetado. Por exemplo, um

requisito de segurança pode restringir um outro requisito fornecendo acesso para certos tipos de dados no sistema de modo que apenas um conjunto certificado de usuários pode acessar os dados.

- o aspecto pode influenciar o requisito afetado com o objetivo de alterar seu comportamento especificado. Similarmente ao exemplo anterior, outro requisito de segurança pode influenciar requisitos de comunicação por alterar seu comportamento para impor restrições de criptografia.

De acordo com [Brito 2008], para realizar um framework orientado a aspectos no contexto de engenharia de requisitos é necessário suportar a separação de todos os interesses, incluindo os interesses transversais, definir mecanismos para especificar composição de interesses e manipular situações conflitantes, e suportar mecanismos para garantir rastreabilidade para o próximo estágio do processo de desenvolvimento. Ainda segundo esta mesma autora, similar a outras abordagens de Engenharia de Requisitos, a AORE consiste de elicitação, análise e especificação, validação, resolução de conflitos e gerenciamento de requisitos. A diferença fundamental entre AORE e a Engenharia de Requisitos Clássica é que a AORE manipula requisitos transversais. Para alcançar isso a AORE requer:

- existência de meios efetivos para identificar requisitos aspectuais em documentos de requisitos;
- habilidade para modularizar requisitos aspectuais;
- facilidades para obter um conjunto completo e consistente de descrições e representações de todos requisitos;
- habilidade para compor aspectos e requisitos não aspectuais;
- procedimento para identificar e resolver conflitos entre requisitos ou *stakeholders*;
- facilidades para rastrear requisitos aspectuais para estágios de desenvolvimento posteriores ou para sua fonte;
- um conjunto de ferramentas para facilitar o gerenciamento de requisitos aspectuais.

O principal problema encontrado na literatura é que a maioria dos trabalhos de AORE não apresentam rastreamento de requisitos aspectuais para estágios de desenvolvimento posteriores, principalmente com a arquitetura, sendo esta a principal motivação para a criação de um novo modelo neste trabalho. Na Seção 4.5 são apresentados mais detalhes de outros problemas encontrados em alguns trabalhos de AORE.

4.2 Diagrama de Requisitos da SysML

Um requisito na SysML é definido como um estereótipo (tipo) de Classe da UML sujeito a um conjunto de restrições [OMG-SysML 2010]. De acordo com a especificação da SysML, um requisito é definido conforme mostrado na Figura 4.1.

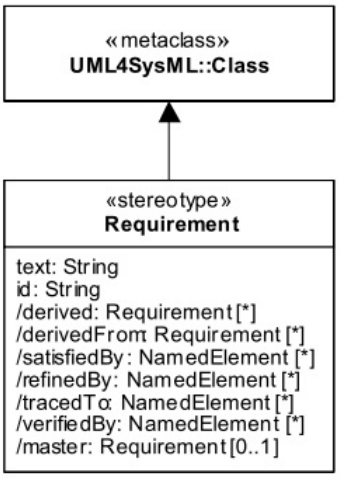

Figura 4.1: Modelo de requisitos da SysML [OMG-SysML 2010].

Os atributos do modelo de requisitos da SysML possuem os seguintes significados:

- *text: String*

  Representação textual do requisito.

- *id: String*

  Identificação única do requisito.

- */derived: Requirement [*]*

  Indica o requisito que foi derivado (cliente) a partir do requisito (fornecedor) em questão.

- */derivedFrom: Requirement [*]*

  Indica a partir de quais requisitos (fornecedores) o requisito em questão foi derivado (cliente).

- */satisfiedBy: NamedElement [*]*

  Indica os elementos (clientes) que satisfazem o requisito (fornecedor) em questão.

- */refinedBy: NamedElement [*]*

  Indica os elementos (clientes) que refinam o requisito (fornecedor) em questão.

- */tracedTo: NamedElement [*]*

  Indica todos os elementos (fornecedores) que são rastreáveis com o requisito (cliente) em questão.

- */verifiedBy: NamedElement [*]*

  Indica todos os elementos (clientes) que verificam o requisito (fornecedor) em questão.

- */master: Requirement [0..1]*

  Essa é uma propriedade derivada que lista o requisito mestre para o requisito escravo em questão. O atributo mestre é derivado a partir do fornecedor que possui uma dependência *Copy* (cópia) com o requisito escravo.

Os relacionamentos entre requisitos da SysML são definidos conforme mostrado na Figura 4.2.

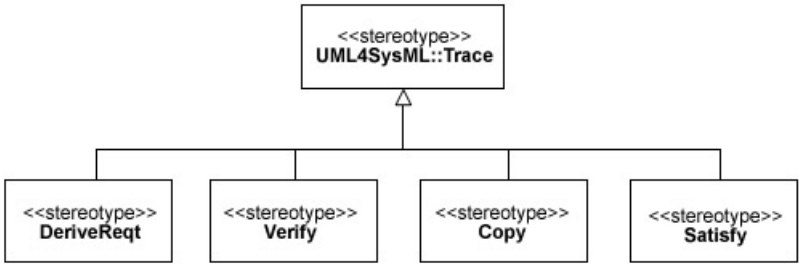

Figura 4.2: Modelo de relacionamentos de requisitos da SysML [OMG-SysML 2010].

Cada relacionamento é brevemente descrito a seguir:

- *deriveReqt* - um relacionamento *deriveReqt* (requisito derivado) é uma dependência entre dois requisitos no qual um requisito cliente pode ser derivado a partir do requisito fornecedor. Por exemplo, um requisito do sistema pode ser derivado a partir de uma necessidade de negócio, ou requisitos mais específicos podem ser derivados a partir de um requisito do sistema. Assim como outras dependências, a seta é direcionada a partir do requisito derivado (cliente) para o requisito do qual é derivado (fornecedor), como pode ser observado na Figura 4.3.

Figura 4.3: Relacionamento *deriveReqt* [OMG-SysML 2010].

- *verify* - um relacionamento *verify* (verificar) é uma dependência entre um requisito e um caso de teste ou outro elemento do modelo que pode determinar se um sistema realiza o requisito. A seta é direcionada a partir do elemento (cliente) para o requisito (fornecedor), como pode ser observado na Figura 4.4.

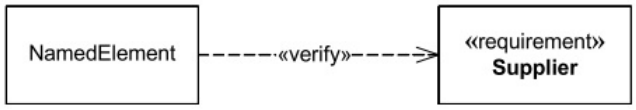

Figura 4.4: Relacionamento *verify* [OMG-SysML 2010].

- *copy* - um relacionamento *copy* (cópia) é uma dependência entre um requisito fornecedor e um requisito cliente que especifica que o texto do requisito cliente é uma cópia somente leitura do texto do requisito fornecedor. Uma dependência de cópia criada entre dois requisitos mantém um relacionamento mestre/escravo entre dois elementos para o propósito de reuso em diferentes contextos. O relacionamento pode ser observado na Figura 4.5. A seta é direcionada a partir do requisito escravo para o requisito mestre.

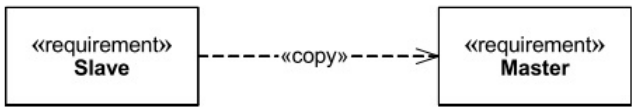

Figura 4.5: Relacionamento *copy* [OMG-SysML 2010].

- *satisfy* - um relacionamento *satisfy* (satisfazer) é uma dependência entre um requisito e um elemento do modelo que realiza o requisito. Assim como outras dependências, a seta é direcionada a partir do elemento do modelo que satisfaz (cliente) para o requisito que é satisfeito (fornecedor).

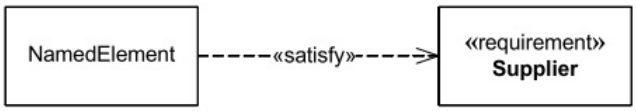

Figura 4.6: Relacionamento *satisfy* [OMG-SysML 2010].

Os relacionamentos de requisitos especificados permitem modelar a relação entre requisitos bem como com outros elementos do modelo [OMG-SysML 2010]. Isso permite a rastreabilidade entre diferentes níveis de desenvolvimento como arquitetura e projeto.

4.3 Extensão do Modelo de Requisitos da SysML para Representar Aspectos

De acordo com a revisão bibliográfica realizada não foram encontrados trabalhos publicados que apresentem a modelagem de aspectos no nível de requisitos com a SysML. A SysML é uma linguagem interessante de ser utilizada pois possui um diagrama específico para modelagem de requisitos. Uma característica interessante do diagrama de Requisitos da SysML é a possibilidade de modelar outros tipos de requisitos além dos funcionais, tais como requisitos não-funcionais [Soares et al. 2011], o que facilita a representação com aspectos já que podem ser criados estereótipos de requisitos.

Os requisitos da SysML podem aparecer em outros diagramas da linguagem para mostrar seu relacionamento com o projeto. Isso facilita representar a relação de aspectos

no nível de requisitos com aspectos no nível de arquitetura, sendo que essa relação não é apresentada na maioria dos trabalhos publicados na área de aspectos iniciais.

O metamodelo que define a representação de aspectos no nível de requisitos é uma extensão do metamodelo da SysML em que são definidos aspectos e seus elementos, as regras de composição e seus operadores. A extensão da SysML é realizada criando estereótipo de estereótipos, ou seja, subestereótipos que são similares à herança em UML. O subestereótipo herda as propriedades do superestereótipo e adiciona suas próprias. O subestereótipo é usado para representar o novo tipo de requisito que no caso é o requisito aspectual.

As próximas subseções apresentam as extensões realizadas no modelo de requisitos da SysML para representar aspectos.

4.3.1 Extensão dos Atributos

Considerando que há outros atributos importantes, além daqueles representados no modelo original, primeiramente o diagrama básico de requisitos da SysML é estendido com novos atributos. Essa extensão é baseada no modelo apresentado em [Soares et al. 2011] que considera um requisito estendido (≪ExtRequirement≫) com outros seis atributos. O modelo estendido pode ser observado na Figura 4.7.

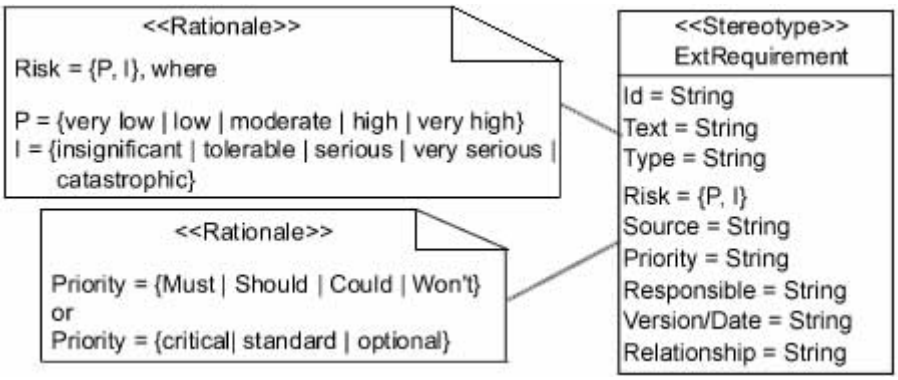

Figura 4.7: Extensão do modelo de requisitos [Soares et al. 2011].

Os atributos do requisito estendido possuem os seguintes significados:

- *Risk* (Risco) - risco é um evento ou condição incerta que, se ocorrer, tem um efeito positivo ou negativo nos objetivos de um projeto. Há pelo menos dois valores importantes a serem adicionados em risco: a probabilidade do risco se tornar real (representada por P) e os efeitos de sua ocorrência (representada por I).
- *Source* (Origem) - se o requisito é derivado de outro requisito, é útil saber sua origem. A propriedade *source* descreve onde o requisito derivado é originado. Essa informação é importante para rastrear requisitos durante o desenvolvimento do sistema.

- *Priority* (Prioridade) - conhecer quais requisitos possuem alta prioridade é útil para analisar riscos durante o desenvolvimento do sistema. A priorização de requisitos informa uma indicação da ordem que os requisitos devem ser abordados. Implementar todos os requisitos em um único lançamento do sistema pode não ser atrativo devido ao alto custo envolvido, falta de pessoal e tempo suficiente e até mesmo por pressão do mercado e do cliente. Essas dificuldades fazem a priorização uma atividade fundamental durante o processo de Engenharia de Requisitos [Greer 2005].
- *Responsible* (Responsável) - pelo menos o principal *stakeholder*, diretamente responsável pelo requisito, deve ser conhecido. No caso de haver mais de um *stakeholder* responsável, a opção é escrever todos eles ou apenas escrever os mais importantes.
- *Version/Date* (Versão/Data) - a versão de requisitos é útil para mostrar se os requisitos foram alterados. Esse atributo é fundamental, uma vez que mudanças descontroladas são a grande fonte de problemas em Engenharia de Requisitos. Além da versão a data de criação/alteração é adicionada.
- *Relationship* (Relacionamento) - tem o objetivo de melhorar a atividade de rastrear requisitos para o modelo de projeto. A propriedade relaciona o requisito específico para o modelo de projeto. Identificar e manter o rastreamento entre requisitos e projeto são consideradas importantes atividades em Engenharia de Requisitos.

Um outro atributo definido para o requisito aspectual, e que não foi definido no requisito estendido, é o atributo *Type*. Esse atributo indica qual o tipo do requisito, funcional ou não-funcional.

O modelo estendido para representar aspectos no nível de requisitos é apresentado na Figura 4.8. Para representar um aspecto no nível de requisitos foi criado o estereótipo ≪*Aspect Requirement*≫ que herda todos os atributos do requisito estendido. Os novos relacionamentos, que são apresentados e justificados na próxima seção, também são representados como atributos no diagrama de Requisitos Aspectuais. Dessa forma, no estereótipo ≪*Aspect Requirement*≫ são definidos os seguintes atributos:

- /BeforeOf: Requirement[*]

  Indica quais requisitos estão relacionados com o requisito aspectual em questão, sendo que este deve ser inserido *before* (antes) do requisito ao qual se relaciona.

- /AfterOf: Requirement[*]

  Indica quais requisitos estão relacionados com o requisito aspectual em questão, sendo que este deve ser inserido *after* (depois) do requisito ao qual se relaciona.

- /AroundOf: Requirement[*]

  Indica quais requisitos estão relacionados com o requisito aspectual em questão, sendo que este deve ser inserido *around* ("em volta") do requisito ao qual se relaciona.

- /ConflictWith: AspectRequirement[*]

  Indica quais requisitos aspectuais estão em conflito com o requisito aspectual em questão.

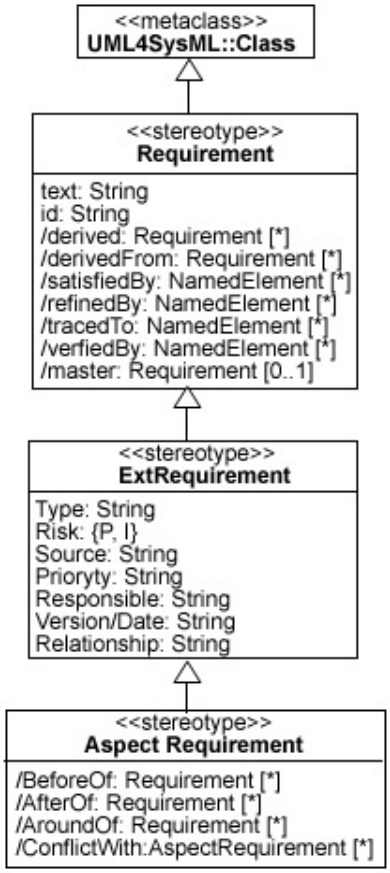

Figura 4.8: Metamodelo estendido.

4.3.2 Extensão dos Relacionamentos

Além dos atributos é também necessário definir novos relacionamentos que representem a composição de aspectos. Na fase de requisitos, composição significa tanto a recombinação de requisitos descritos separadamente quanto a consideração de confrontos entre a semântica desses requisitos [Araújo e Baniassad 2007], ou seja, como um requisito aspectual influencia ou restringe o comportamento de um requisito não aspectual.

Para definir esses relacionamentos alguns trabalhos anteriormente publicados foram analisados.

Segundo [Araújo et al. 2002], as abordagens de *overlapping*, *overriding* e *wrapping* são frequentemente usadas em várias outras abordagens de separação de interesses para definir a parte da composição do modelo.

- *Overlap*: os requisitos aspectuais modificam os requisitos funcionais que eles interferem. Nesse caso, os requisitos aspectuais podem ser necessários antes que os funcionais ou eles podem ser necessários depois deles.
- *Override*: os requisitos aspectuais sobrepõem os requisitos funcionais que eles interferem. Nesse caso, o comportamento descrito pelo requisito aspectual substitui o comportamento dos requisitos funcionais.
- *Wrap*: os requisitos aspectuais "encapsulam" os requisitos funcionais que eles interferem. Nesse caso, o comportamento descrito pelos requisitos funcionais é encapsulado pelo comportamento descrito pelos requisitos aspectuais.

Em [Brito 2008] é definida uma regra de composição inspirada na linguagem LOTOS (*Language Of Temporal Ordering Specification*). As regras são:

- *Enabling* (denotado por C1 $\gg$ C2): Essa é uma composição sequencial e significa que o comportamento de C2 inicia se e somente se C1 termina com sucesso.
- *Disabling* (denotado por C1 [> C2): Essa é uma composição de desativação e significa que C2 interrompe o comportamento de C1 quando ele inicia seu próprio comportamento. Isso permite a representação de interrupções.
- *Full synchronization* (denotado por C1 || C2): Esse é um operador paralelo e significa que o comportamento de C1 deve ser sincronizado com o comportamento de C2. Ele representa composição concorrente com sequência.
- *Pure interleaving* (denotado por C1|||C2): Esse é um operador paralelo e significa que C1 evolui separadamente de C2, ou seja, representa composição concorrente sem interação.

Em [Sánchez et al. 2010] também são apresentadas regras de composição baseadas na linguagem LOTOS. As regras são:

- *Enable* (T1$\gg$T2): composição sequencial.
- *Disable* (T1 [> T2): o comportamento de T1 é substituído pelo comportamento de T2.
- *Parallel* (T1 || T2): os comportamentos de T1 e T2 devem ser sincronizados.
- *Choice* (T1 [ ] T2): somente um dos interesses será satisfeito (T1 ou T2).

Em [Chitchyan et al. 2007b] também são descritos relacionamentos de ordem temporal entre aspectos e requisitos. Os relacionamentos são:

- X *before* (*after*) Y - Há um intervalo temporal entre os requisitos X e Y. X foi completado e Y ainda não foi iniciado.

- X *meets* (*met by*) Y - Não há intervalo temporal entre requisitos. X termina e o requisito Y inicia (a partir da perspectiva de X).
- X *overlaps* (*overlapped by*) Y - Requisito X inicia antes do requisito Y; Y começa enquanto X está em processo; X completa enquanto Y está em processo (a partir da perspectiva de X).
- X *during* (*through*) Y - Requisito Y inicia antes do requisito X; X começa enquanto Y está em processo; X completa enquanto Y está em processo (a partir da perspectiva de X).
- X *starts* (*started by*) Y - Requisito X inicia simultaneamente com requisito Y; X completa enquanto Y está em processo (a partir da perspective de X). Esse é um sub-tipo de *during*.
- X *finishes* (*finished by*) Y- Requisito Y inicia antes do requisito X; X começa enquanto Y está em processo; X e Y completam simultaneamente (a partir da perspectiva de X). Esse é um sub-tipo de *during*.
- X *concurrent* Y - Requisitos X e Y são iniciados e completados com o mesmo exato intervalo temporal.

No nível de arquitetura, assim como no nível de implementação, os relacionamentos *before*, *after* e *around* são frequentemente utilizados para representar a composição de um aspecto e um elemento não aspectual [Pinto et al. 2011]. Como citado no Capítulo 2, o tipo *before* significa que o aspecto é executado antes do elemento não aspectual, o tipo *after* significa que o aspecto é executado depois do elemento não aspectual, o tipo *around* pode substituir a execução do elemento não aspectual ou pode executar parte do aspecto antes e outra parte depois do elemento não aspectual.

Em AspectJ, que é uma das linguagens de programação orientada a aspectos mais conhecidas, o relacionamento *after* é subdividido em três tipos: o aspecto pode ser executado depois que o elemento não aspectual é completado normalmente, o aspecto pode ser executado depois que o elemento não aspectual dispara uma exceção, ou pode ser executado depois que faz ou um ou outro [AspectJ 2012].

Analisando os relacionamentos para composição de requisitos citados anteriormente pode-se observar que eles são equivalentes aos relacionamentos *before*, *after* e *around*. Em [Chitchyan et al. 2007a] é mostrada uma tabela que faz o relacionamento dos operadores da linguagem de descrição de requisitos apresentados em [Chitchyan et al. 2007b] com os operadores *before*, *around*, *after* e *concurrent-with* (que é um outro tipo utilizado no trabalho citado). Devido a essa característica e para manter a rastreabilidade dos aspectos no nível de requisitos com os aspectos no nível de arquitetura, os relacionamentos *before*, *after* e *around* são utilizados neste trabalho para representar os relacionamentos de composição no nível de requisitos.

Outro relacionamento utilizado neste trabalho é *conflict*. Como já citado na seção 4.1 um procedimento importante que se deve ter na Engenharia de Requisitos Orientada a Aspectos é a identificação e resolução de conflitos entre requisitos aspectuais. O relacionamento *conflict* vai colaborar para a identificação de conflitos no modelo proposto de requisitos com aspectos.

O modelo de relacionamentos de requisitos estendido com os novos relacionamentos é mostrado na Figura 4.9.

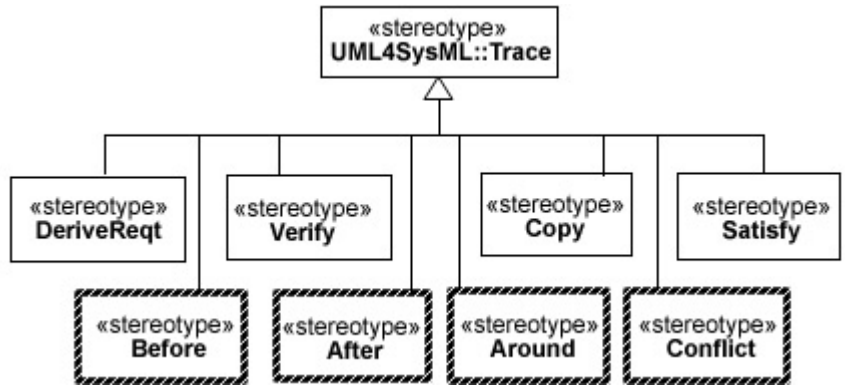

Figura 4.9: Modelo de relacionamentos estendido.

No novo modelo de relacionamento de requisitos, o relacionamento *before* é uma dependência entre um requisito aspectual (cliente) e um requisito não aspectual (fornecedor). Ele indica que o requisito aspectual deve ser executado antes do requisito não aspectual. A representação do relacionamento *before* pode ser observada na Figura 4.10.

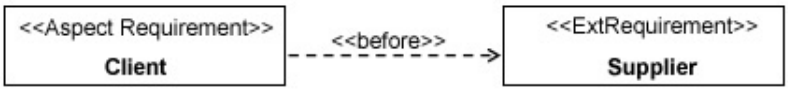

Figura 4.10: Relacionamento *before*.

O relacionamento *after* é uma dependência entre um requisito aspectual (cliente) e um requisito não aspectual (fornecedor). Ele indica que o requisito aspectual deve ser executado depois do requisito não aspectual. A representação do relacionamento *after* pode ser observada na Figura 4.11.

Figura 4.11: Relacionamento *after*.

Um relacionamento *around* é uma dependência entre um requisito aspectual (cliente) e um requisito não aspectual (fornecedor). Ele indica que o requisito aspectual sobrepõe

o requisito não aspectual. Nesse caso, o comportamento descrito pelo requisito aspectual substitui o comportamento do requisito não aspectual ou pode executar parte do aspecto antes e outra parte depois do requisito não aspectual. A representação do relacionamento pode ser observada na Figura 4.12.

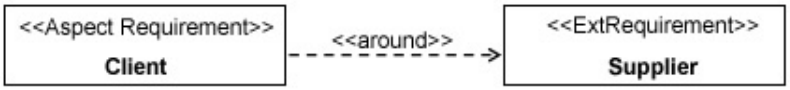

Figura 4.12: Relacionamento *around*.

Um relacionamento *conflict* (conflito) é uma dependência entre um requisito aspectual (fornecedor/cliente) e outro requisito aspectual (fornecedor/cliente). O relacionamento *conflict* foi criado para indicar se há *trade-off*, ou seja, conflito de escolhas entre dois requisitos aspectuais. A representação do relacionamento *conflit* é mostrada na Figura 4.13.

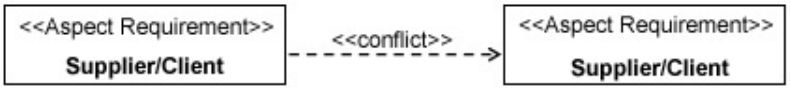

Figura 4.13: Relacionamento *conflict*

4.3.3 Utilizando Pacotes

Para melhorar a visualização do diagrama, os requisitos aspectuais podem ser representados dentro de um pacote. No diagrama de Pacotes são estabelecidas dependências entre os pacotes e/ou elementos do modelo dentro dos pacotes [OMG-SysML 2010]. Os pacotes da SysML são especificados com uma camada de extensão do metamodelo UML. Um exemplo do diagrama de Pacotes é mostrado na Figura 4.14.

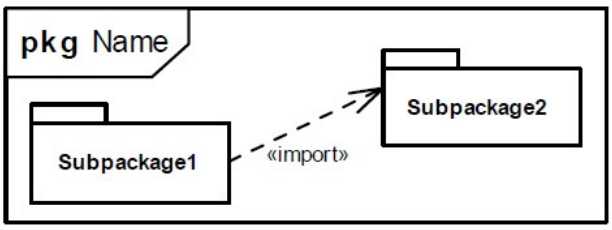

Figura 4.14: Exemplo de pacote [OMG-SysML 2010].

Para mostrar o relacionamento entre um pacote de aspectos e um requisito estendido o relacionamento ≪*crosscut*≫ é utilizado. Para criar este relacionamento o modelo de dependência foi estendido como é mostrado na Figura 4.15.

Na Figura 4.16 é mostrada a representação do relacionamento *crosscut*.

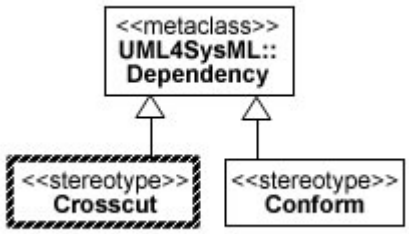

Figura 4.15: Extensão do relacionamento de dependência.

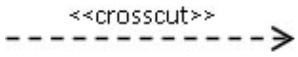

Figura 4.16: Representação do relacionamento *crosscut*.

4.4 Processo para Representar Aspectos no Nível de Requisitos

Uma vez que aspectos no nível de requisitos foram identificados, é importante representá-los, bem como especificar seu impacto e influência em outro requisito do sistema [Rashid et al. 2006b]. Isso é realizado utilizando uma composição que deve especificar como requisitos aspectuais influenciam ou restringem outros requisitos na especificação.

O processo para representar aspectos no nível de requisitos é adaptado da proposta de [Rashid et al. 2003]. O processo pode ser observado na Figura 4.17.

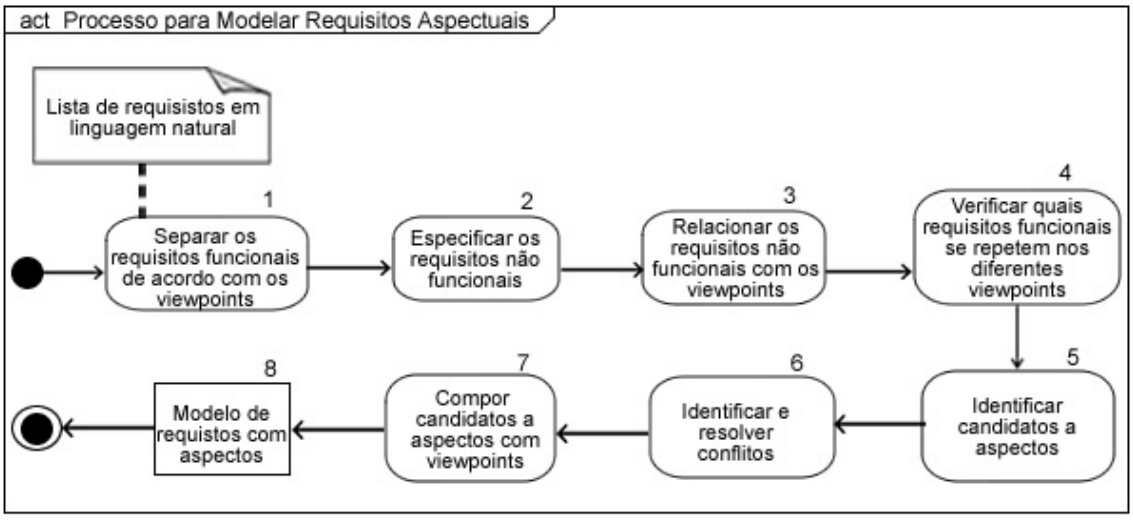

Figura 4.17: Processo para modelar requisitos aspectuais.

No estágio 1 os requisitos funcionais são separados de acordo com os *viewpoints*. *Viewpoints* geralmente representam *stakeholders* ou outros sistemas que tem um interesse específico no sistema corrente [Sampaio et al. 2007]. Os requisitos elicitados representados em linguagem natural são a entrada para o modelo. Para representar os *viewpoints*

são utilizados *viewpoint* e *view* da SysML e também o modelo estendido para requisitos (≪*ExtRequirement*≫). Um exemplo de *viewpoint* pode ser observado na Figura 4.18.

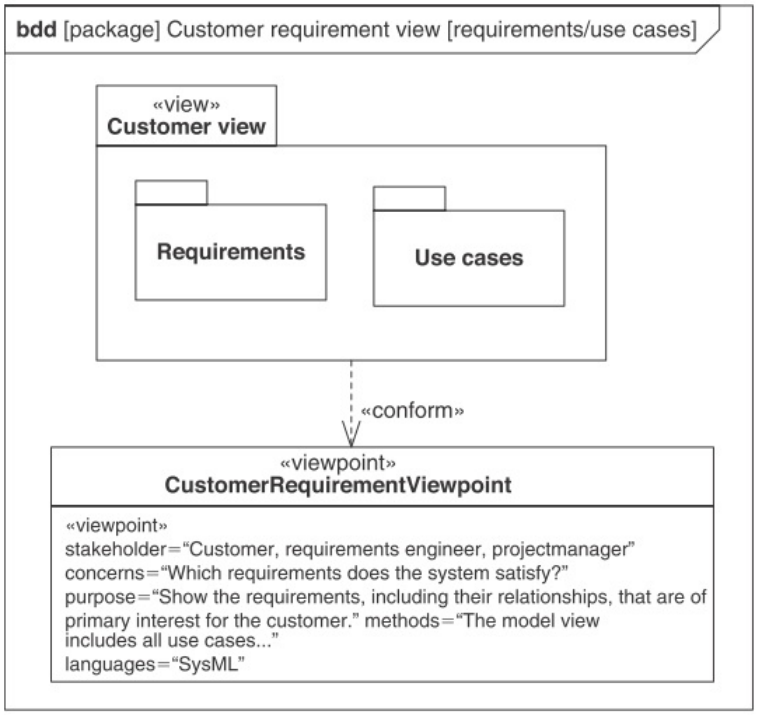

Figura 4.18: Exemplo de *viewpoint* [OMG-SysML 2010].

No estágio 2 os requisitos não-funcionais são representados separadamente, pois segundo [Sampaio et al. 2007], os requisitos não-funcionais são geralmente candidatos naturais a aspectos no nível de engenharia de requisitos, uma vez que são propriedades de amplo escopo que tendem a restringir outros requisitos. Para representar requisitos não-funcionais é utilizado o diagrama de Requisitos da SysML considerando o modelo estendido (≪ExtRequirement≫).

No estágio 3 os requisitos não-funcionais são relacionados com os *viewpoints*. Essa é uma forma de identificar quais requisitos não-funcionais afetam os funcionais e assim permite verificar quais são os candidatos a aspecto. Esse relacionamento é realizado por meio da tabela de requisitos da SysML estendida. O formato tabular facilita no rastreamento dos requisitos durante o ciclo de vida do software. A rastreabilidade ajuda na identificação da origem, destino e ligações entre requisitos e aspectos. Na Tabela 4.1 é mostrada a tabela de requisitos da SysML estendida para relacionar requisitos não-funcionais com *viewpoint*. A tabela é composta pela identificação do requisito não-funcional (id1), nome do requisito não-funcional, identificação do requisito funcional (id2), nome do requisito funcional e nome do *viewpoint* que o requisito funcional pertence. A coluna *relation* é opcional. Ela apenas indica que há uma relação entre um requisito não-funcional e um *viewpoint*.

No estágio 4 os requisitos funcionais que se repetem em diferentes *viewpoints* são

id1	*non-functional requirement*	*relation*	*id2*	*functional requirement*	*viewpoint*

Tabela 4.1: Tabela da SysML estendida para relacionar requisitos não-funcionais com *viewpoint*.

identificados. Essa verificação é realizada para identificar os requisitos funcionais que podem ser candidatos a aspectos. Caso seja encontrado algum requisito funcional que se repete em mais de um *viewpoint* este também é registrado em uma tabela da SysML estendida conforme é mostrado na Tabela 4.2. A coluna *repeat* é opcional. Ela apenas indica a repetição de requisitos funcionais.

id1	*functional requirement1*	*viewpoint1*	*repeat*	*id2*	*functional requirement2*	*viewpoint2*

Tabela 4.2: Tabela da SysML estendida para relacionar requisitos funcionais com requisitos funcionais.

No estágio 5 os candidatos a aspectos são identificados. Para os candidatos a aspectos não-funcionais essa identificação é realizada verificando quais os requisitos não-funcionais se inserem em mais de um *viewpoint*. A verificação é realizada analisando a Tabela 4.1. Para os candidatos a aspectos funcionais essa identificação é realizada verificando quais os requisitos funcionais se repetem em mais de um *viewpoint*. A verificação é realizada analisando a Tabela 4.2. Candidatos a aspectos são registrados na Tabela 4.3. Nessa tabela a coluna *relationship* é preenchida com os tipos de relacionamentos que podem ser *before*, *after* ou *around*.

id1	*aspect*	*relation*	*id2*	*functional requirement*	*viewpoint*

Tabela 4.3: Tabela da SysML estendida para representar o relacionamento de candidatos a aspectos com *viewpoints*.

No estágio 6 conflitos de interesses entre os aspectos são identificados. A composição de interesses aspectuais com interesses base podem aumentar situações de conflito que necessitam ser identificados e resolvidos. Um conflito é detectado sempre que dois ou mais interesses contribuem negativamente um para o outro e, no entanto, possuem a mesma importância na composição [Brito et al. 2007]. Por exemplo, segurança e tempo de resposta podem estar presentes em um mesmo requisito de um *viewpoint* e podem contribuir negativamente um com o outro. Com o objetivo de manter a rastreabilidade de conflitos os mesmos podem ser também representados em uma tabela estendida da SysML conforme é mostrado na Tabela 4.4. A coluna *conflict* é opcional. Ela apenas indica que há conflito entre aspectos.

No estágio 7 os candidatos a aspectos são compostos com os *viewpoints*. As tabelas definidas anteriormente já indicam a composição dos aspectos com os requisitos. No entanto, nessa etapa os aspectos são compostos no modelo gráfico da representação de

id1	*aspect1*	*conflict*	*id2*	*aspect2*	*viewpoint*

Tabela 4.4: Tabela da SysML estendida para representar conflitos entre requisitos aspectuais.

requisitos e para isso são utilizadas as extensões para aspectos do diagrama de Requisitos da SysML.

Finalmente no estágio 8 o modelo de requisitos com aspectos é obtido.

4.5 Comparação com Outros Modelos de Engenharia de Requisitos Orientada a Aspectos

Para avaliar o modelo de requisitos com aspectos proposto nesse trabalho é apresentada na Tabela 4.5 uma comparação com outros cinco modelos de requisitos com aspectos. Esses modelos foram escolhidos baseados na revisão bibliográfica sendo selecionados os trabalhos mais citados de acordo com o número de citações apresentado na ferramenta de busca Google. Os trabalhos são:

- A - *Modularisation and Composition of Aspectual Requirements* [Rashid et al. 2003];
- B - *Theme: An Approach for Aspect-Oriented Analysis and Design* [Baniassad e Clarke 2004];
- C - *Scenario Modelling with Aspects* [Whittle e Araujo 2004];
- D - *Semantics-based Composition for Aspect-Oriented Requirements Engineering* [Chitchyan et al. 2007b];
- E - *Mining Aspects in Requirements* [Sampaio et al. 2005];
- F - Integração de Aspectos no Diagrama de Requisitos da SysML (modelo apresentado neste trabalho).

Os seguintes símbolos são utilizados:
■ - significa que o critério avaliado é totalmente satisfeito;
⊡ - significa que o critério avaliado é parcialmente satisfeito;
□ - significa que o critério avaliado não é satisfeito.

No modelo A, apresentado por [Rashid et al. 2003], é definido um processo para identificar os requisitos transversais, no entanto não são identificados os requisitos transversais funcionais. O modelo apresenta um conjunto de regras de composição que define o relacionamento entre requisitos e requisitos aspectuais. No entanto não apresenta relacionamento entre requisitos. Um processo para identificar e resolver conflitos é apresentado. É utilizada uma tabela que mostra de qual forma um aspecto afeta outro. Para resolver conflitos são atribuídos pesos aos aspectos conflitantes. Os pesos variam de 0 a 10, em que

Critérios de Comparação	A	B	C	D	E	F
Processo para identificar requisitos transversais	■	■	■	□	■	■
Identificação de requisitos transversais funcionais e não-funcionais	⊡	■	⊡	□	□	■
Composição de Aspectos com Requisitos	■	□	■	■	□	■
Identificação de Conflitos	■	□	■	■	□	⊡
Resolução de Conflitos	■	□	□	□	□	□
Modelagem Gráfica	□	■	■	□	□	■
Diagrama de Requisitos Específico	□	□	□	□	□	■
Relacionamento entre requisitos	□	□	□	□	□	■
Relação com a UML	□	□	■	□	□	■
Extensibilidade do Modelo	■	□	■	■	□	■
Relacionamento com a Arquitetura	■	□	□	□	□	■
Rastreabilidade com o Projeto	■	■	□	□	□	□
Suporte a Automatização	□	⊡	⊡	⊡	⊡	□

Tabela 4.5: Comparação entre modelos de requisitos com aspectos.

o aspecto com o maior peso é o mais importante. O modelo é representado pela linguagem XML e não possui modelagem gráfica. Assim, não possui um diagrama específico para requisitos. XML é uma linguagem extensível. Sendo assim, é possível impor restrições sobre a especificação de regras de composição quando novos operadores e/ou ações são introduzidos. O modelo apresenta uma tabela listando os aspectos e informando se esses aspectos influenciam em outros estágios de desenvolvimento.

O modelo B, apresentado por [Baniassad e Clarke 2004], é baseado em linguagem natural. É apresentada uma ferramenta semi-automática para a identificação de comportamentos transversais em especificações de requisitos. O processo de identificação é baseado em uma análise léxica do documento de requisitos, no qual pesquisa o documento por palavras-chave fornecidas pelo desenvolvedor. A ferramenta produz automaticamente uma visão gráfica que mapeia o relacionamento entre o comportamento, que pode ajudar o desenvolvedor a identificar os candidatos a aspectos. No entanto, a representação gráfica não modela explicitamente os aspectos, ela serve apenas para ajudar o desenvolvedor a identificar os aspectos. O modelo também não apresenta um processo para compor requisitos com requisitos aspectuais. Também não é apresentado um método para identificar e resolver conflitos. No trabalho é apresentada outra abordagem chamada Theme/UML para representar aspectos no nível de projeto. No entanto, no nível de requisitos não há relação com a UML.

A ideia do modelo C, apresentado por [Whittle e Araujo 2004], é compor requisitos baseados em cenários e aspectos também representados como cenários. Cenários aspectuais são representados como IPS (*Interaction Pattern Specifications*). A notação para PS (*Pattern Specifications*) é baseada na linguagem UML. Cenários não aspectuais são representados como diagramas de Sequência da UML. São também utilizados casos de

uso na representação. A identificação de aspectos é realizada verificando quais casos de uso são afetados por requisitos não-funcionais. A identificação de conflitos é realizada através de protótipagem, mas não é detalhado no trabalho como o processo é realizado. É apresentada uma técnica para compor cenários aspectuais e não aspectuais usando uma instância de IPS e síntese de máquina de estado. A coleção de cenários pode então ser traduzida automaticamente em um conjunto de máquina de estado usando um algoritmo de síntese de máquina de estado. O modelo não apresenta relação nem com a arquitetura nem com o projeto de software.

No modelo D, definido em [Chitchyan et al. 2007b], é apresentada uma linguagem de descrição de requisitos (RDL) enriquecendo uma especificação de requisitos em linguagem natural com informações semânticas. As informações são derivadas a partir da própria semântica da linguagem natural. No trabalho não é apresentado um processo para identificar os aspectos mas apenas para compor aspectos. Especificações de composição são escritas baseadas nessas semânticas ao invés de sintaxe de requisitos. A RDL é baseada em visões simétricas de AOSD, ou seja, é utilizada a mesma abstração. Um mesmo interesse representa tanto elementos transversais e não transversais.

O principal objetivo da abordagem E, apresentada em [Sampaio et al. 2005], é determinar potenciais candidatos a aspectos em documentos de requisitos indiferente de como eles são estruturados. A abordagem utiliza técnicas de processamento de linguagem natural que fornecem suporte para análise sensitiva de contexto de requisitos. Uma ferramenta de suporte é fornecida para ajudar o desenvolvedor a automaticamente minerar e modelar os interesses transversais sem ter anteriormente lido o documento de requisitos. A tarefa de identificar requisitos transversais pode ser suportada em diferentes formas pela ferramenta. A ferramenta pode automaticamente procurar por palavras que indicam ações, as quais podem ser identificadas como verbos pela ferramenta WMATRIX e produz um modelo que representa o relacionamento entre os requisitos. No trabalho não é definido que tipo de requisito aspectual é identificado, funcional ou não-funcional. No trabalho também não é apresentado um modelo de composição e a identificação de conflitos. Dos critérios avaliados na Tabela 4.5, o trabalho apresenta somente um processo para identificar requisitos transversais e fornece suporte a automatização.

No modelo proposto neste trabalho é apresentado um processo para identificar aspectos tanto de origem funcional como não-funcional. Com os novos relacionamentos criados no modelo de requisitos da SysML é realizada a composição de requisitos com requisitos aspectuais. No modelo é possível indicar se há conflitos entre aspectos utilizando um novo relacionamento criado. No entanto não é apresentado uma forma de resolver esses conflitos. A modelagem gráfica é realizada com as extensões da SysML. A SysML possui um diagrama específico de requisitos e relacionamento entre requisitos. Com os relacionamentos da SysML também é possível relacionar requisitos com a arquitetura. Este modelo não possui suporte a automatização e não foi definido como é a relação de aspectos no

nível de projeto.

Capítulo 5

Arquitetura de Software Orientada a Aspectos com SysML

Este capítulo foi baseado no artigo *"Extensions of SysML for Modeling an Aspect Oriented Software Architecture with Multiple Views"* aceito na *10th International Conference on Information Technology : New Generations*. Este capítulo tem o objetivo de propor um modelo de arquitetura de software orientada a aspectos com múltiplas visões. A linguagem de modelagem SysML foi novamente estendida para representar a arquitetura. Na Seção 5.1 é realizada uma introdução à arquitetura de software orientada a aspectos. Na Seção 5.2 é apresentada a proposta de uma arquitetura orientada a aspectos com múltiplas visões. Na Seção 5.3 são apresentadas as extensões da linguagem de modelagem SysML para representar a arquitetura proposta e na Seção 5.4 é apresentada uma comparação com outros modelos de arquitetura orientada a aspectos propostos na literatura.

5.1 Introdução à Arquitetura de Software Orientada a Aspectos

Uma Arquitetura de Software Orientada a Aspectos tem como foco a localização e a especificação de interesses transversais no projeto da arquitetura. Um aspecto na arquitetura é um interesse que "corta transversalmente" artefatos arquiteturais [Baniassad et al. 2006].

A arquitetura de software orientada a aspectos é foco de vários trabalhos de pesquisa, pois identificar aspectos ainda no projeto da arquitetura pode melhorar a modularidade do projeto de software, detectar inicialmente conflitos de interesses, reduzir custos de manutenção do software e preservar a noção de aspectos no processo de desenvolvimento garantindo a rastreabilidade [Navasa et al. 2002].

A arquitetura de software orientada a aspectos pode ser classificada em duas principais abordagens: a abordagem simétrica, que adapta o espaço de interações para manipular

a orientação a aspectos, o que significa que não é feita uma distinção explícita entre aspectos e componentes não aspectuais [Garcia et al. 2009]; e a abordagem assimétrica, que introduz elementos adicionais de modelo orientado a aspecto tais como componentes aspectuais, interfaces aspectuais e conectores aspectuais. Nessa abordagem é feita uma distinção explícita entre aspectos e componentes não aspectuais [Bennouar et al. 2010]. Este trabalho utiliza uma abordagem assimétrica, pois há uma clara distinção entre a estrutura convencional e a estrutura de aspectos.

A arquitetura de software orientada a aspectos, assim como a programação orientada a aspectos, usa conceitos como *joinpoint*, *pointcut*, *advice*, tipo de *advice* e *weaver* [Miles 2005], que foram citados no Capítulo 2. A composição de aspectos requer a identificação de *joinpoints* relevantes arquiteturalmente nos quais aspectos e outros elementos arquiteturais são naturalmente combinados juntos.

Arquiteturas de software podem separar interesses transversais mais apropriadamente se for utilizada uma ADL orientada a aspectos (AO-ADL) [Pinto et al. 2012].

A maioria das AO-ADL (incluindo as linguagens de modelagem) são motivadas pela integração dos conceitos existentes das linguagens de descrição de arquitetura (tais como, componentes, interfaces, conectores e configurações) com novas abstrações orientadas a aspecto (tais como, aspectos, *joinpoint*, *pointcuts* e *advice*) com o objetivo de abordar a modelagem de interesses transversais na arquitetura [Batista et al. 2006].

Há uma diversidade de pontos de vista de como aspectos devem ser modelados nas ADLs. Geralmente os aspectos são representados por componentes [Batista et al. 2006]. No entanto, qualquer adaptação de ADLs existentes ou a engenharia de novas ADLs devem ser baseadas em um claro entendimento do que os desafios de aspectos representam no nível de arquitetura que não podem ser manipulados com abstrações das ADLs existentes [Batista et al. 2006].

Abordagens correntes orientadas a aspectos no nível de arquitetura tendem a imitar modelos de *joinpoint* da linguagem de programação, porém negligenciam conceitos de arquitetura [Chavez et al. 2009]. Como há diferentes visões no processo de projeto de arquitetura convencional, é necessário também considerar essas visões no projeto de arquitetura orientada a aspectos [Tekinerdogan et al. 2007]. O uso de múltiplas visões permite abordar separadamente os interesses de *stakeholders*: usuário final, desenvolvedores, engenheiros de sistemas, e manipular separadamente os requisitos funcionais e não funcionais [Kruchten 1995]. Em muitos trabalhos o conceito de múltiplas visões não é utilizado. Na Seção 5.4 são mostrados outros problemas encontrados em alguns trabalhos de arquitetura com aspectos, como por exemplo falta de rastreabilidade com aspectos no nível de requisitos.

5.2 Proposta de uma Arquitetura de Software Orientada a Aspectos com Múltiplas Visões

Considerando que diferentes *stakeholders* necessitam visualizar o software a partir de várias perspectivas, a utilização de uma arquitetura em múltiplas visões é fundamental. Por isso, este trabalho propõe usar um modelo de arquitetura de software orientado a aspectos com múltiplas visões para definir a estrutura de softwares orientado a aspectos. A arquitetura é obtida em conformidade com o modelo de requisitos com aspectos que foi definido no capítulo 4.

Para representar a arquitetura orientada a aspectos as visões escolhidas são a visão estrutural, a visão de desenvolvimento e a visão de cenários + requisitos, como pode ser observado na Figura 5.1.

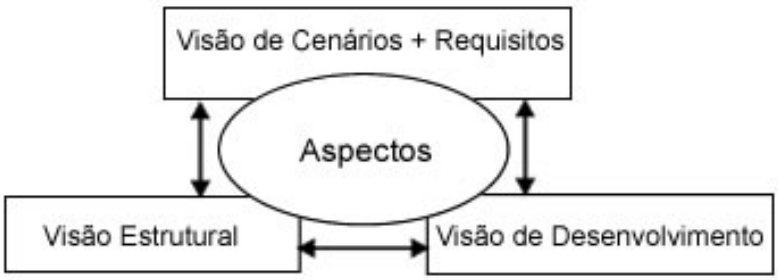

Figura 5.1: Arquitetura orientada a aspectos em múltiplas visões.

No modelo de arquitetura orientada a aspectos, as visões estão relacionadas umas com as outras e aspectos estão inseridos em todas as visões.

O objetivo da visão estrutural é representar o sistema de forma lógica considerando para isso os requisitos funcionais e como os aspectos se relacionam com esses requisitos.

A visão de desenvolvimento objetiva descrever o ambiente de desenvolvimento definindo a plataforma de hardware/software que dará suporte ao sistema e também a definição dos subsistemas, módulos, pacotes e camadas, e a interação de uns com os outros. A visão de desenvolvimento também define claramente as opções tecnológicas como hardware, sistema operacional, rede, banco de dados e linguagens.

A visão de cenários é obtida a partir dos casos de uso que ilustram as interações de um ator com o sistema. Um ator pode ser um humano ou entidade máquina que interage com o sistema para executar um trabalho significativo. Associado à visão de cenários estão os requisitos que são modelados juntamente com aspectos.

As visões definidas na arquitetura orientada a aspectos se relacionam com o modelo de requisitos definido no capítulo 4 conforme é mostrado na Figura 5.2.

A visão de Cenários + Requisitos está relacionada com os requisitos do usuário. Os requisitos do usuário são representações em alto nível e indicam o que o sistema irá fazer,

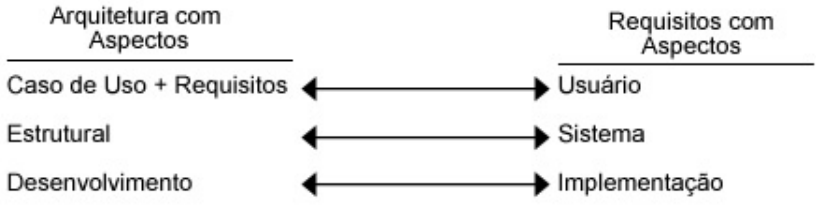

Figura 5.2: Relacionamento entre requisitos e arquitetura.

ou seja, as funcionalidades e restrições a partir do ponto de vista do usuário.

A visão estrutural está relacionada com os requisitos do sistema. O propósito dos requisitos de sistema é refinar os requisitos do usuário aumentando o nível de detalhes [Soares e Vrancken 2011]. Estes requisitos são usados como entradas para projetar software usando diagramas de uma linguagem de modelagem.

A visão de desenvolvimento está relacionada com os requisitos de implementação. No nível de implementação os requisitos devem ser especificados de maneira detalhada. O nível de detalhes é próximo às especificações de algoritmo [Soares e Vrancken 2011].

5.3 Utilização da SysML para modelar as visões arquiteturais

Assim como na representação dos requisitos, a arquitetura é representada pela SysML que novamente foi estendida para expressar os relacionamentos de aspectos com os outros elementos convencionais.

As próximas subseções descrevem a arquitetura orientada a aspectos em múltiplas visões usando uma extensão da SysML como linguagem de modelagem.

5.3.1 Visão de Cenários

A visão de cenários é realizada por meio de casos de uso. Segundo [Jacobson 1992], casos de uso são uma sequência de ações realizadas pelo sistema para produzir um resultado observável de valores para um usuário em particular.

Para que seja possível representar aspectos na visão de cenários o diagrama de Casos de Uso foi estendido. Para representar casos de uso com aspectos foi criado um estereótipo do tipo $\ll$*Aspect UseCase*$\gg$, como é representado na Figura 5.3. Na Figura 5.4 é mostrado como fica a representação do estereótipo no caso de uso.

Para relacionar um caso de uso com aspecto e um caso de uso comum o relacionamento $\ll$*crosscut*$\gg$ definido no Capítulo 4 é utilizado.

Relacionada à visão de cenários estão os requisitos que também são modelados usando aspectos conforme definido no Capítulo 4.

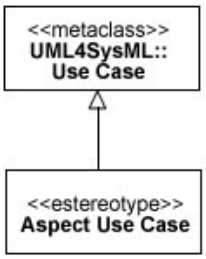

Figura 5.3: Modelo de caso de uso estendido.

Figura 5.4: Estereótipo em casos de uso.

Casos de uso com aspectos podem ser representados dentro de um pacote para melhorar a visualização gráfica. O relacionamento $\ll crosscut \gg$ é também aplicado para conectar um pacote de casos de uso com aspectos e um caso de uso comum.

5.3.2 Visão Estrutural

Para a representação da visão estrutural foi utilizado o diagrama de Blocos da SysML com extensões. O diagrama de definição de bloco é um estereótipo do diagrama de Classe da UML com restrições e extensões adicionais definidas pela SysML [OMG-SysML 2010].

Blocos da SysML podem ser usados em todas as fases de especificação e projeto do sistema e podem ser aplicados para muitos diferentes tipos de sistemas [OMG-SysML 2010]. Blocos descrevem elementos relacionados à estrutura de um sistema. Esses elementos podem ser de diferentes níveis de abstração. Blocos da SysML possuem informações sobre o próprio bloco (atributo), ou eles referenciam outros blocos que se encontram vinculados (associação). Adicional à estrutura estática, blocos também descrevem operações, ou seja, o comportamento que eles podem executar [Weilkiens 2008]. O modelo de Bloco da SysML é mostrado na Figura 5.5.

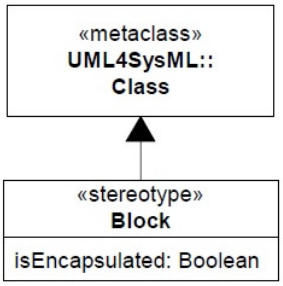

Figura 5.5: Modelo de bloco da SysML [OMG-SysML 2010].

No bloco é definido o atributo *isEncapsulated*. Se este atributo é verdadeiro, então o bloco é tratado como uma caixa preta. Uma parte tipada por essa caixa preta pode

somente ser conectada via suas portas ou diretamente para seu limite exterior. Se falso ou se um valor não está presente então conexões podem ser estabelecidas para elementos de sua estrutura externa através das extremidades dos conectores aninhados [OMG-SysML 2010].

Os Blocos da SysML permitem ter múltiplos compartimentos, cada um identificado opcionalmente com o próprio nome do compartimento. O compartimento pode dividir as características mostradas de acordo com vários critérios. Alguns compartimentos padrões são definidos pela própria SysML e outros podem ser definidos pelo usuário. Compartimentos podem aparecer em qualquer ordem. A SysML define dois compartimentos adicionais, *namespace* e *structure*, os quais podem conter nós gráficos ao invés de restrições textuais ou definições de características. Na Figura 5.6 é mostrado o modelo de Bloco da SysML com compartimentos.

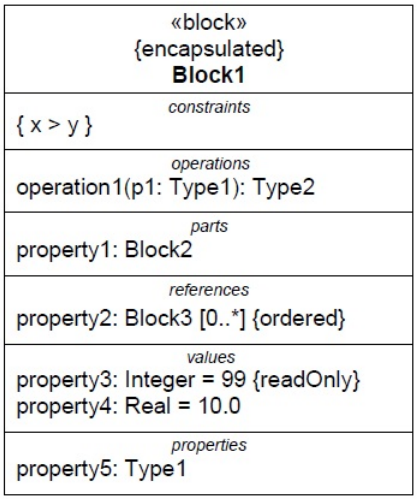

Figura 5.6: Bloco da SysML com compartimentos [OMG-SysML 2010].

Para a representação de aspectos o Bloco da SysML foi estendido como é mostrado na Figura 5.7.

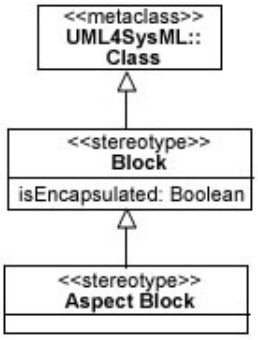

Figura 5.7: Modelo de bloco estendido para representar aspectos.

Aspectos são definidos como blocos, mas inicialmente nenhum atributo é definido, pois os elementos aspectuais serão representados em compartimentos como é mostrado

na Figura 5.8. Para representar os *pointcuts* e consequentemente os *joinpoints* um compartimento chamado *pointcut* é criado. Nesse compartimento, como mostrado na Figura 5.8, é definido o atributo *Joinpoint* do tipo *String*. No atributo *Joinpoint* são definidos os pontos onde o aspecto será inserido.

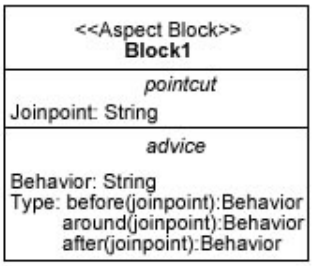

Figura 5.8: Compartimentos no bloco para aspectos.

Para indicar o comportamento do aspecto é criado o compartimento *advice*, como também pode ser observado na Figura 5.8. No compartimento *advice* é definido o atributo *Behavior* do tipo *String* em que é definido o comportamento do aspecto. Como um aspecto pode ter vários comportamentos então estes são numerados. Outro atributo definido no compartimento *advice* é *Type*, que pode assumir três tipos, *before*, *around* ou *after*. É necessário associar a cada um desses tipos o *joinpoint* definido no atributo *pointcut* e também o comportamento definido em *Behavior*.

Para conectar o bloco para aspectos com o bloco simples, o relacionamento $\ll crosscut \gg$ definido no Capítulo 4 é utilizado. Os blocos para aspectos também podem ser representados dentro de um pacote. Para conectar este pacote com um bloco simples o relacionamento $\ll crosscut \gg$ também é utilizado.

5.3.3 Visão de Desenvolvimento

Como mencionado na seção anterior os blocos da SysML podem ser usados em todas as fases de especificação e projeto do sistema. Isso inclui modelar também a decomposição lógica ou física de um sistema e a especificação de software, hardware ou elementos humanos [OMG-SysML 2010]. Por isso, na visão de desenvolvimento também são utilizadas as extensões de blocos que foram definidas na seção anterior e também os diagramas de Pacotes. Os Blocos podem ser usados como definição de camadas de abstração de um modelo de implementação.

5.4 Comparação com Outros Modelos de Arquitetura de Software Orientada a Aspectos

Para avaliar o modelo de arquitetura com aspectos proposto nesse trabalho é apresentada na Tabela 5.1 uma comparação com outros cinco modelos de arquitetura com

aspectos. Esses modelos foram escolhidos baseados na revisão sistemática sendo selecionados os trabalhos mais citados de acordo com o número de citações apresentado na ferramenta de busca Google. Os trabalhos são:

- A - *Taming Dynamically Adaptive Systems using Models and Aspects* [Morin et al. 2009];
- B - *An ADL Dealing with Aspects at Software Architecture Stage* [Navasa et al. 2009];
- C - *Integrating Aspects in Software Architectures: PRISMA Applied to Robotic Tele-Operated Systems* [Perez et al. 2008];
- D - *Malaca: A Component and Aspect-Oriented Agent Architecture* [Amor e Fuentes 2009];
- E - *Ambient-PRISMA: Ambients in Mobile Aspect-Oriented Software Architecture* [Ali et al. 2010b];
- F - Arquitetura de Software Orientada a Aspectos com SysML;

Os seguintes símbolos são utilizados:
■ - significa que o critério avaliado é totalmente satisfeito;
⊡ - significa que o critério avaliado é parcialmente satisfeito;
□ - significa que o critério avaliado não é satisfeito.

No trabalho de [Morin et al. 2009] (A) é apresentada como abordagem de modelagem orientada a aspectos é utilizada em sistemas adaptativos dinamicamente. O trabalho foca na arquitetura de sistemas de execução. A abordagem combina técnicas dirigidas a modelos e orientação a aspectos com o objetivo de limitar o número de artefatos necessários para realizar a variabilidade dinâmica. É utilizada SmartAdapters, uma ferramenta de modelagem orientada a aspectos para compor aspectos no nível de modelo. SmartAdapters automaticamente gera um framework extensível de modelagem orientada a aspectos específico para o metamodelo do trabalho, no entanto não é apresentada qual linguagem de modelagem é utilizada em SmartAdapters. Conforme apresentado no trabalho a representação é realizada em termos de componentes e conectores, assim deduz-se que a visão modelada é somente estrutural. Pelo trabalho não é possível concluir que tipo de abordagem (simétrica ou assimétrica) é utilizada. Há mecanismos de composição entre um aspecto e outro elemento do modelo, no entanto não há rastreabilidade dos aspectos com o nível de requisitos e também não há representação gráfica.

No trabalho apresentado por [Navasa et al. 2009] (B) é proposto utilizar um modelo de arquitetura orientada a aspectos para definir a estrutura de sistemas orientados a aspectos. Depois que a estrutura arquitetural é obtida, ela é formalizada por meio de uma ADL que suporta os conceitos de orientação a aspectos chamada de AspectLEDA.

Critérios de Comparação	A	B	C	D	E	F
Múltiplas Visões	□	■	□	■	□	■
Quais visões	estrutural	estrutural e comportamental	estrutural	estrutural e comportamental	estrutural	Casos de Uso + Requisitos, Desenvolvimento e Estrutural
Rastreabilidade com aspectos no nível de requisitos	□	⊡	□	□	□	■
Abordagem simétrica ou assimétrica	indefinido	assimétrica	simétrica	indefinido	simétrica	assimétrica
Ferramenta (suporte a automatização)	■	■	■	□	■	□
Representação gráfica	□	■	■	■	■	■
Composição de aspectos com a arquitetura	■	■	■	■	■	■
Linguagem de Modelagem	Não definido	UML e AspectLEDA	PRISMA	MaDL (baseada em XML) e UML	AmbientPRISMA	SysML

Tabela 5.1: Comparação entre modelos de arquitetura com aspectos.

AspectLEDA é baseada em uma ADL regular chamada LEDA, que foi estendida com suporte para aspectos. O modelo segue uma abordagem assimétrica, que segundo os autores fornece a vantagem de suportar a evolução por adicionar aspectos de forma natural. O modelo possui representação gráfica utilizando diagramas de Caso de Uso, diagramas de Sequência, e componentes arquiteturais, deduzindo que assim possui visões estruturais e comportamentais. Apesar de ser uma abordagem assimétrica, na representação de casos de uso não fica explícita a separação de aspectos e um caso de uso comum. O modelo possui suporte para gerar uma simulação Java executável da arquitetura. Como o modelo utiliza casos de uso em sua representação é possível manter rastreabilidade com aspectos no nível de requisitos, no entanto isso não é abordado no trabalho.

No trabalho proposto por [Perez et al. 2008] (C) é utilizado PRISMA, um modelo orientado a aspectos de abordagem simétrica aplicado no nível arquitetural. Em PRISMA, a detecção de elementos arquiteturais e aspectos é realizada a partir do documento de requisitos de forma intuitiva. No entanto não possui rastreabilidade de aspectos no nível de requisitos. PRISMA possui uma representação gráfica própria para representar a arquitetura. Depois que os modelos são verificados eles podem gerar automaticamente o código. Os elementos arquiteturais de PRISMA fornecem a visão funcional da arquitetura

de software, não sendo apresentada outras visões do modelo.

No trabalho de [Amor e Fuentes 2009] (D) é apresentada Malaca, uma arquitetura de agentes que combina Engenharia de Software Baseada em Componentes e tecnologias de AOSD para o projeto e desenvolvimento de agentes de software. A arquitetura interna de agentes é especificada usando a linguagem MaDL que é específica de domínio orientado a aspectos baseada em XML. O modelo Malaca define a interface de componentes e aspectos e também mecanismo de composição entre eles, os quais são implementados por uma entidade chamada *Mediator*. Os serviços do *Mediator* são representados no topo do diagrama UML modelado como interfaces, e a estrutura interna do *Mediator* é mostrada na parte de baixo do diagrama. Com as informações apresentadas no trabalho não foi possível definir qual o tipo de abordagem utilizada (simétrica ou assimétrica). No trabalho também não é citado algum tipo de automação. Também não é apresentada uma relação com aspectos no nível de requisitos.

No trabalho de [Ali et al. 2010b] (E) é apresentada uma abordagem chamada de Ambient-PRISMA para projetar modelos de arquitetura de software orientada a aspectos para sistemas móveis. Ambient-PRISMA enriquece a abordagem PRISMA com o conceito de ambiente a partir de *Ambient Calculus*. Um *middleware* chamado Ambient-PRISMANET mapeia o metamodelo para a tecnologia .NET e suporta o ambiente de execução distribuído necessário para executar aplicações móveis. Como Ambient-PRISMA estende o metamodelo PRISMA os critérios avaliados na Tabela 5.4 são iguais.

No modelo proposto nesse trabalho (F) são definidas explicitamente as visões abordadas na arquitetura orientada a aspectos enquanto que nos outros trabalhos avaliados, mesmo quando há mais de uma visão, elas não são explicitamente definidas. Neste trabalho os aspectos no nível de arquitetura são rastreáveis com os aspectos no nível de requisitos. Nos outros trabalhos citados na Tabela 5.4 apenas um deles permite a rastreabilidade mesmo assim de forma limitada. Este trabalho segue uma abordagem assimétrica que permite explicitamente separar os aspectos e outros elementos arquiteturais. A representação gráfica utilizando a SysML e as extensões propostas representa de forma clara os aspectos e sua composição com outros elementos do modelo. Por usar uma linguagem de modelagem padronizada pela OMG e derivada da UML, essa abordagem torna mais simples a aplicação e o aprendizado pelo arquiteto de software.

Capítulo 6

Estudo de Caso

Este capítulo tem como objetivo realizar a aplicação das extensões da SysML para modelar aspectos de um sistema de informação na área de saúde (*Health Watcher*). O capítulo está dividido em 4 seções. Na Seção 6.1 são mostradas as modificações realizadas na ferramenta ArgoUML para representar os modelos de requisitos e a arquitetura de software com aspectos. Na Seção 6.2 é apresentada uma breve descrição do estudo de caso com o objetivo de ajudar o leitor a entender melhor o problema. Na Seção 6.3 é apresentada a modelagem dos requisitos com aspectos, e na Seção 6.4 é apresentada a modelagem da arquitetura com aspectos.

6.1 Extensão da Ferramenta ArgoUML

A ferramenta ArgoUML é um ambiente usado na análise e projeto de sistemas de software orientado a objetos. Nesse sentido, é similar a muitas ferramentas CASE (*Computer Aided Software Engineering*) comerciais que são vendidas como ferramentas para modelagem de sistemas. A ferramenta ArgoUML suporta projeto, desenvolvimento e documentação de aplicações de software orientados a objetos [Ramirez et al. 2011].

A ArgoUML suporta padrões abertos como UML, XMI, SVG, OCL e outros. É uma aplicação 100% Java e isso permite à ferramenta executar em todas as plataformas em que Java está disponível. A ArgoUML possui código aberto, o que permite extensão ou customização da ferramenta, e é distribuída sob a EPL (*Eclipse Public License*) [Ramirez et al. 2011].

Uma visualização da ferramenta ArgoUML com as extensões propostas pode ser observada na Figura 6.1. As setas desenhadas na figura indicam os botões dos novos elementos inseridos na ferramenta. No ambiente de modelagem está a representação de cada elemento. Na sequência os elementos ilustrados são bloco (*«Block»*), bloco com aspectos (*«Aspect Block»*), requisitos (*«Requirement»*) e requisitos com aspectos (*«Aspect Requirement»*). No diagrama de requisitos, mesmo sendo definido como *«Requirement»*, é considerado a utilização do diagrama de requisitos estendido (*«ExtRequirement»*). O elemento

«Block» possui os compartimentos definidos pela SysML (*constraints*, *parts*, *references*, *values*, *properties* e *operations*) e o elemento *«Aspect Block»*, além dos compartimentos definidos pela SysML também possui os compartimentos definidos nesse trabalho (*pointcut* e *advice*).

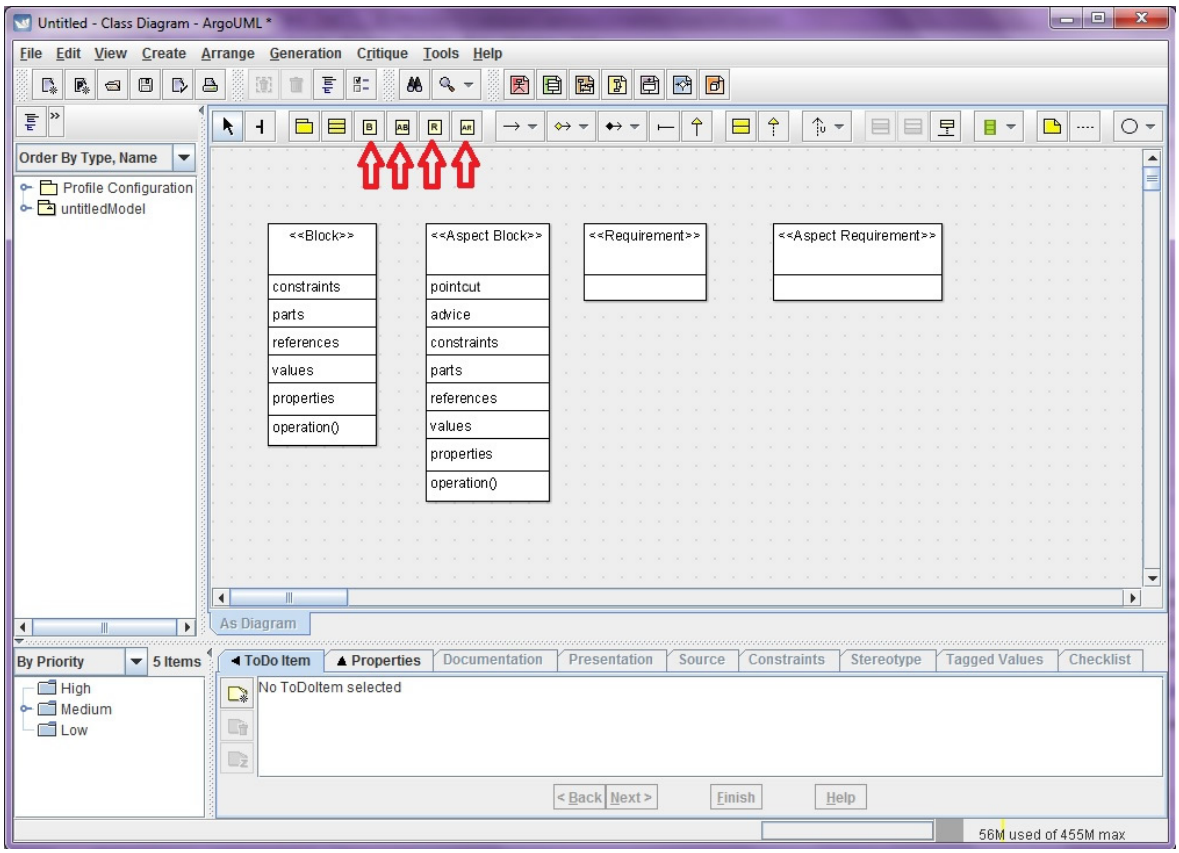

Figura 6.1: Visualização da ferramenta ArgoUML com extensão.

A versão da ArgoUML utilizada foi a 0.35.1. Para realizar as modificações no código fonte da ArgoUML foi utilizada a técnica de clone de código [Baker 1995]. Clone de código é uma atividade comum em desenvolvimento de software. Nessa abordagem são copiados fragmentos de código que em seguida são reutilizados adicionando ou não modificações [Roy e Cordy 2007]. A compreensão do código fonte foi iniciada com o processo de *debug* (depurar) na ferramenta de desenvolvimento de software Eclipse (Versão Juno). *Debug* é conhecido como um processo de localizar e corrigir erros no código de um programa [Kidwell 1998]. A partir do processo de *debug* foi possível localizar algumas classes que implementam o elemento classe da ArgoUML. Dessa forma, todo o código que constrói o elemento classe foi clonado e sendo modificado para a construção dos novos elementos. Na Tabela 6.1 é mostrada uma visão geral do tipo de alterações realizadas no código fonte da ArgoUML.

As classes alteradas juntamente com os métodos alterados/acrescentados são mostradas no diagrama de Classes apresentado na Figura 6.2.

A ArgoUML não suporta a utilização de espaços em textos que são digitados pelo

Módulo	Pacote	Nome da Classe	Descrição das funcionalidades
argouml-app	org.argouml.application.helpers	ResourceLoaderWrapper.java	Faz aparecer o nome dos botões quando o mouse é posicionado em cima de cada botão.
argouml-app	org.argouml.application.i18n	button.properties	Nessa classe são adicionadas as propriedades dos novos botões.
argouml-app	org.argouml.application.images	-	Neste arquivo são armazenadas as imagens que representam os novos botões.
argouml-app	org.argouml.uml.diagram.static_structure.ui	UMLClassDiagram.java	Implementa comandos para a adição dos novos diagramas.
argouml-app	org.argouml.uml.diagram.ui	FigCompartment.java	Permite exibir os nomes dos compartimentos e acrescentar separadores entre eles.
argouml-app	org.argouml.uml.diagram.ui	FigCompartmentBox.java	Permite esconder o compartimento de operações nos diagramas de requisitos.
argouml-app	org.argouml.uml.diagram.use_case.ui	UMLUseCaseDiagram.java	Permite criar pacotes no diagrama de Casos de Uso.
argouml-core-model	org.argouml.model	MetaTypes.java	Contém métodos para recuperar objetos que representam diferentes tipos UML.
argouml-core-model	org.argouml.model	Model.java	Adiciona as strings dos títulos dos novos compartimentos.
argouml-core-model-mdr	org.argouml.model.mdr	CoreFactoryMDRImpl.java	Adição dos métodos para construção dos blocos e requisitos.
argouml-core-model-mdr	org.argouml.model.mdr	MetaTypesMDRImpl	Implementação dos métodos definidos em MetaTypes.
argouml-core-model-mdr	org.argouml.model.mdr	UmlFactoryMDRImpl	Decide qual método chamar de acordo com o tipo de elemento a ser criado.

Tabela 6.1: Visão geral do tipo de modificações realizadas na ArgoUML.

usuário. A ArgoUML também não suporta a utilização de alguns caracteres, como por exemplo ",", ":" e "( )". Desse modo, na modelagem do estudo de caso, o caracter "-" foi utilizado para indicar um espaço ou uma "," dentro de um texto, os caracteres "< >" foram utilizados para indicar "( )" e o caracter "/" foi utilizado para indicar ":".

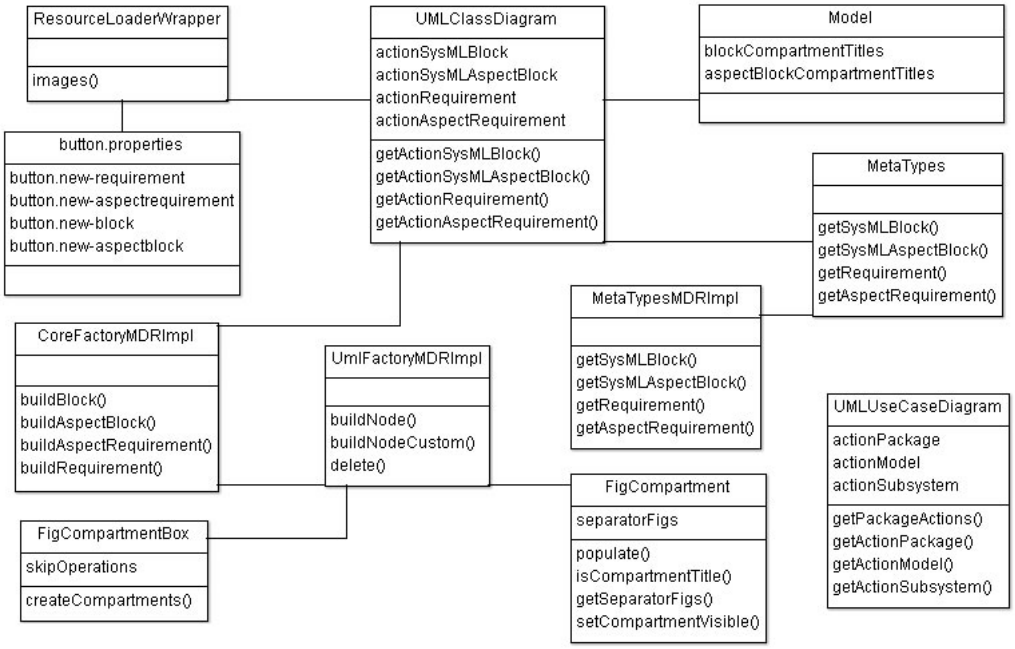

Figura 6.2: Diagrama das classes modificadas.

6.2 Descrição do Estudo de Caso

O sistema *Health Watcher* (HW) é um sistema de informação baseado na Web desenvolvido para melhorar a qualidade dos serviços oferecidos pela Secretaria de Saúde de uma prefeitura [Massoni et al. 2006]. O sistema HW permite que o público registre vários tipos de denúncias relacionadas à saúde. A partir das denúncias, a Secretaria de Saúde pode investigar imediatamente as mesmas e realizar as ações necessárias. O sistema também é usado para notificar as pessoas sobre informações importantes da Secretaria de Saúde que sejam do interesse da população. HW tem sido usado como uma referência para o desenvolvimento de software orientado a aspectos devido à heterogeneidade de interesses transversais encontrados em sua implementação [Chavez et al. 2009].

O sistema HW deve também trocar informações com o sistema *Sanitary Surveillance* (SSVS) (Vigilância Sanitária). Inicialmente essa troca envolve a busca de licenças sanitárias. Subsequentemente, quando o SSVS possuir o módulo de controle de denúncias desenvolvido, as denúncias sobre Vigilância Sanitária serão trocadas entre os dois sistemas.

O documento completo da descrição original dos requisitos pode ser obtido em [Massoni et al. 2006]. Neste documento foi estabelecida prioridade dos requisitos e para isso foram adotadas as denominações "essencial", "importante" e "desejável". "Essencial" é o requisito sem o qual o sistema não entra em funcionamento. Requisitos essenciais são requisitos imprescindíveis que necessitam ser implementados impreterivelmente. "Importante" é o requisito sem o qual o sistema entra em funcionamento, mas de forma não satisfatória. Requisitos importantes devem ser implementados, mas se não forem o sis-

tema poderá ser implantado e usado mesmo assim. "Desejável" é o requisito que não compromete as funcionalidades básicas do sistema, ou seja, o sistema pode funcionar de forma satisfatória sem ele. Requisitos desejáveis são requisitos que podem ser deixados para versões posteriores do sistema, caso não haja tempo hábil para implementá-los na versão que está sendo especificada.

A seguir é descrita a lista de requisitos contida no documento. A versão original em inglês do documento pode ser observada no Apêndice A.

6.2.1 Requisitos Funcionais

O sistema terá os usuários *Citizen* (cidadão) e *Employee* (funcionário).

1. *Employee* - são os servidores da secretaria de saúde e são subdivididos em inspetores, atendentes e gerentes.
2. *Employee* pode realizar as seguintes ações no sistema:

   2.1. login - permite o acesso do funcionário a operações restritas do sistema. Prioridade essencial;

   2.2. cadastrar tabelas - permite o cadastro de tabelas no sistema. Prioridade essencial;

   2.3. atualizar denúncia - permite realizar a atualização do andamento de uma denúncia. Prioridade essencial;

   2.4. cadastrar novos funcionários - permite o cadastramento de novos usuários no sistema. Prioridade essencial;

   2.5. atualizar funcionário - permite a atualização de dados do funcionário no sistema. Prioridade essencial;

   2.6. atualizar unidade de saúde - permite a atualização de dados da unidade de saúde no sistema. Prioridade essencial;

   2.7. trocar funcionário logado - permite que outro funcionário acesse o sistema. Prioridade essencial.
3. *Citizen* - é toda e qualquer pessoa que deseja interagir com o sistema.
4. *Citizen* pode interagir com o sistema para consultar informações e registrar denúncias.
5. Consulta - possui prioridade importante e pode ser:

   5.1. unidades de saúde que atendem uma determinada especialidade;

   5.2. especialidades de uma unidade de saúde em particular;

5.3. informações sobre uma denúncia feita pelo cidadão (detalhes da denúncia, situação (ABERTA, SUSPENSA ou FECHADA), parecer técnico, data do parecer e funcionário que realizou o parecer);

5.4. informações sobre doenças (descrição, sintomas, duração).

6. Denúncia - possui prioridade essencial e pode ser:

6.1. antes de realizar uma denúncia o usuário deve estar logado;

6.2. animal - animais doentes, infestações (por exemplo, roedores, escorpiões ou morcegos), doenças relacionadas a mosquitos (dengue, filariose), animais maltratados;

6.3. comida - casos onde há suspeita de comida estragada;

6.4. especial - casos relacionados a várias razões que não foram mencionados (restaurantes com problemas de higiene, fossas a céu aberto, caminhões de transporte de água de procedimento suspeito, entre outros).

Os três tipos de denúncia tem as seguintes informações em comum:

- dados da denúncia - descrição (obrigatório) e observações (opcional);
- dados do denunciador - nome, rua, complemento, bairro, cidade, estado, CEP, número do telefone e email (todas essas informações são opcionais);
- situação da denúncia (obrigatório) - pode ser ABERTA, SUSPENSA e FECHADA. No registro da denúncia a situação deve ser ABERTA;
- o sistema deve registrar a data que a denúncia foi realizada.

Adicional aos dados anteriores cada tipo de denúncia possui seus dados específicos:

- animal - tipo de animal (obrigatório), quantidade de animais (obrigatório), data que o problema foi observado (obrigatório), dados da localização do problema (rua, complemento, cidade, estado, CEP, e telefone (campos opcionais));
- comida - nome da vítima (obrigatório), dados da vítima (rua, complemento, cidade, estado, CEP, e telefone (campos opcionais)). Quantidade de pessoas que comeram a comida, quantidade de pessoas doentes, quantidade de pessoas internadas e quantidade de pessoas falecidas (todos os campos são obrigatórios). Localização onde os pacientes foram atendidos, refeição suspeita (campos opcionais);
- especial - idade (obrigatório), escolaridade (opcional) e ocupação (opcional). Local mais próximo da ocorrência da denúncia (rua, complemento, cidade, estado, CEP, e número do telefone (campos opcionais)).

6.2.2 Requisitos Não-Funcionais

Usabilidade (prioridade importante)

7. O sistema deve ter uma interface de fácil utilização, visto que o sistema pode ser utilizado por qualquer pessoa que tenha acesso a Internet;

8. O sistema deve ter um HELP on-line para ser consultado por qualquer pessoa que acesse o sistema.

Disponibilidade (prioridade importante)

9. O sistema deve estar disponível 24 horas por dia e 07 dias da semana.

10. Por não ser um sistema crítico, o sistema poderá ficar fora do ar até que seja corrigida alguma falha que possa ocorrer.

Desempenho (prioridade essencial)

11. O sistema deve prover acesso a 20 usuários simultaneamente;

12. O tempo de resposta não deve ultrapassar 05 segundos por acesso.

Segurança (prioridade importante)

13. O sistema deve utilizar algum protocolo de segurança para envio de dados pela Internet;

14. Para ter acesso aos recursos de registro de denúncias, o usuário deve estar habilitado pelo controle de acesso ao sistema.

Padrões (prioridade importante)

15. O sistema deve ser desenvolvido dentro dos padrões estabelecidos pela empresa X, responsável pelas normas e padronização de sistemas da prefeitura.

Distribuição (prioridade importante)

16. O sistema deve ser capaz de ser executado em máquinas diferentes. Por exemplo, o núcleo do sistema pode estar sendo executado em uma máquina e os *servlets* em outra.

Manipulação de erros e exceções (prioridade importante)

17. Várias funcionalidades podem disparar erros enquanto o usuário interage com o sistema e requer diferentes técnicas de manipulação.

17.1 Erros gerais que se aplicam à maioria dos casos são devido a falta de informação e o sistema envia um sinal do erro e quais campos devem ser preenchidos.

17.2 Outros erros podem estar relacionados com entradas inválidas de dados e o mecanismo de manipulação de erros deve tentar evitá-los ou levantar o erro e sugerir a correção.

Interface com o usuário (prioridade essencial)

18. A interface do usuário deve ser implementada utilizando *servlets*.

Meio de armazenamento (prioridade essencial)

19. O sistema deve ser passível de extensão no que se refere ao meio de armazenamento, podendo por exemplo, serem usados *arrays* ou diferentes bancos de dados (MySQL, Oracle e outros).

No documento de requisitos original, manipulação de erros e exceções não é apresentada na lista de requisitos, é apresentada como um fluxo alternativo. No documento de requisitos original não é citado que para o usuário realizar uma denúncia é necessário acessar o sistema por meio de um login e uma senha. No entanto, esse requisito foi considerado neste trabalho.

6.3 Modelagem dos Requisitos com Aspectos

Seguindo o processo descrito na Figura 4.17 do Capítulo 4 primeiramente os requisitos funcionais são representados de acordo com os *viewpoints*.

Os *viewpoints* identificados são: *Employee* e *Citizen*. Os requisitos funcionais do *viewpoint Employee* são apresentados na Figura 6.3 e os requisitos funcionais do *viewpoint Citizen* são apresentados na Figura 6.4.

Estes modelos foram representados utilizando a ferramenta ArgoUML estendida. Por questões de espaço, no modelo de requisitos não são mostrados todos os atributos que foram definidos no capítulo 4.

No segundo passo do processo de modelagem de requisitos com aspectos, os requisitos não-funcionais são representados. As extensões da ferramenta ArgoUML novamente foram utilizadas para representar os requisitos não-funcionais.

No sistema *Health Watcher* os requisitos não-funcionais são:

- *Usability* (Figura 6.5);

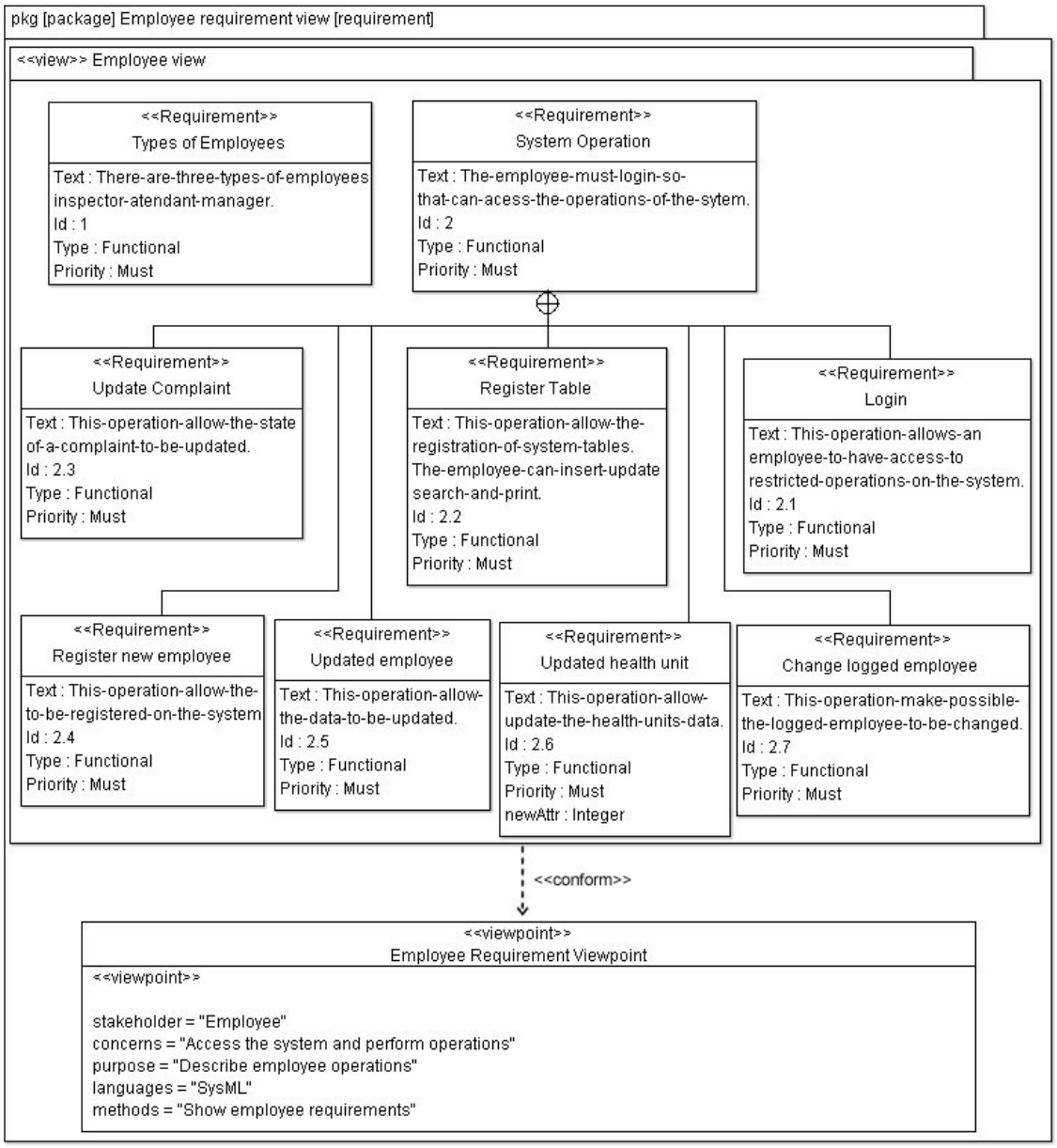

Figura 6.3: *Viewpoint Employee.*

- *Availability* (Figura 6.6);
- *Performance* (Figura 6.7);
- *Security* (Figura 6.8);
- *Standards* (Figura 6.9);
- *Distribution* (Figura 6.10);
- *User Interface* (Figura 6.11);
- *Storage Medium* (Figura 6.12);
- *Error and Exception Handling* (Figura 6.13).

No terceiro passo os requisitos não-funcionais são relacionados com os *viewpoints*. Esse relacionamento é realizado por meio das tabelas estendidas da SysML conforme

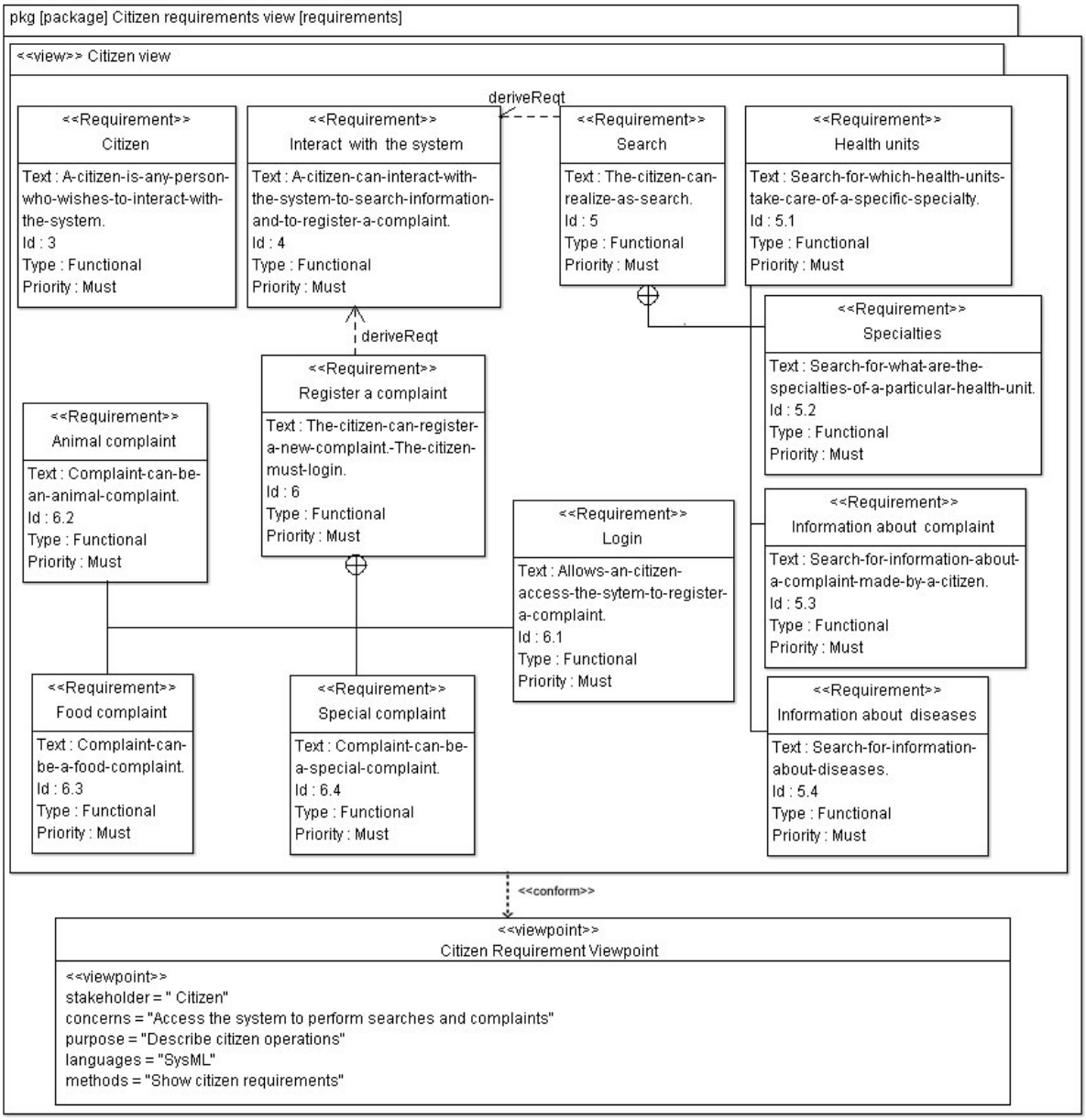

Figura 6.4: *Viewpoint Citizen.*

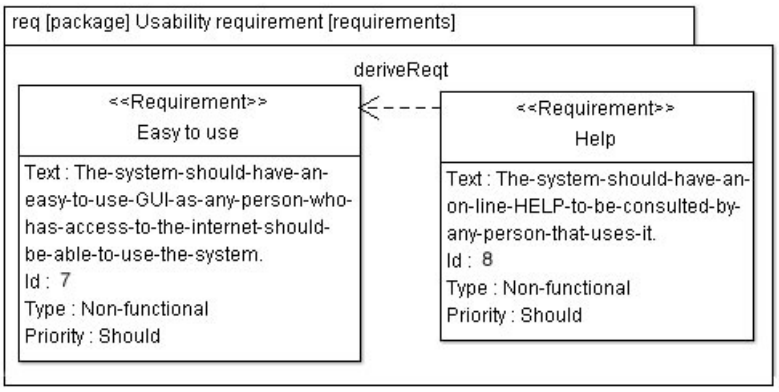

Figura 6.5: Requisito não-funcional *usability.*

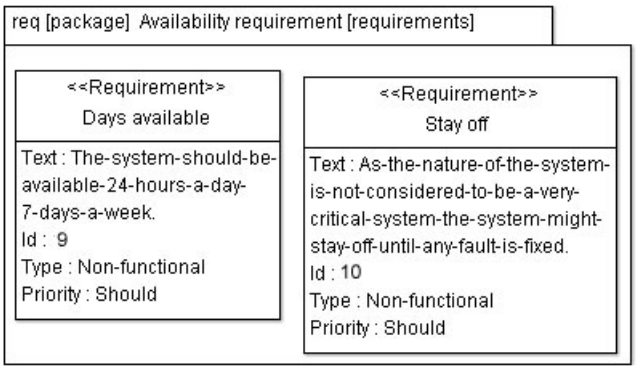

Figura 6.6: Requisito não-funcional *availability*.

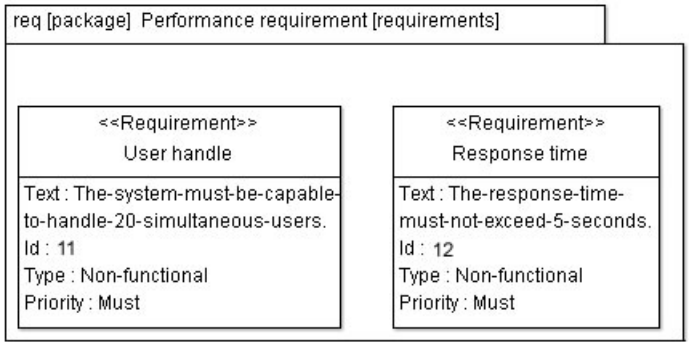

Figura 6.7: Requisito não-funcional *performance*.

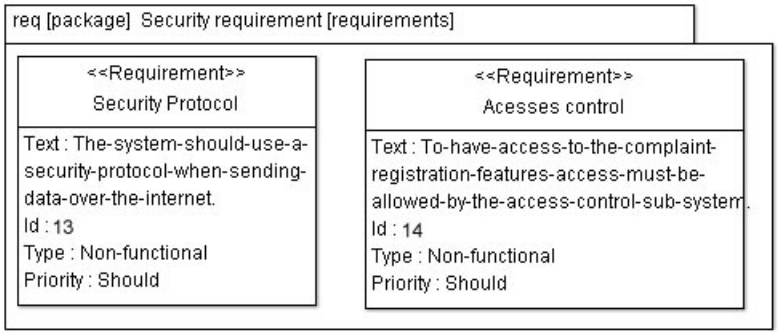

Figura 6.8: Requisito não-funcional *security*.

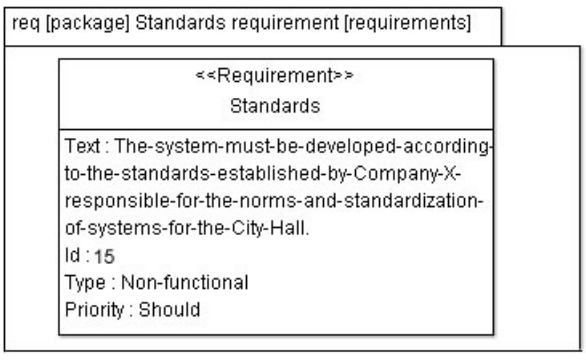

Figura 6.9: Requisito não-funcional *standards*.

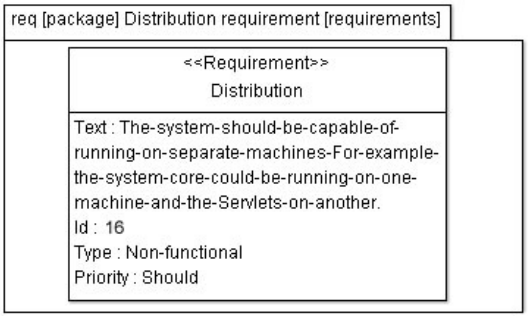

Figura 6.10: Requisito não-funcional *distribution*.

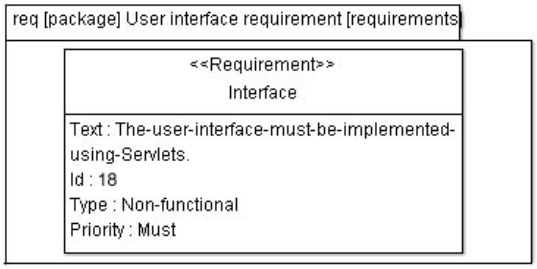

Figura 6.11: Requisito não-funcional *user interface*.

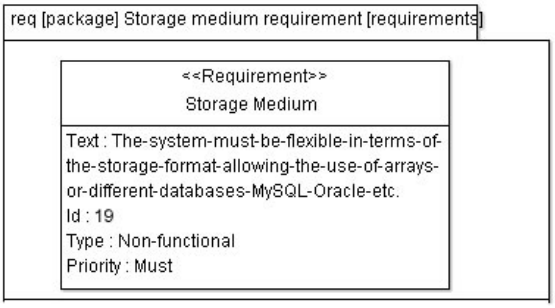

Figura 6.12: Requisito não-funcional *storage medium*.

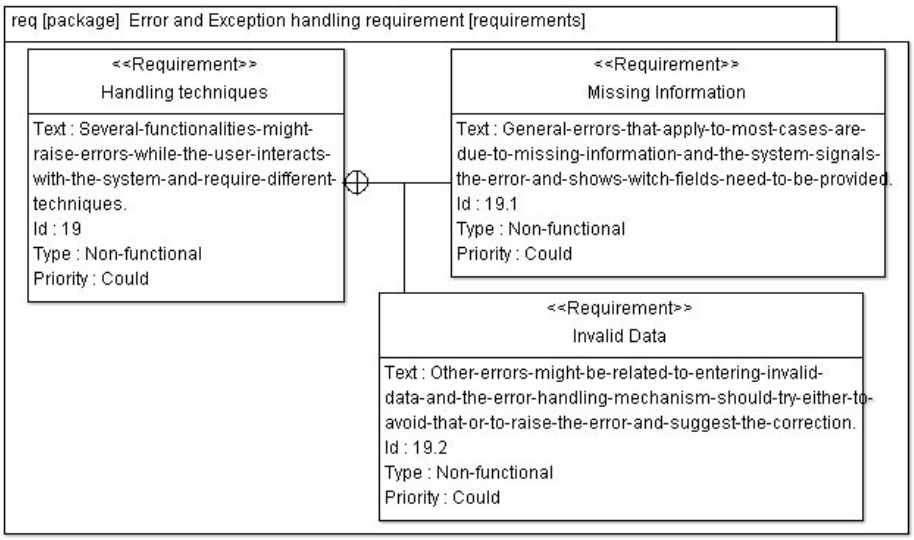

Figura 6.13: Requisito não-funcional *error and exception handling*.

apresentadas no capítulo 4. A ferramenta ArgoUML não permite a criação de tabelas, nesse caso utiliza-se qualquer editor de texto que permite a criação de tabelas.

Na Tabela 6.2 é realizado o relacionamento do requisito não-funcional *Usability* com os *viewpoints* do estudo de caso.

id1	*non-functional requirement*	*relation*	*id2*	*functional requirement*	*viewpoint*
7	Easy to use	-	2	System Operation	Employee
7	Easy to use	-	4	Interact with the system	Citizen
7	Easy to use	-	5	Search	Citizen
7	Easy to use	-	6	Register a complaint	Citizen
8	Help	-	2	System Operation	Employee
8	Help	-	4	Interact with the system	Citizen
8	Help	-	5	Search	Citizen
8	Help	-	6	Register a complaint	Citizen

Tabela 6.2: Tabela de relacionamento do requisito *usability* com os *viewpoints*.

Na Tabela 6.3 é realizado o relacionamento do requisito não-funcional *Availability* com os *viewpoint* do estudo de caso.

id1	*non-functional requirement*	*relation*	*id2*	*functional requirement*	*viewpoint*
9	Days available	-	2	System Operation	Employee
9	Days available	-	4	Interact with the system	Citizen
9	Days available	-	5	Search	Citizen
9	Days available	-	6	Register a complaint	Citizen
10	Stay off	-	2	System Operation	Employee
10	Stay off	-	4	Interact with the system	Citizen
10	Stay off	-	5	Search	Citizen
10	Stay off	-	6	Register a complaint	Citizen

Tabela 6.3: Tabela de relacionamento do requisito *availability* com os *viewpoints*.

Na Tabela 6.4 é realizado o relacionamento do requisito não-funcional *performance* com os *viewpoint* do estudo de caso.

id1	*non-functional requirement*	*relation*	*id2*	*functional requirement*	*viewpoint*
11	User handle	-	2	System Operation	Employee
11	User handle	-	4	Interact with the system	Citizen
11	User handle	-	5	Search	Citizen
11	User handle	-	6	Register a complaint	Citizen
12	Response time	-	2	System Operation	Employee
12	Response time	-	4	Interact with the system	Citizen
12	Response time	-	5	Search	Citizen
12	Response time	-	6	Register a complaint	Citizen

Tabela 6.4: Tabela de relacionamento do requisito *performance* com os *viewpoints*.

Na Tabela 6.5 é realizado o relacionamento do requisito não-funcional *Security* com os *viewpoint* do estudo de caso.

Na Tabela 6.6 é realizado o relacionamento do requisito não-funcional *standards* com os *viewpoint* do estudo de caso.

id1	*non-functional requirement*	*relation*	*id2*	*functional requirement*	*viewpoint*
13	Security protocol	-	2	System Operation	Employee
13	Security protocol	-	4	Interact with the system	Citizen
13	Security protocol	-	5	Search	Citizen
13	Security protocol	-	6	Register a complaint	Citizen
14	Access control	-	2	System Operation	Employee
14	Access control	-	6	Register a complaint	Citizen

Tabela 6.5: Tabela de relacionamento do requisito *security* com os *viewpoints*.

id1	*non-functional requirement*	*relation*	*id2*	*functional requirement*	*viewpoint*
15	System developed	-	1	Types of Employee	Employee
15	System developed	-	2	System Operations	Employee
15	System developed	-	3	Citizen	Citizen
15	System developed	-	4	Interact with the system	Citizen
15	System developed	-	5	Search	Citizen
15	System developed	-	6	Register a complaint	Citizen
15	System developed	-	7	Types of complaints	Complaint
15	System developed	-	8	Complaints data	Complaint
15	System developed	-	9	Complaint addressed	Complaint

Tabela 6.6: Tabela de relacionamento do requisito *standards* com os *viewpoints*.

Na Tabela 6.7 é realizado o relacionamento do requisito não-funcional *distribution* com os *viewpoint* do estudo de caso.

id1	*non-functional requirement*	*relation*	*id2*	*functional requirement*	*viewpoint*
16	Distribution	-	2	System Operation	Employee
16	Distribution	-	4	Interact with the system	Citizen
16	Distribution	-	5	Search	Citizen
16	Distribution	-	6	Register a complaint	Citizen

Tabela 6.7: Tabela de relacionamento do requisito *distribution* com os *viewpoints*.

Na Tabela 6.8 é realizado o relacionamento do requisito *error and exception handling* com os *viewpoint* do estudo de caso.

id1	*non-functional requirement*	*relation*	*id2*	*functional requirement*	*viewpoint*
17	Error and Exception Handling	-	2	System Operation	Employee
17	Error and Exception Handling	-	4	Interact with the system	Citizen
17	Error and Exception Handling	-	5	Search	Citizen
17	Error and Exception Handling	-	6	Register a complaint	Citizen

Tabela 6.8: Tabela de relacionamento do requisito *error and exception handling* com os *viewpoints*.

No quarto passo são verificados quais requisitos funcionais se repetem nos diferentes *viewpoints*. Nesse caso apenas o requisito *Login* foi repetido em diferentes *viewpoints* como é mostrado na Tabela 6.9.

No quinto passo são identificados os candidatos a aspectos. Na Tabela 6.10 é mostrado o relacionamento de candidatos a aspectos com os *viewpoints*.

id1	*functional requirement1*	*viewpoint1*	*repeat*	*id2*	*functional requirement2*	*viewpoint2*
2.1	Login	Employee	-	6.1	Login	Citizen

Tabela 6.9: Relacionamento de requisitos funcionais com requisitos funcionais.

id1	*aspect*	*relation*	*id2*	*functional requirement*	*viewpoint*
7	Easy to use	before	2	System Operation	Employee
7	Easy to use	before	4	Interact with the system	Citizen
7	Easy to use	before	5	Search	Citizen
7	Easy to use	before	6	Register a complaint	Citizen
8	Help	around	2	System Operation	Employee
8	Help	around	4	Interact with the system	Citizen
8	Help	around	5	Search	Citizen
8	Help	around	6	Register a complaint	Citizen
9	Days available	around	2	System Operation	Employee
9	Days available	around	4	Interact with the system	Citizen
9	Days available	around	5	Search	Citizen
9	Days available	around	6	Register a complaint	Citizen
10	Stay off	around	2	System Operation	Employee
10	Stay off	around	4	Interact with the system	Citizen
10	Stay off	around	5	Search	Citizen
10	Stay off	around	6	Register a complaint	Citizen
11	User handle	around	2	System Operation	Employee
11	User handle	around	4	Interact with the system	Citizen
11	User handle	around	5	Search	Citizen
11	User handle	around	6	Register a complaint	Citizen
12	Response time	around	2	System Operation	Employee
12	Response time	around	4	Interact with the system	Citizen
12	Response time	around	5	Search	Citizen
12	Response time	around	6	Register a complaint	Citizen
13	Security protocol	around	2	System Operation	Employee
13	Security protocol	around	4	Interact with the system	Citizen
13	Security protocol	around	5	Search	Citizen
13	Security protocol	around	6	Register a complaint	Citizen
14	Access control	before	2	System Operation	Employee
14	Access control	before	6	Register a complaint	Citizen
15	System developed	before	1	Types of Employee	Employee
15	System developed	before	2	System Operations	Employee
15	System developed	before	3	Citizen	Citizen
15	System developed	before	4	Interact with the system	Citizen
15	System developed	before	5	Search	Citizen
15	System developed	before	6	Register a complaint	Citizen
16	Distribution	around	2	System Operation	Employee
16	Distribution	around	4	Interact with the system	Citizen
16	Distribution	around	5	Search	Citizen
16	Distribution	around	6	Register a complaint	Citizen
17	Error and Exception Handling	around	2	System Operation	Employee
17	Error and Exception Handling	around	4	Interact with the system	Citizen
17	Error and Exception Handling	around	5	Search	Citizen
17	Error and Exception Handling	around	6	Register a complaint	Citizen
2.1	Login	before	2	System Operations	Employee
2.1	Login	before	6	Register a Complaint	Citizen

Tabela 6.10: Relacionamento de candidatos a aspectos com os *viewpoints*.

No sexto passo é verificado se há conflito de interesses entre os aspectos. Nesse estudo

de caso, há uma possibilidade de conflito entre *Security Protocol* e *Response Time* devido à implementação de segurança poder comprometer o tempo de resposta. Na Tabela 6.11 são mostrados os possíveis conflitos no estudo de caso.

id1	*aspect1*	*conflict*	*id2*	*aspect2*	*viewpoint*
13	Security Protocol	-	12	Response Time	Employee

Tabela 6.11: Conflitos entre requisitos aspectuais.

No sétimo passo os candidatos a aspectos são compostos com os *viewpoints* no modelo gráfico. Na Figura 6.14 é mostrada a composição com o *viewpoint Employee*.

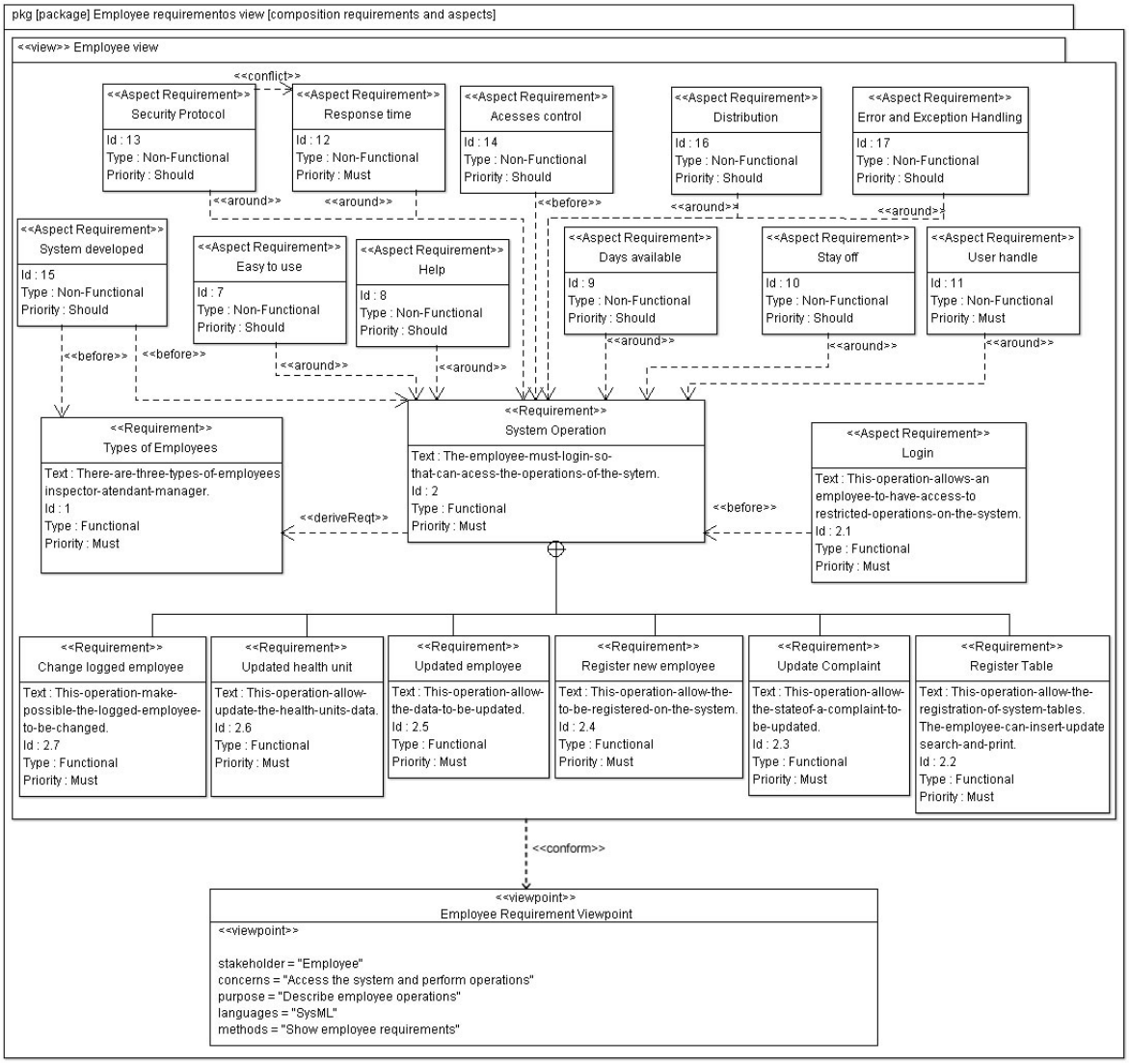

Figura 6.14: Modelo de composição do *viewpoint employee* com aspectos.

6.4 Modelagem da Arquitetura Orientada a Aspectos

Conforme definido no Capítulo 5, a arquitetura orientada a aspectos é representada nas visões estrutural, de desenvolvimento e cenários + requisitos.

A visão de cenários é obtida pelos casos de uso. Na Figura 6.15 são mostrados os casos de uso para *Employee* e na Figura 6.16 são mostrados os casos de uso para *Citizen*.

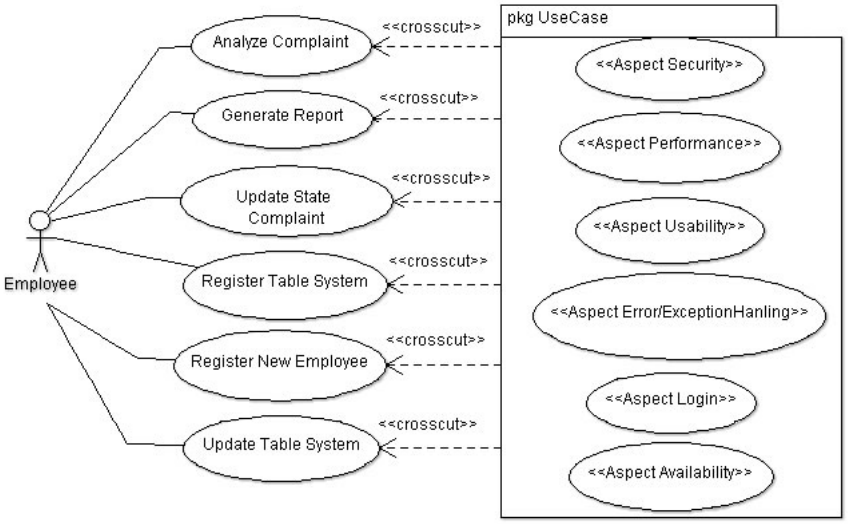

Figura 6.15: Casos de uso *Employee*.

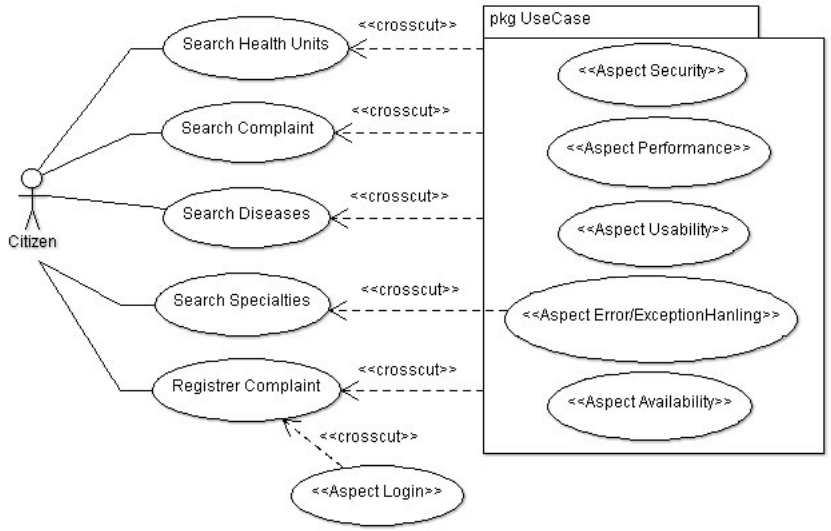

Figura 6.16: Casos de uso *Citizen*.

Na Figura 6.17 é apresentada a visão estrutural do sistema HW.

Para evitar que o diagrama fique com vários relacionamentos, o que dificulta a visualização, todos os aspectos exceto o aspecto *Login* foram representados dentro de um pacote.

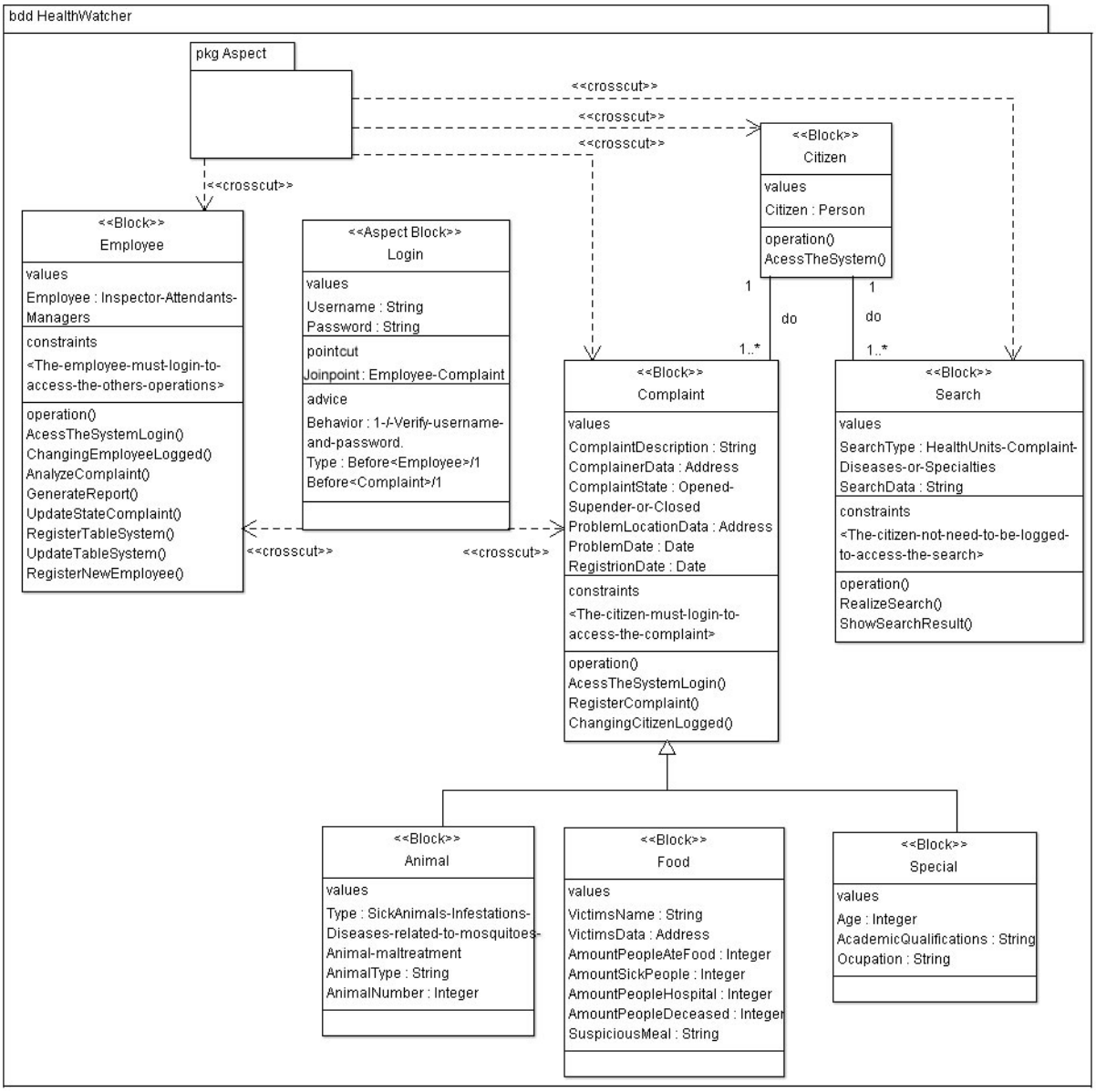

Figura 6.17: Visão estrutural do sistema *Health Watcher*.

O aspecto *Login* se relaciona apenas com o bloco *Employee* e *Complaint*, por isso não foi colocado dentro do pacote. O pacote contendo os aspectos é mostrado na Figura 6.18.

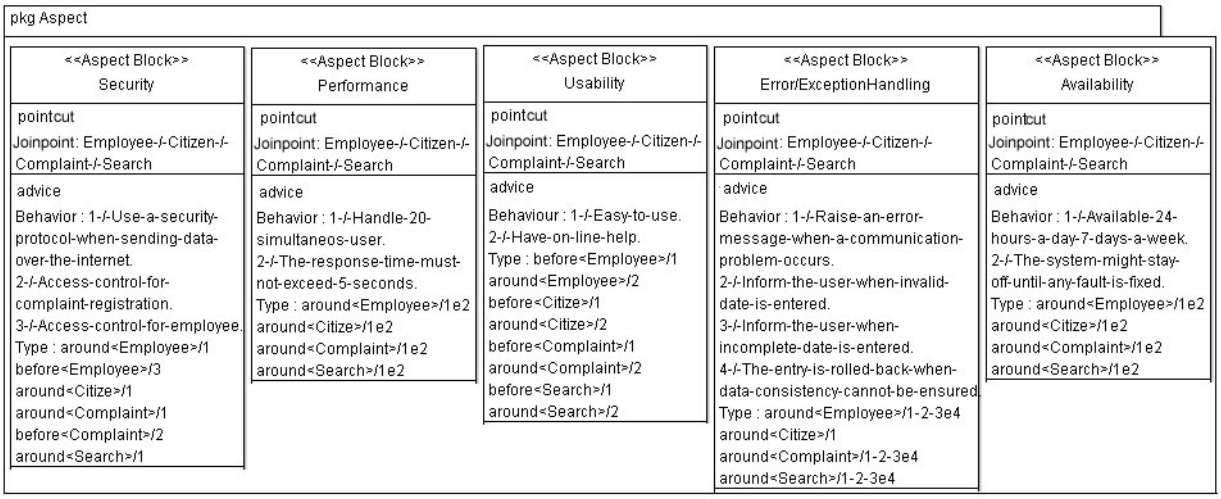

Figura 6.18: Pacote de aspectos.

6.5 Estudo de Caso Health Watcher Executado em Outros Modelos

O estudo de caso Health Watcher foi conduzido nos modelos apresentados por [Silva 2006] e [Sampaio et al. 2007] que também utilizaram o documento de requisitos apresentado em [Massoni et al. 2006].

Em [Silva 2006] é apresentado o modelo AOV-grap. AOV-graph consiste no modelo V-graph [Yu et al. 2004] estendido. Esta extensão consiste em adicionar o relacionamento transversal ao V-graph. O V-graph é um gráfico com forma geral da letra V. No topo dois vértices de V representam respectivamente requisitos funcionais e não-funcionais em termos de modelos de metas (*goal*). Requisitos não-funcionais são representados em termos de *softgoals*, isto é, metas sem claros critérios. Ambos modelos são árvores AND/OR com ligações de correlação lateral. Na Figura 6.19 é representado o estudo de caso Health Watcher com o modelo AOV-grap.

Como pode ser observado na figura o modelo gráfico de AOV-graph é extenso. Com o acréscimo de novos requisitos este modelo tende a aumentar graficamente, o que torna a visualização difícil. No modelo apresentado neste trabalho a representação gráfica também pode ficar extensa, no entanto a utilização de pacotes ajuda a diminuir elementos gráficos.

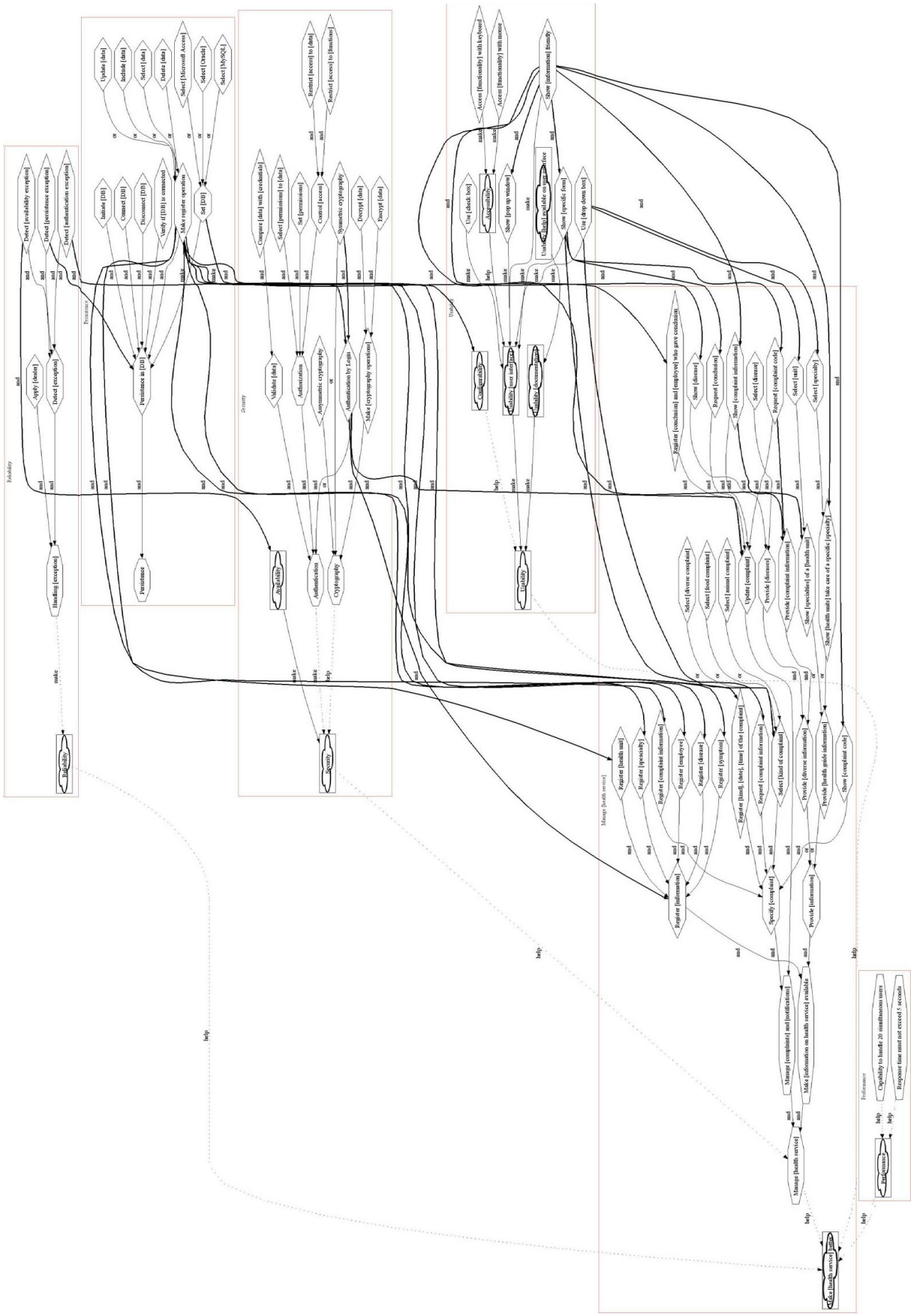

Figura 6.19: Health Watcher com o modelo AOV-grap [Silva 2006].

No modelo apresentado por [Silva 2006] também é realizada a especificação em XML. A seguir uma pequena parte da especificação em XML é mostrada.

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE aspect_oriented_model (View Source for full doctype...)>
- <aspect_oriented_model>
 - <goal_model id="GM1" name="Manage [health service]">
    <softgoal id="G1" name="Make [health service] better"
     decomposition_label="and" />
  - <goal id="G2" name="Manage [health service]" decomposition_label
    = "and">
   - <goal id="G3" name="Manage [complaints] and [notifications]"
     decomposition_label="and">
    - <task id="T3.1" name="Specify [complaint]"
       decomposition_label="and">
      - <task id="T3.1.1" name="Select [kind of complaint]"
         decomposition_label="and">
         <task id="T3.1.1.1" name="Select [diverse complaint]"
          decomposition_label="or" />
         <task id="T3.1.1.2" name="Select [food complaint]"
          decomposition_label="or" />
         <task id="T3.1.1.3" name="Select [animal complaint]"
          decomposition_label="or" />
       </task>
       ...

       ...
- <crosscutting_relationship>
  - <source>
     <task_ref id="T15.1.1" decomposition_label="and" />
    </source>
  - <pointcut id="P15.1" name="make register operation">
    - <pointcut_expression>
      - <operand primitive="include">
         <regular_expression text="Register.*" param="name"
          type="task" />
        </operand>
        <and />
      - <pointcut_expression>
```

```
        <not />
       - <operand primitive="include">
          <task_ref id="T4.1" decomposition_label="and" />
         </operand>
        </pointcut_expression>
       </pointcut_expression>
      </pointcut>
    - <pointcut id="P15.2" name="set DB">
     - <pointcut_expression>
      - <operand primitive="include">
          <softgoal_ref id="S3.2" decomposition_label="and" />
       </operand>
      </pointcut_expression>
     </pointcut>
....
```

Em [Sampaio et al. 2007] é apresentada a ferramenta EA-Miner que utiliza técnicas baseadas em processamento de linguagem natural para automatizar a tarefa de AORE. A ferramenta é utilizada para identificar potenciais interesses transversais em documentos de requisitos. Em [Sampaio 2007] é apresentada a análise do sistema Health Watcher usando a AORE baseada em viewpoint e a ferramenta EA-Miner. A documentação do sistema Health Watcher foi utilizada como entrada na ferramenta EA-Miner. EA-Miner então produz várias visões mostrando os *viewpoints* identificados, aspectos iniciais (de natureza funcional ou não-funcional) e o relacionamento (isto é, as influências transversais entre aspectos iniciais e *viewpoints*). O engenheiro de requisitos utiliza as informações produzidas por EA-Miner e seu conhecimento e experiência para produzir a especificação de AORE baseada em *viewpoints*.

O resultado final do estudo de caso é a representação em linguagem natural dos *viewpoints*, aspectos iniciais, relacionamentos e *trade-offs*. As informações obtidas também são estruturadas na linguagem XML. Na sequência são mostradas partes do resultado do estudo de caso obtido depois da aplicação da ferramenta EA-Miner.

Viewpoint: Employee

Requirements:

1. There are 3 types of employees: inspectors, attendants, managers

2. The employee must login so that he/she can access the various operations of the system, which are:

2.1. Login: This operation allows an employee to have access to restricted operations on the Health-Watcher system.

2.2. Register tables: This operation allows the registration of system tables. The

following operations are possible: insert, update, delete, search and print. The available tables include:

...

Early Aspect: Availability

Requirements:

1. The system should be available 24 hours a day, 7 days a week.

2. As the nature of the system is not considered to be a very critical system, the system might stay off until any fault is fixed.

Relacionamento transversal

Na Tabela 6.12 são mostrados os aspectos que se relacionam com um determinado *viewpoint* e quais requisitos os aspectos afetam. Esse tipo de informação é fornecida pela ferramenta EA-Miner.

Early Aspect	*Viewpoints Crosscut (requirements)*
Availability	*Employee (requirement 2 and all sub-requirements); Citizen (requirements 2 to 4 and all subrequirements)*
Security	*Employee (requirement 2 and all sub-requirements); Citizen (requirements 2 to 4 and all subrequirements); SSVS system (all requirements)*
Performance	*Employee (requirement 2 and all sub-requirements); Citizen (requirements 2 to 4 and all subrequirements)*
Concurrency	*Employee (requirement 2 and all sub-requirements); Citizen (requirements 2 to 4 and all subrequirements)*
Persistence	*Employee (all requirements) Citizen (all requirements) Complaint (all requirements)*
Distribution	*Employee (requirement 2 and all sub-requirements); Citizen (requirements 2 to 4 and all subrequirements)*
Error and Exception Handling	*Employee (requirement 2 and all sub-requirements); Citizen (requirements 2 to 4 and all subrequirements)*
Compatibility	*SSVS system (all requirements)*
Usability	*Employee (requirement 2 and all sub-requirements); Citizen (requirements 2 to 4 and all subrequirements)*
Standards *Operational environment*	*All viewpoints (except SSVS System) and all requirements* *All viewpoints (except SSVS System) and all requirements*

Tabela 6.12: *Viewpoints Crosscut by each Early Aspect* [Sampaio 2007].

Na Tabela 6.13 é apresentada uma discussão resumida dos *trade-offs* para cada *viewpoint*.

Viewpoint	*Trade-off points*
Employee	*Most early aspects (availability, security, performance, concurrency, persistence, distribution, error and exception handling, usability, standards, operational environment) apply to the employee viewpoint. Performance generally contributes negatively to security and positively to availability. Distribution can contribute positively to performance as different servers can be used to load the balance (application and web server). Concurrency can contribute negatively to persistence as concurrent users can corrupt the data, therefore requiring a good transaction management approach. Error handling contributes positively to usability as users can know when errors happen and what actions to take to correct them.*
Citizen	*The same early aspects and explanation for Employee applies.*
SSVS System	*(Security and Compatibility) early aspects apply to all requirements of SSVS system. The health watcher system interacts with the SSVS system and the interaction must respect the same protocol and be secure at the same time. Ensuring security might contribute negatively for compatibility of the security mechanism selected overcomplicates the protocol for establishing the data exchange between the two systems.*
Complaint	*(Persistence, standards and operation environment) apply to this viewpoint. There is no restriction from the operational environment to use a specific database and with respect to standards of the company need to be checked to see if any constraints apply. Therefore, apparently no trade-offs need to be considered here.*

Tabela 6.13: *Trade-off points* [Sampaio 2007].

O modelo gerado por EA-Miner não oferece visualização gráfica. O resultado final são especificações em linguagem natural indicando separadamente os aspectos. Também não há nenhuma rastreabilidade dos aspectos no nível arquitetural.

Na Tabela 6.14 é realizada a comparação dos estudos de casos apresentados em [Silva 2006] e [Sampaio et al. 2007] utilizando os mesmos critérios apresentados no Capítulo 4 seção 4.5.

Os seguintes símbolos são utilizados:

■ - significa que o critério avaliado é totalmente satisfeito;

⊡ - significa que o critério avaliado é parcialmente satisfeito;

□ - significa que o critério avaliado não é satisfeito.

Critérios de Comparação	[Silva 2006]	[Sampaio et al. 2007]	Estudo de Caso realizado nesse trabalho
Processo para identificar requisitos transversais	■	■	■
Identificação de requisitos transversais funcionais e não-funcionais	⊡	⊡	■
Composição de Aspectos com Requisitos	■	■	■
Identificação de Conflitos	□	⊡	⊡
Resolução de Conflitos	□	□	□
Modelagem Gráfica	■	□	■
Diagrama de Requisitos Específico	□	□	■
Relacionamento entre requisitos	□	□	■
Relação com a UML	□	□	■
Extensibilidade do Modelo	■	■	■
Relacionamento com a Arquitetura	□	□	■
Rastreabilidade com o Projeto	□	□	□
Suporte a Automatização	□	⊡	□

Tabela 6.14: Comparação entre estudos de caso.

No modelo AOV-graph também não há uma distinção explícita da representação de aspectos, o que pode tornar difícil a análise do modelo por pessoas que não dominam AOV-graph. O modelo também não oferece algum tipo de rastreabilidade dos aspectos para o nível arquitetural.

Capítulo 7

Conclusão

Neste capítulo é apresentado um resumo dos modelos propostos nesse trabalho bem como suas contribuições, limitações e possíveis trabalhos futuros. Na Seção 7.1 é apresentado o resumo do trabalho. Na Seção 7.2 são apresentadas as contribuições do trabalho e na Seção 7.3 são apresentadas suas limitações. Na Seção 7.4 são apresentados possíveis trabalhos futuros.

7.1 Resumo do Trabalho

A separação de interesses tem sido identificada como um dos princípios fundamentais da Engenharia de Software e o Desenvolvimento Orientado a Aspectos utiliza-se do princípio da separação de interesses em todos os estágios do ciclo de vida do software aumentando sua interdependência. A separação de interesses fornece melhores resultados e oferece vantagens como redução da complexidade e melhor modularidade, no reuso e na manutenção dos artefatos de software. Inicialmente pesquisas na área de desenvolvimento orientado a aspectos focaram principalmente no nível de implementação, porém o conceito tem sido aplicado nos estágios iniciais do desenvolvimento de software.

Os modelos para representação de aspectos nos estágios iniciais do ciclo de desenvolvimento de software apresentados na literatura abordam diferentes domínios de aplicação assim como diferentes linguagens de modelagem. Em muitos trabalhos não é realizado algum tipo de mapeamento entre modelos de requisito orientado a aspectos e arquitetura orientada a aspectos e o arquiteto de software não tem nenhuma ajuda para identificar interesses transversais no nível arquitetural a partir das especificações de requisitos.

A falta do estabelecimento de uma relação da Engenharia de Requisitos com a Arquitetura de Software pode dificultar a representação dos aspectos iniciais e consequentemente a identificação dos aspectos nas outras fases do desenvolvimento de software. Diante da importância em considerar aspectos nos estágios iniciais do desenvolvimento de software é que este trabalho teve como objetivo geral estabelecer uma relação da Engenharia de Requisitos com a Arquitetura de Software considerando a representação com aspectos.

Este trabalho aborda os problemas de entrelaçamento e espalhamento de características transversais durante a modelagem de requisitos e arquitetura de software. Para isso, foi desenvolvido um modelo para representar aspectos no nível de requisitos e um modelo para representar aspectos no nível arquitetural.

A SysML, por permitir uma modelagem de propósito geral para aplicação em engenharia de sistemas, suportando a especificação, análise, projeto, verificação e validação de um amplo espectro de sistemas e sistemas de sistemas foi utilizada como linguagem de modelagem para representar os modelos desenvolvidos através da criação de estereótipos.

Para validar o trabalho foi realizado um estudo de caso. Os requisitos definidos do sistema Health Watcher em linguagem natural foram aplicados nos modelos de requisitos e de arquitetura com aspectos propostos nesse trabalho. Posteriormente, foi realizada uma comparação com outros cinco modelos de requisitos com aspectos e cinco modelos de arquitetura com aspectos. Essa comparação foi baseada em um conjunto de critérios importantes na definição de requisitos e arquitetura.

Em comparação com as outras abordagens analisadas pode-se observar que o modelo proposto nesse trabalho cobre importantes características que os outros modelos não cobrem. O modelo de requisitos possui um processo para identificar requisitos transversais tanto de origem funcional como não-funcional, possui um modelo de composição entre requisitos e requisitos aspectuais, identifica conflitos, e ainda possui rastreabilidade com a arquitetura. O modelo de arquitetura apresenta múltiplas visões, possui rastreabilidade com os requisitos, possui uma abordagem assimétrica que facilita a separação de um aspecto com outro elemento do modelo, possui um modelo de composição entre aspectos e outro elemento e possui também representação gráfica.

7.2 Contribuições

Após a execução deste trabalho, algumas contribuições foram realizadas para a comunidade científica. As principais foram:

- Uma revisão sistemática na literatura em relação a arquitetura de software orientada a aspectos. Com a revisão sistemática foi possível avaliar os benefícios de introduzir aspectos em arquitetura de software. Foi possível verificar quais são os interesses mais comuns quando é projetada uma arquitetura de software utilizando aspectos e também que a introdução de aspectos é útil na manutenção de software. A revisão sistemática é apresentada no Capítulo 3.
- Um modelo que permite representar aspectos no nível de requisitos. O modelo define as atividades de identificação de requisitos aspectuais, separação e composição de requisitos e requisitos aspectuais, e identificação e resolução de conflitos entre aspectos. O modelo é apresentado no Capítulo 4.

- Uma arquitetura com múltiplas visões que permite representar aspectos no nível arquitetural. As visões utilizadas foram estrutural, casos de uso + requisitos e desenvolvimento. Em todas as visões é utilizado o conceito de aspectos. O modelo é apresentado no Capítulo 5. Os modelos de requisitos e arquitetura com aspectos não são realizados de forma independente sendo possível manter a rastreabilidade de aspectos no nível de requisitos com aspectos no nível arquitetural.
- A extensão da linguagem de modelagem SysML para representar os modelos desenvolvidos. Ainda não havia trabalhos publicados que apresentassem a modelagem de aspectos no nível de requisitos e arquitetura com a SysML. Uma importante característica da SysML é que a linguagem possui um diagrama específico para requisitos. Outra importante característica é que os requisitos da SysML podem aparecer em outros diagramas da linguagem para mostrar seu relacionamento com o projeto. Isso facilita representar a relação de aspectos no nível de requisitos com aspectos no nível de arquitetura, sendo que essa relação não é apresentada na maioria dos trabalhos publicados na área de aspectos iniciais. As tabelas da SysML também foram utilizadas. O formato tabular facilita o rastreamento dos requisitos durante o ciclo de vida do sistema. A rastreabilidade ajuda na identificação da origem, destino e ligações entre requisitos e aspectos. As extensões da SysML são apresentadas nos Capítulos 4 e 5.
- A extensão da ferramenta ArgoUML com os novos elementos da SysML. A ferramenta ArgoUML é um ambiente usado na análise e projeto de sistemas de software orientado a objetos. Para representar os modelos de requisitos e a arquitetura de software com aspectos utilizando a linguagem de modelagem SysML a ferramenta ArgoUML foi estendida. Uma visão geral das extensões realizadas na ferramenta ArgoUML é apresentada no Capítulo 6.

A pesquisa também teve como resultado a publicação de artigos em congressos científicos internacionais:

- O artigo *"A Systematic Review on Aspects in Software Architecture Design"* foi aceito na *XXXI International Conference of the Chilean Computer Science Society* (Qualis B3), e apresentado em Novembro de 2012 em Valparaíso-Chile.
- O artigo *"Extensions of SysML for Modeling an Aspect Oriented Software Architecture with Multiple Views"* foi aceito na *10th International Conference on Information Technology : New Generations* (Qualis B1), e será apresentado em Abril de 2013 em Las Vegas-US.

O artigo *"Modeling Aspects in Requirements using SysML Extensions"* foi submetido para a *15th International Conference on Enterprise Information Systems - ICEIS 2013* (Qualis B1).

7.3 Limitações do Trabalho

O estudo de caso do sistema Health Watcher é um bom exemplo por possuir vários interesses transversais. No entanto, pode ser melhor representado se outras funcionalidades forem adicionadas com o objetivo de fornecer flexibilidade para explorar interesses transversais a partir de requisitos funcionais. Nesse trabalho o modelo foi baseado somente nas funcionalidades apresentadas na especificação de requisitos. Assim, foram encontrados interesses transversais tradicionais, como segurança, disponibilidade e usabilidade. Também não foi possível obter informações suficientes para modelar a visão de desenvolvimento.

Mais avaliações e discussões da aplicabilidade dos modelos são necessárias. Foi realizada uma comparação com outros modelos de requisitos e arquitetura com aspectos, no entanto é necessário um número maior de aplicações (estudos de caso) e comparações com outros modelos assim como mais critérios de comparação.

Tanto no modelo de requisitos quanto no modelo de arquitetura todo o processo é manual, não há nenhum suporte a automatização como acontece em alguns outros modelos da literatura.

A extensão da ferramenta ArgoUML não consegue representar todos os elementos da SysML, como por exemplo *viewpoint* que é utilizado neste trabalho. Assim é necessário utilizar uma ferramenta de edição gráfica para acrescentar esses elementos. A ferramenta ArgoUML também não permite representar as tabelas da SysML utilizadas no modelo de requisitos deste trabalho. É necessário utilizar alguma outra ferramenta para a criação das tabelas.

7.4 Trabalhos Futuros

Dentre outras pesquisas correlatas derivadas desta, pode-se citar:

- Realizar a modelagem de outros estudos de caso em outros domínios de aplicação para tentar comprovar as conclusões alcançadas neste trabalho. Realizar a modelagem também usando outras abordagens tradicionais de Engenharia de Requisitos e Arquitetura de Software e comparar com a abordagem orientada a aspectos apresentada nesse trabalho. Além disso, realizar a avaliação por meio da técnica TAM. A técnica TAM [Davis 1985] foi proposta para abordar por que usuários aceitam ou rejeitam uma tecnologia da informação. O propósito principal da técnica TAM é fornecer uma base para traçar o impacto de variáveis externas em intenções, atitudes e opiniões internas. Nesta técnica a facilidade de uso percebida e a utilidade percebida são os dois fatores mais importantes em explicar o uso da tecnologia [Legris et al. 2003]. A técnica TAM tem provado ser um modelo teórico útil para ajudar a entender e explicar o comportamento de uso em implementações de sistemas de informação.

- Para a representação de aspectos em outros diagramas da SysML, como trabalho futuro, pode-se estender outros diagramas da SysML para permitir a representação com aspectos. Por exemplo, diagrama de Atividade, diagrama de Sequência e diagrama Paramétrico.
- Para permitir uma modelagem completa com os elementos da SysML, como trabalho futuro, pode-se inserir outros elementos da SysML na ferramenta ArgoUML.

Apêndice A

Requirements Document Health-Watcher

A.1 System overview

The purpose of the system is to collect then manage public health related complaints and notifications. The system is also used to notify people about important information regarding the Health System.

The Health Watcher system must also exchange information with the SSVS system (Sanitary Surveillance System). Initially, this exchange will involve the querying of sanitary licenses. Subsequently, when the SSVS has the Complaint Control module deployed, Sanitary Surveillance complaints will be exchanged between the two systems.

A.1.1 Broadening and related systems

With the deployment of HEALTH-WATCHER system, the Public Health System will considerably improve:

- The complaint control (registering and notifications).
- Quality of service regarding the dissemination of information; for example: vaccination campaigns, disease prevention, health guides, obtaining birth/death certificates and application details for a sanitary license.

The system will be managed by DIEVS and will exchange information with the Sanitary Surveillance system.

A citizen can access the system through the internet or dialing 1520, and make their complaint or ask information about the health services. In the event of a complaint being made, it will be registered on the system and addressed by a specific department. This department will be able to handle the complaint in an appropriate manner and return

a response when complaint has been dealt with. This response will be registered on the system and available to be queried.

The system will be for public use in kiosks at several strategic points, on which the citizen will be able to register complaints and request information.

A.1.2 Users description

The HEALTH-WATCHER system will support the following users:

- Attendants/DIEVS staff

  Health System employee, placed on DIEVS.

- Citizen

  Any person who wishes to interact with the system.

A.2 Functional requirements (use cases)

The following actors have been identified on the system:

Citizen
Any person who wishes to interact with the system.

Employee
Health System employees (inspectors, attendants and managers).

A.2.1 Use cases associated to a citizen

Use cases pertaining to Citizen are the ones that follow:

- FR01 - Query information
- FR02 - Specify complain

[FR01] Query information

This use case allows a citizen to perform queries.

Query Health Guide
The citizen might query:

- Which health units take care of a specific specialty.
- What are the specialties of a particular health unit.

Query Specialty Information

The citizen might query:

- Information about a complaint made by a citizen:
  - Complaint details.
  - Situation (OPENED, SUSPENDED, or CLOSED).
  - Technical analysis.
  - Analysis date.
  - Employee that made the analysis.
- Information about diseases:
  - Description.
  - Symptoms.
  - Duration.

Priority: Important

Inputs and pre-conditions:

- The data to be queried must be registered on the system

Outputs and post-conditions:

- The query result to the citizen

[FR02] Complaint specification

This use case allows a citizen to register complaints. Complaints can be:

Animal Complaint - DVA

- Sick animals.
- Infestations (rodents, scorpions, bats, etc.)
- Diseases related to mosquitoes (dengue, filariose).
- Animal maltreatment.

Food Complaint - DVISA

- Cases where there is a suspicion infected food being eaten.

Special Complaint - DVISA

- Cases related to several reasons, which are not mentioned above (restaurants with hygiene problems, leaking sewerage, suspicious water transporting trucks, etc.).

The three kinds of complaints have the following information in common:

- Complaint data: description (mandatory) and observations (optional);
- Complainer data: name, street, complement, district, city, state/province, zip code, telephone number and e-mail. All these fields are optional.
- Complaint state (mandatory), which might be: OPENED, SUSPENDED or CLOSED. When a complaint is first registers its state must be OPENED.
- The system must register the complaint registration date.

In addition to the above data, each complaint type has its own specific data, including:

Animal Complaint - DVA

- Type of animal (mandatory), amount of animals (mandatory), date problem was observed (mandatory).
- Problem location data: street, complement, district, city, state/province, zip code and telephone number. All of these fields are optional.

Food Complaint - DVISA

- Victim's name (mandatory).
- Victim's data: street, complement, district, city (or closest one), state/province, zip code and telephone number. All of these fields are optional.
- Amount of people who ate the food, amount of sick people, amount of people who were sent to a hospital and amount of deceased people. All mandatory.
- Location where the patients were treated, suspicious meal. All optional.

Special Complaint - DVISA

- Age (mandatory), academic qualifications (optional), occupation (optional).
- Street, complement, district, city, state/province, zip code and telephone number of the closest location to the complaint location. All optional.

Priority: Essential

Inputs and pre-conditions:

- None

Outputs and post-conditions:

- The complaint saved on the system

A.2.2 Use Cases Related to Employee

The employee must login so that he/she can access the various operations of the system, which are:

- FR10 - Login.
- FR11 - Register tables.
- FR12 - Update complaint.
- FR13 - Register new employee.
- FR14 - Update employee.
- FR15 - Update health unit.
- FR16 - Change logged employee.

[FR10] Login

This use case allows an employee to have access to restricted operations on the Health-Watcher system.

Priority: Essential

Inputs and pre-conditions:

- None

Outputs and post-conditions:

- Password validated by the system

[FR11] Register tables

This use case allows the registration of system tables. The following operations are possible: insert, update, delete, search and print.

The available tables include:

- Health unit (unit code, unit description).
- Specialty (code and description).
- Health unit / Specialty (health unit and specialty).
- Employee (login, name and password).
- Type of disease (code, name, description, symptom and duration).
- Symptom (code and description).
- Type of disease / Symptom (type of disease and symptom).

Priority: Essential

Inputs and pre-conditions:

- Verified employee

Outputs and post-conditions:

- Updated data on the tables

[FR12] Update complaint

This use case allows the state of a complaint to be updated.

Priority: Essential

Inputs and pre-conditions:

- The complaint must be registered and have the OPENED state.
- Verified employee.

Outputs and post-conditions:

- Complaint updated and with state CLOSED.

[FR13] Register new employee

This use case allows new employees to be registered on the system.

Priority: Essential

Inputs and pre-conditions:

- Verified employee

Outputs and post-conditions:

- New employee registered on the system

[FR14] Update employee

This use case allows of the employee's data to be updated on the system.

Priority: Essential

Inputs and pre-condition:

- Verified employee

Outputs and post-conditions:

- Employee's data updated on the system

[FR15] Update health unit

This use case allows the health unit's data to be updated.

Priority: Essential

Inputs and pre-conditions:

- Verified employee

Outputs and post-conditions:

- Health unit's data updated on the system.

[FR16] Change logged employee

This use case allows the currently logged employee to be changed.

Priority: Essential

Inputs and pre-conditions:

- Verified employee.

Outputs and post-conditions:

- First employee signed out and new employee logged-in.

A.3 Non-functional requirements

Usability

The system should have an easy to use GUI, as any person who has access to the internet should be able to use the system. The system should have an on-line HELP to be consulted by any person that uses it.

Priority: Important

Availability

The system should be available 24 hours a day, 7 days a week. The nature of the system not being a critical system, the system might stay off until any fault is fixed.

Priority: Important

Performance
The system must be capable to handle 20 simultaneous users. The response time must not exceed 5 seconds.

Priority: Essential

Security The system should use a security protocol when sending data over the internet. To have access to the complaint registration features, access must be allowed by the access control sub-system.

Priority: Important

Standards
The system must be developed according to the standards established by X1, responsible for the norms and standardization of systems for the City Hall.

Priority: Important

Hardware and software
This section lists the hardware and software to be used for the system to operate in a desirable fashion.

SOFTWARE

- One license for the Microsoft Windows for the workstation

HARDWARE

- One computer with: Pentium III processor, 256 MB of RAM memory, net card 3Com 10/100. This equipment shall be used by the attendant as a workstation.

Distribution
The system should be capable of running on separate machines. For example, the system core could be running on one machine and the Servlets on another.

Priority: Important

User interface

The user interface must be implemented using Servlets.

Priority: Essential

Storage medium

The system must be flexible in terms of the storage format allowing the use of arrays or different databases (MySQL, Oracle, etc.)

Priority: Essential

Referências Bibliográficas

[Ali et al. 2010a] Ali, M. S., Ali Babar, M., Chen, L., e Stol, K.-J. (2010a). A Systematic Review of Comparative Evidence of Aspect-Oriented Programming. *Information and Software Technology*, 52:871–887.

[Ali et al. 2010b] Ali, N., Ramos, I., e Solis, C. (2010b). Ambient-PRISMA: Ambients in Mobile Aspect-Oriented Software Architecture. *Journal of Systems and Software*, 83:937–958.

[Amor e Fuentes 2009] Amor, M. e Fuentes, L. (2009). Malaca: A Component and Aspect-Oriented Agent Architecture. *Information and Software Technology*, 51:1052–1065.

[Araújo e Baniassad 2007] Araújo, J. e Baniassad, E. (2007). Transactions on Aspect-Oriented Software Development III. chapter Guest Editors' Introduction: Early Aspects-Analysis, Visualization, Conflicts and Composition, pp. 1–3. Springer-Verlag.

[Araújo et al. 2005] Araújo, J., Baniassad, E., Clements, P., Moreira, A., Rashid, A., e Tekinerdogan, B. (2005). Early Aspects: The Current Landscape. Technical notes, CMU/SEI and Lancaster University.

[Araújo et al. 2002] Araújo, J., Moreira, A., Brito, I., e Rashid, A. (2002). Aspect-Oriented Requirements with UML. In *Workshop: Aspect-oriented Modeling with UML, 1st International Conference on Aspect-Oriented Software Development.*

[AspectJ 2012] AspectJ (2012). http://www.eclipse.org/aspectj/doc/released/progguide/semantics-advice.html. Acessado em 12/06/2012.

[Avgeriou et al. 2011] Avgeriou, P., Grundy, J., Hall, J., Lago, P., e Mistrík, I. (2011). *Relating Software Requirements and Architectures.* Springer, Dordrecht-Netherlands.

[Baker 1995] Baker, B. S. (1995). On Finding Duplication and Near-Duplication in Large Software Systems. In *Proceedings of the Second Working Conference on Reverse Engineering*, WCRE '95, pp. 86–95. IEEE Computer Society.

[Baniassad e Clarke 2004] Baniassad, E. e Clarke, S. (2004). Theme: An Approach for Aspect-Oriented Analysis and Design. In *Proceedings of the 26th International Conference on Software Engineering*, ICSE '04, pp. 158–167. IEEE Computer Society.

[Baniassad et al. 2006] Baniassad, E., Clements, P. C., Araujo, J., Moreira, A., Rashid, A., e Tekinerdogan, B. (2006). Discovering Early Aspects. *IEEE Software*, 23:61–70.

[Bartsch e Harrison 2008] Bartsch, M. e Harrison, R. (2008). An Exploratory Study of the Effect of Aspect-Oriented Programming on Maintainability. *Software Quality Control*, 16:23–44.

[Bass et al. 1998] Bass, L., Clements, P., e Kazman, R. (1998). *Software Architecture in Practice*. Addison Wesley, Massachusetts.

[Batista et al. 2006] Batista, T., Chavez, C., Garcia, A., Rashid, A., Sant'Anna, C., Kulesza, U., e Filho, F. C. (2006). Reflections on Architectural Connection: Seven Issues on Aspects and ADLs. In *Proceedings of the 2006 International Workshop on Early aspects at ICSE*, EA '06, pp. 3–10. ACM.

[Bennouar et al. 2010] Bennouar, D., Khammaci, T., e Henni, A. (2010). A New Approach for Components Port Modeling in Software Architecture. *Journal of System and Software*, 83(8):1430–1442.

[Berry 2004] Berry, D. M. (2004). The Inevitable Pain of Software Development: Why There is no Silver Bullet. In *Radical Innovations of Software and System Engineering in the Future*, pp. 50–74.

[Booch 1994] Booch, G. (1994). *Object-Oriented Analysis and Design with Applications*. Benjamin-Cummings Publishing Company, Inc., Redwood City, CA, USA, 2nd edition.

[Bosch 2004] Bosch, J. (2004). Software Architecture: The Next Step. *Software Architecture, First European Workshop (EWSA)*, 3047:194–199.

[Boucke et al. 2008] Boucke, N., Weyns, D., Hilliard, R., Holvoet, T., e Helleboogh1, A. (2008). Characterizing Relations Between Architectural Views. In *ECSA '08 Proceedings of the 2nd European conference on Software Architecture*, pp. 66–81. Springer.

[Brito et al. 2007] Brito, I. S., Vieira, F., Moreira, A., e Ribeiro, R. A. (2007). Transactions on Aspect-Oriented Software Development III. chapter Handling Conflicts in Aspectual Requirements Compositions, pp. 144–166. Springer-Verlag, Berlin, Heidelberg.

[Brito 2008] Brito, I. S. S. (2008). *Aspect-Oriented Requirements Analysis*. PhD thesis, Faculdade de Ciências e Tecnologia - Universidade Nova de Lisboa, Quinta da Torre, 2829-516 Caparica, Portugal.

[Brooks 1987] Brooks, Jr., F. P. (1987). No Silver Bullet Essence and Accidents of Software Engineering. *Computer*, 20(4):10–19.

[Chavez et al. 2009] Chavez, C., Garcia, A., Batista, T., Oliveira, M., Sant'Anna, C., e Rashid, A. (2009). Composing Architectural Aspects Based on Style Semantics. In *AOSD '09 Proceedings of the 8th ACM International Conference on Aspect-Oriented Software Development*, pp. 111–122. ACM.

[Chitchyan et al. 2007a] Chitchyan, R., Pinto, M., Rashid, A., e Fuentes, L. (2007a). Transactions on Aspect-Oriented Software Development IV. chapter COMPASS: Composition-Centric Mapping of Aspectual Requirements to Architecture, pp. 3–53. Springer-Verlag, Berlin, Heidelberg.

[Chitchyan et al. 2007b] Chitchyan, R., Rashid, A., Rayson, P., e Waters, R. (2007b). Semantics-Based Composition for Aspect Oriented Requirements Engineering. In *Proceedings of the 6th International Conference on Aspect-Oriented Software Development*, AOSD '07, pp. 36–48. ACM.

[Chung e Prado Leite 2009] Chung, L. e Prado Leite, J. C. (2009). Conceptual Modeling: Foundations and Applications. chapter On Non-Functional Requirements in Software Engineering, pp. 363–379. Springer-Verlag, Berlin, Heidelberg.

[Clements 1996] Clements, P. C. (1996). A Survey of Architecture Description Languages. In *Proceedings of the 8th International Workshop on Software Specification and Design*, IWSSD '96, pp. 16–25. IEEE Computer Society.

[Conejero et al. 2012] Conejero, J. M., Figueiredo, E., Garcia, A., Hernández, J., e Jurado, E. (2012). On the Relationship of Concern Metrics and Requirements Maintainability. *Information and Software Technology*, 54:212–238.

[Conejero et al. 2010] Conejero, J. M., Hernándeza, J., Jurado, E., e Berg, K. V. D. (2010). Mining Early Aspects Based on Syntactical and Dependency Analyses. *Science of Computer Programming*, 75:1113–1141.

[Costa-Soria et al. 2009] Costa-Soria, C., Perez, J., e Carsi, J. A. (2009). Handling the Dynamic Reconfiguration of Software Architectures Using Aspects. In *CSMR '09 Proceedings of the 2009 European Conference on Software Maintenance and Reengineering*, pp. 263–266. IEEE Computer Society.

[Dai e Cooper 2006] Dai, L. e Cooper, K. (2006). Modeling and Performance Analysis for Security Aspects. *Journal Science of Computer Programming*, 61:58–71.

[Damian et al. 2004] Damian, D., Zowghi, D. L., Vaidyanathasamy, L., e Pal, Y. (2004). An Industrial Case Study of Immediate Benefits of Requirements Engineering Process Improvement at the Australian Center for Unisys Software. *Empirical Software Engineering*, 9:45–75.

[Davis 1985] Davis, F. (1985). *A technology Acceptance Model for Empirically Testing new End-User Information Systems: Theory and Results*. PhD thesis, MIT Sloan School of Management, Cambridge, MA.

[Dijkstra 1997] Dijkstra, E. W. (1997). *A Discipline of Programming*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 1st edition.

[Farooquia et al. 1995] Farooquia, K., Logrippoa, L., e Meer, J. (1995). The ISO Reference Model for Open Distributed Processing: An Introduction. *Computer Networks and ISDN Systems*, 27:1215–1229.

[Filman et al. 2004] Filman, R. E., Elrad, T., Clarke, S., e Aksit, M. (2004). *Aspect-Oriented Software Development*. Addison Wesley Professional, Boston, MA, USA.

[Fleissner e Baniassad 2007] Fleissner, S. e Baniassad, E. (2007). Epi-Aspects: Aspect-Oriented Conscientious Software. In *OOPSLA 2007: Proceedings of the 22nd ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications*, pp. 659–674. ACM.

[Freeman 2010] Freeman, R. E. (2010). *Strategic Management: A Stakeholder Approach*. Cambridge University Press, New York, USA.

[Friedenthal et al. 2012] Friedenthal, S., Moore, A., e Steiner, R. (2012). *A Pratical Guide to SysML: The Systems Modeling Language*. Morgan Kaufmann, Burlington, Massachusetts, USA, 2nd edition.

[Fuentes et al. 2008] Fuentes, L., Pinto, M., e Sanchez, P. (2008). Generating CAM Aspect-Oriented Architectures Using Model-Driven Development. *Information and Software Technology*, 50:1248–1265.

[Garcia et al. 2009] Garcia, A. F., Figueiredo, E. M. L., Sant'Anna, C. N., Pinto, M., e Fuentes, L. (2009). Representing Architectural Aspects with a Symmetric Approach. In *Proceedings of the 15th Workshop on Early Aspects*, EA '09, pp. 25–30. ACM.

[Garlan et al. 2010] Garlan, D., Bachmann, F., Ivers, J., Stafford, J., Bass, L., Clements, P., e Merson, P. (2010). *Documenting Software Architectures: Views and Beyond*. Addison-Wesley Professional, Boston, MA, USA, 2nd edition.

[Greer 2005] Greer, D. (2005). *Requirements Engineering for Sociotechnical Systems*, chapter Requirements Prioritisation for Incremental and Iterative Development, pp. 100–118. London, UK.

[Grundy 1999] Grundy, J. C. (1999). Aspect-Oriented Requirements Engineering for Component-Based Software Systems. In *RE '99 Proceedings of the 4th IEEE International Symposium on Requirements Engineering*, pp. 84–91. IEEE Computer Society.

[Hilliard 2000] Hilliard, R. (2000). *IEEE-Std-1471-2000 Recommended Practice for Architectural Description of Software-Intensive Systems*.

[Ian e Stevens 2002] Ian, F. A. e Stevens, R. (2002). *Writing Better Requirements*. Addison-Wesley, Boston, MA, USA, 1st edition.

[Jacobson 1992] Jacobson, I. (1992). *Object-Oriented Software Engineering: A Use Case Driven Approach*. Addison-Wesley Professional, Boston, MA, USA.

[Juristo et al. 2002] Juristo, N., Moreno, A. M., e Silva, A. (2002). Is the European Industry Moving Toward Solving Requirements Engineering Problems? *IEEE Software*, 19:70–77.

[Kiczales et al. 1997] Kiczales, G., Lamping, J., Mendhekar, A., Maeda, C., Lopes, C. V., Loingtier, J., e Irwin, J. (1997). Aspect-Oriented Programming. In *Proceedings of the 11th European Conference on Object-Oriented Programming (ECOOP'97)*, pp. 220–242. Springer-Verlag.

[Kidwell 1998] Kidwell, P. A. (1998). Stalking the Elusive Computer Bug. *IEEE Annals History Computing*, 20(4):5–9.

[Kienzle et al. 2009] Kienzle, J., Wisam, A. A., e Klein, J. (2009). Aspect-Oriented Multi-View Modeling. In *AOSD '09 Proceedings of the 8th ACM International Conference on Aspect-Oriented Software Development*, pp. 87–98. ACM.

[Kitchenham e Charters 2007] Kitchenham, B. e Charters, S. (2007). Guidelines for Performing Systematic Literature Reviews in Software Engineering. Technical report, School of Computer Science and Mathematics - Keele University and Department of Computer Science - University of Durham.

[Kitchenham et al. 2009] Kitchenham, B., Pearl Brereton, O., Budgen, D., Turner, M., Bailey, J., e Linkman, S. (2009). Systematic Literature Reviews in Software Engineering - A Systematic Literature Review. *Information and Software Technology*, 51(1):7–15.

[Kruchten 1995] Kruchten, P. (1995). The 4+1 View Model of Software Architecture. *IEEE Software*, 12(6):42–50.

[Kuhn et al. 2009] Kuhn, E., Mordinyi, R., Keszthelyi, L., Schreiber, C., Bessler, S., e Tomic, S. (2009). Aspect-Oriented Space Containers for Efficient Publish/Subscribe Scenarios in Intelligent Transportation Systems. In *Software Architecture 2009 European Conference on Software Architecture WICSA/ECSA 2009 Joint Working IEEEIFIP Conference on (2009)*, pp. 432–448. Springer.

[Legris et al. 2003] Legris, P., Ingham, J., e Collerette, P. (2003). Why do People use Information Technology? A Critical Review of the Technology Acceptance Model. *Information & Management*, 40:191–204.

[Leist e Zellner 2006] Leist, S. e Zellner, G. (2006). Evaluation of Current Architecture Frameworks. In *Proceedings of the 2006 ACM Symposium on Applied Computing*, SAC '06, pp. 1546–1553. ACM.

[Lobato et al. 2008] Lobato, C., Garcia, A., Romanovsky, A., e Lucena, C. (2008). An Aspect-Oriented Software Architecture for Code Mobility. *Software Practice and Experience*, 38:1365–1392.

[Lopes 1997] Lopes, C. V. (1997). *A Language Framework for Distributed Programming.* PhD thesis, College of Computer Science, Northeastern University, Boston, USA.

[Madeyski e Szala 2007] Madeyski, L. e Szala, L. (2007). Impact of Aspect-Oriented Programming on Software Development Efficiency and Design Quality: An Empirical Study. *IET Software*, 1(5):180–187.

[Malek et al. 2010] Malek, S., Krishnan, H. R., e Srinivasan, J. (2010). Enhancing Middleware Support for Architecture-Based Development Through Compositional Weaving of Styles. *Journal of Systems and Software*, 83(12):2513–2527.

[Massoni et al. 2006] Massoni, T., Soares, S., e Borba, P. (2006). Requirements Document Health-Watcher. Technical report. version 2.0.

[Medvidovic e Taylor 2000] Medvidovic, N. e Taylor, R. N. (2000). A Classification and Comparison Framework for Software Architecture Description Languages. *IEEE Transactions Software Engineering*, 26(1):70–93.

[Miles 2005] Miles, R. (2005). *AspectJ Cookbook Real World Aspect Oriented Programming with Java.* OReilly Media, Inc, USA.

[Moreira et al. 2002] Moreira, A., Araújo, J. a., e Brito, I. (2002). Crosscutting Quality Attributes for Requirements Engineering. In *Proceedings of the 14th International Conference on Software Engineering and Knowledge Engineering*, SEKE '02, pp. 167–174. ACM.

[Morin et al. 2009] Morin, B., Barais, O., Nain, G., e Jezequel, J.-M. (2009). Taming Dynamically Adaptive Systems Using Models and Aspects. In *Proceedings of the 31st International Conference on Software Engineering*, ICSE '09, pp. 122–132. IEEE Computer Society.

[Nakagama et al. 2011] Nakagama, E. Y., Ferrari, F. C., Sasaki, M. M. F., e Maldonado, J. C. (2011). An Aspect-Oriented Reference Architecture for Software Engineering Environments. *Journal of Systems and Software*, 84:1670–1684.

[Navarro et al. 2007] Navarro, E., Letelier, P., e Ramos, I. (2007). Requirements and Scenarios: Running Aspect-Oriented Software Architectures. In *WICSA '07 Proceedings of the Sixth Working IEEE/IFIP Conference on Software Architecture*, pp. 23–27. IEEE Computer Society.

[Navasa et al. 2009] Navasa, A., Pérez-Toledano, M. A., e Murillo, J. M. (2009). An ADL Dealing with Aspects at Software Architecture Stage. *Information and Software Technology*, 51(2):306–324.

[Navasa et al. 2002] Navasa, A., Pérez, M. A., Murillo, J. M., e Hernández, J. (2002). Aspect Oriented Software Architecture: A Structural Perspective. In *AOSD '02: Proceedings of the 1st International Conference on Aspect-Oriented Software Development.*

[Nuseibeh e Easterbrook 2000] Nuseibeh, B. e Easterbrook, S. (2000). Requirements Engineering: A Roadmap. In *Proceedings of the Conference on The Future of Software Engineering*, ICSE '00, pp. 35–46. ACM.

[OMG-SysML 2010] OMG-SysML (2010). *Systems Modeling Language (SysML) Specification - version 1.1.*

[OMG-UML 2011] OMG-UML (2011). *OMG Unified Modeling LanguageTM (OMG UML), Superstructure - version 2.4.1.*

[Parnas 1972] Parnas, D. L. (1972). On the Criteria to be Used in Decomposing Systems into Modules. *Communications of the ACM*, 15:1053–1058.

[Perez et al. 2005] Perez, J., Ali, N., Carsi, J. A., e Ramos, I. (2005). Dynamic Evolution in Aspect-Oriented Architectural Models. In *Second European Workshop on Software Architecture*, pp. 59–76. Springer-Verlag.

[Perez et al. 2006] Perez, J., Ali, N., Carsi, J. A., e Ramos, I. (2006). Designing Software Architectures with an Aspect-Oriented Architecture Description Language. In *International Symposium on Component-Based Software Engineering*, pp. 123–138. Springer-Verlag.

[Perez et al. 2008] Perez, J., Ali, N., Carsi, J. A., Ramos, I., Alvarez, B., e Pastor, P. S. J. A. (2008). Integrating Aspects in Software Architectures: PRISMA Applied to Robotic Tele-Operated Systems. *Information and Software Technology*, 50:969–990.

[Perry e Wolf 1992] Perry, D. E. e Wolf, A. L. (1992). Foundations for the Study of Software Architecture. *SIGSOFT Software Engineering Notes*, 17(4):40–52.

[Pinto et al. 2012] Pinto, M., Fuentes, L., e Fernández, L. (2012). Deriving Detailed Design Models from an Aspect-Oriented ADL Using MDD. *Journal of Systems and Software*, 85(3):525–545.

[Pinto et al. 2011] Pinto, M., Fuentes, L., e Troya, J. M. (2011). Specifying Aspect-Oriented Architectures in AO-ADL. *Information and Software Technology*, 53:1165–1182.

[Pinto et al. 2009a] Pinto, M., Fuentes, L., Valenzuela, J. A., Pires, P. F., e Delicato, F. C. (2009a). Promoting the Software Evolution in AOSD with Early Aspects: Architecture-Oriented Model-Based Pointcuts. In *Proceedings of the 2009 ICSE Workshop on Aspect-Oriented Requirements Engineering and Architecture Design*, EA '09, pp. 31–37. IEEE Computer Society.

[Pinto et al. 2009b] Pinto, M., Fuentes, L., Valenzuela, J. A., Pires, P. F., Delicato, F. C., e Marinho, E. (2009b). On the Need of Architectural Patterns in AOSD for Software Evolution. In *European Conference on Software Architecture WICSA/ECSA*, pp. 245–248. Springer.

[Ramirez et al. 2011] Ramirez, A., Vanpeperstraete, P., Rueckert, A., Odutola, K., Bennett, J., Tolke, L., e van der Wulp, M. (2011). *ArgoUML User Manual - A tutorial and reference description.*

[Rashid et al. 2006a] Rashid, A., Garcia, A., e Moreira, A. (2006a). Aspect-Oriented Software Development Beyond Programming. In *Proceedings of the 28th International Conference on Software Engineering*, ICSE '06, pp. 1061–1062. ACM.

[Rashid et al. 2006b] Rashid, A., Garcia, A., e Moreira, A. (2006b). Aspect-Oriented Software Development beyond Programming. In *ICSE '06 Proceedings of the 28th International Conference on Software Engineering*, pp. 1061–1062. ACM.

[Rashid et al. 2003] Rashid, A., Moreira, A., e Araújo, J. (2003). Modularisation and Composition of Aspectual Requirements. In *Proceedings of the 2nd International Conference on Aspect-Oriented Software Development*, AOSD '03, pp. 11–20. ACM.

[Rashid et al. 2002] Rashid, A., Sawyer, P., Moreira, A. M. D., e Araújo, J. a. (2002). Early Aspects: A Model for Aspect-Oriented Requirements Engineering. In *Proceedings of the 10th Anniversary IEEE Joint International Conference on Requirements Engineering*, RE '02, pp. 199–202. IEEE Computer Society.

[Reade 1989] Reade, C. (1989). *Elements of functional programming.* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

[Roy e Cordy 2007] Roy, C. K. e Cordy, J. R. (2007). A Survey on Software Clone Detection Research. *School of Computing TR 2007-541, Queens's University*, 115.

[Rumbaugh et al. 1991] Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F., e Lorensen, W. (1991). *Object-Oriented Modeling and Design.* Prentice-Hall, Upper Saddle River, NJ, USA.

[Runeson e Host 2009] Runeson, P. e Host, M. (2009). Guidelines for Conducting and Reporting Case Study Research in Software Engineering. *Empirical Software Engineering*, 14:131–164.

[Sampaio 2007] Sampaio, A. (2007). Analysis of the Health Watcher System Using Viewpoint-based AORE and the EA-Miner Tool. Technical report, Lancaster University, Lancaster, United Kingdom.

[Sampaio et al. 2005] Sampaio, A., Loughran, N., Rashid, A., e Rayson, P. (2005). Mining Aspects in Requirements. *Workshop on Early Aspects (held with AOSD 2005)*. Chicago, Illinois, USA.

[Sampaio e Rashid 2008] Sampaio, A. e Rashid, A. (2008). Mining Early Aspects from Requirements with EA-Miner. In *ICSE Companion '08 Companion of the 30th International Conference on Software Engineering*, pp. 911–912. ACM.

[Sampaio et al. 2007] Sampaio, A., Rashid, A., Chitchyan, R., e Rayson, P. (2007). Transactions on Aspect-Oriented Software Development III. chapter EA-Miner: Towards Automation in Aspect-Oriented Requirements Engineering, pp. 4–39. Springer-Verlag, Berlin, Heidelberg.

[Sánchez et al. 2010] Sánchez, P., Moreira, A., Fuentes, L., Araújo, J. a., e Magno, J. (2010). Model-Driven Development for Early Aspects. *Information and Software Technology*, 52(3):249–273.

[Shaker e Peters 2005] Shaker, P. e Peters, D. K. (2005). An Introduction to Aspect-Oriented Software Development. In *Newfoundland Electrical and Computer Engineering Conference*.

[Shaw 2002] Shaw, M. (2002). What Makes Good Research in Software Engineering? *International Journal of Software Tools for Technology Transfer*, 4:01–07.

[Shaw e Garlan 1996] Shaw, M. e Garlan, D. (1996). *Software Architecture: Perspectives on an Emerging Discipline*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA.

[Silva 2006] Silva, L. F. (2006). *Uma Estratégia Orientada a Aspectos para Modelagem de Requisitos*. PhD thesis, Pontifícia Universidade Católica do Rio de Janeiro, Rio de Janeiro, Brazil.

[Soares et al. 2011] Soares, M. d. S., Vrancken, J., e Verbraeck, A. (2011). User Requirements Modeling and Analysis of Software-Intensive Systems. *Journal of System and Software*, 84(2):328–339.

[Soares 2010] Soares, M. S. (2010). *Architecture-Driven Integration of Modeling Languages for the Design of Software-Intensive Systems*. PhD thesis, Delft University of Technology, Delft, Netherlands.

[Soares e Vrancken 2011] Soares, M. S. e Vrancken, J. (2011). A Framework for Multilayered Requirements Documentation and Analysis. In *Proceedings of the 2011 IEEE 35th Annual Computer Software and Applications Conference*, COMPSAC '11, pp. 308–313, Washington, DC, USA. IEEE Computer Society.

[Sommerville 2005] Sommerville, I. (2005). Integrated Requirements Engineering: A Tutorial. *IEEE Software*, 22(1):16–23.

[Sommerville 2010] Sommerville, I. (2010). *Software Engineering*. Addison Wesley, Essex, UK, 9 edition.

[Tarr et al. 1999] Tarr, P., Ossher, H., Harrison, W., e Sutton, Jr., S. M. (1999). N Degrees of Separation: Multi-Dimensional Separation of Concerns. In *Proceedings of the 21st International Conference on Software Engineering*, ICSE '99, pp. 107–119. ACM.

[Tekinerdogan et al. 2007] Tekinerdogan, B., Scholten, F., Pinto, M., e Fuentes, L. (2007). Approach for Identifying Aspects in Architectural Views. Technical report.

[Van Landuyt et al. 2009] Van Landuyt, D., Op de beeck, S., Truyen, E., e Joosen, W. (2009). Domain-Driven Discovery of Stable Abstractions for Pointcut Interfaces. In *Proceedings of the 8th ACM International Conference on Aspect-Oriented Software Development*, AOSD '09, pp. 75–86. ACM.

[Vliet 2008] Vliet, H. v. (2008). *Software Engineering: Principles and Practice*. John Wiley & Sons, England, 3rd edition.

[Weilkiens 2008] Weilkiens, T. (2008). *Systems Engineering with SysML/UML. Modeling, Analysis, Design*. Morgan Kaufmann.

[Whittle e Araujo 2004] Whittle, J. e Araujo, J. (2004). Scenario Modelling with Aspects. In *IEE Proceedings Software*, pp. 157–171. IEEE Computer Society.

[Wong et al. 2011] Wong, W. E., Tse, T. H., Glass, R. L., Basili, V. R., e Chen, T. Y. (2011). Controversy Corner: An Assessment of Systems and Software Engineering Scholars and Institutions (2003-2007 and 2004-2008). *Journal of System and Software*, 84(1):162–168.

[Woodside et al. 2009] Woodside, M., Petriu, D. C., Petriu, D. B., Xu, J., Israr, T., Georg, G., France, R., Bieman, J. M., Houmb, S. H., e Jurjens, J. (2009). Performance Analysis of Security Aspects by Weaving Scenarios Extracted from UML Models. *Journal of Systems and Software*, 82:56–74.

[Yin 2009] Yin, R. K. (2009). *Case Study Research: Design and Methods*. SAGE Publications Inc., California, United States of America, 4th edition.

[Yu 1997] Yu, E. S. K. (1997). Towards Modeling and Reasoning Support for Early-Phase Requirements Engineering. In *Proceedings of the 3rd IEEE International Symposium on Requirements Engineering*, RE '97, pp. 226–235. IEEE Computer Society.

[Yu et al. 2004] Yu, Y., Leite, J. C. S. d. P., e Mylopoulos, J. (2004). From Goals to Aspects: Discovering Aspects from Requirements Goal Models. In *Proceedings of the Requirements Engineering Conference, 12th IEEE International*, RE '04, pp. 38–47. IEEE Computer Society.

[Zave 1997] Zave, P. (1997). Classification of Research Efforts in Requirements Engineering. *ACM Computing Surveys*, 29(4):315–321.