Um processo para customização de sistemas de software utilizando componentes orientados a aspectos

Thiago Bianchi

SERVIÇO DE PÓS-GRADUAÇÃO DO ICMC-USP

Data de Depósito:

Assinatura:_____

Um processo para customização de sistemas de software utilizando componentes orientados a aspectos

Thiago Bianchi

Orientadora: **Profa. Dra. Elisa Yumi Nakagawa**

Dissertação apresentada ao Instituto de Ciências Matemáticas e de Computação - ICMC-USP, como parte dos requisitos para obtenção do título de Mestre em Ciências - Ciências de Computação e Matemática Computacional. EXEMPLAR DE DEFESA

USP – São Carlos
Abril de 2013

Ficha catalográfica elaborada pela Biblioteca Prof. Achille Bassi
e Seção Técnica de Informática, ICMC/USP,
com os dados fornecidos pelo(a) autor(a)

Bianchi, Thiago
B577p Um processo para customização de sistemas de software utilizando componentes orientados a aspectos / Thiago Bianchi; orientadora Elisa Yumi Nakagawa. -- São Carlos, 2013.
95 p.

Dissertação (Mestrado - Programa de Pós-Graduação em Ciências de Computação e Matemática Computacional) -- Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, 2013.

1. Manuteção de Software. 2. Customização de Software. 3. Programação Orientada a Aspectos. 4. Componentes de Software. I. Nakagawa, Elisa Yumi, orient. II. Título.

Dedico à minha amada esposa, Talita,
sem a qual a minha vida não seria completa.

Agradecimentos

À minha esposa, Talita Bianchi, por tudo. Agradeço pelo carinho, compreensão, incentivo e companheirismo. Sou grato a Deus pelo nosso casamento e pela bênção que é tê-la ao meu lado.

Aos meus pais, Lucas e Mary Bianchi. Vocês me deram amor, carinho e investiram na minha educação. Divido com vocês o sucesso deste trabalho.

Ao meu irmão, Lucas Bianchi, pelo incentivo e ânimo nos momentos mais difíceis.

Aos meus tios, Silas e Rosa Bianchi, pelo exemplo de vida acadêmica e incentivo para concluir este trabalho.

À orientadora e amiga Profa. Dra. Elisa Yumi Nakagawa, pela amizade, companheirismo e pelos valiosos conselhos. Nos conhecemos há muito tempo e espero que continuemos a trabalhar juntos.

Ao amigo Prof. Dr. Fernando Paulovich, pelas contribuições preciosas na análise dos resultados deste trabalho.

Aos amigos da Igreja Presbiteriana de São Carlos, pela estreita comunhão e irmandade em Cristo.

Aos professores que contribuíram para minha formação acadêmica: Rodrigo F. Mello, José Carlos Maldonado, Renata Pontin e Ellen Barbosa.

Obrigado a todas as pessoas que contribuíram de alguma forma para o desenvolvimento deste trabalho.

Ao Laboratório de Software da IBM Brasil.

À Universidade de São Paulo.

“O temor do SENHOR é o princípio do conhecimento; os loucos desprezam a sabedoria e a instrução.”

Provérbios 1:7

Resumo

A manutenção de software é reconhecida como uma atividade bastante importante, senão indispensável, da Engenharia de Software. As alterações realizadas no software durante a sua manutenção, se não forem realizadas de forma sistematizada, acabam por descaracterizar e prejudicar a manutenibilidade desse software. Nesse contexto, a customização de sistemas de software, ou seja, a manutenção adaptativa de sistemas de software para domínios e ambientes específicos, vem sendo largamente adotada pela indústria. Em outra perspectiva, a Programação Orientada a Aspectos (POA) foi proposta como uma abordagem para melhorar a separação de interesses em sistemas de software. Em paralelo, tem-se Componentes de Software que enfatizam o projeto e a construção de sistemas de software visando o aumento do reúso e da produtividade. Ainda nessa linha, algumas propostas têm sido feitas com o objetivo de combinar os conceitos de componentes de software com o paradigma de orientação a aspectos, ou seja, a criação de componentes que utilizem aspectos como interface para entrecortar outros sistemas. Vale destacar que na indústria, a manutenção de software, incluindo a customização, não tem sido realizada de maneira sistemática o que contribui para a degradação da qualidade dos sistemas. Portanto, a principal contribuição deste trabalho é propor um processo para customização de sistemas de software utilizando componentes orientados a aspectos visando a diminuição do acoplamento na adaptação de sistemas de software e, consequentemente, melhorando sua manutenibilidade. Além disso, um estudo de caso foi conduzido no qual a abordagem proposta foi utilizada na customização de um sistema de software comercial de grande porte. Os resultados desse estudo de caso foram analisados qualitativamente e contribuíram para avaliar a aplicabilidade da abordagem proposta neste trabalho como um processo sistematizado de customização de sistemas de software utilizando componentes orientados a aspectos.

Abstract

The software maintenance is recognized as a very important activity, if not indispensable, of Software Engineering. The changes made on the software during its maintenance, if not performed in a systematic way, eventually disfigure and damage the maintainability of this software. In this context, the customization of software systems, in other words, the adaptive maintenance of software systems for domains and for specific environments, has been widely adopted by industry. In another perspective, Aspect-Oriented Programming (AOP) has been proposed as an approach to improve separation of concerns in software systems. In parallel, there is Software Components that emphasize design and construction of software systems in order to increase reuse and productivity. Also in this subject, some proposals have been made with the objective to combine the concepts of software components with the aspect oriented paradigm, in other words, the creation of components that use aspects as interface to crosscut other systems. It is important to highlight that in the industry, software maintenance, including customization, has not been performed on a systematic way which contributes to the software quality degradation. Therefore, the main contribution of this work is to propose a process for customization of software systems using aspect oriented components aiming the reduction of coupling on software systems adaptation and thus improving its maintainability. Furthermore, a case study was conducted wherein the proposed approach was used to customize a large scale software system. The results of this case study were analyzed qualitatively and have contributed to evaluate the applicability of the proposed approach in this work as a systematic process to customize software systems using aspect-oriented components.

Sumário

1 Introdução **1**
1.1 Contextualização 1
1.2 Motivação 4
1.3 Objetivo 5
1.4 Organização 5

2 Revisão Bibliográfica **7**
2.1 Considerações Iniciais 7
2.2 Manutenção de Software 8
2.2.1 Conceitos Fundamentais 9
2.2.2 Evolução de Software 10
2.2.2.1 Classificações SPE e SPE+ 11
2.2.2.2 Leis de Evolução de Software 13
2.3 Orientação a Aspectos 14
2.3.1 Conceitos Fundamentais 14
2.3.2 Tecnologias de Apoio à POA 17
2.3.2.1 POA em Java 19
2.3.2.2 POA em Outras Linguagens 19
2.3.2.3 AspectJ 20
2.4 Componentes de Software 23
2.4.1 Conceitos Básicos 24
2.4.2 Métodos para o Desenvolvimento de Componentes 25
2.4.2.1 O Método Catalysis 26
2.4.2.2 O Método KobrA 26
2.4.2.3 O Método UML Components 27
2.4.3 Tecnologias de Apoio ao Desenvolvimento de Componentes 29
2.4.3.1 Componente Object Model (COM) 30
2.4.3.2 Enterprise Java Beans (EJB) 31
2.4.3.3 Commom Object Request Broker Architecture (CORBA) 32
2.4.3.4 Open Source Gateway Initiative (OSGi) 33
2.4.4 Componentes Orientados a Aspectos 34
2.4.4.1 Aspect Oriented Component Engineering (AOCE) 35
2.4.4.2 Componentes e Aspectos com UML 35

2.4.4.3 Desenvolvimento de Componentes de Prateleira com Aspectos . . . 36
2.5 Considerações Finais . . . 37

3 O Processo Prometheus **39**
3.1 Considerações Iniciais . . . 39
3.2 Visão Geral do Processo . . . 40
3.3 Detalhamento das Atividades . . . 45
3.3.1 Atividade "Receber Solicitações de Mudanças" . . . 45
3.3.2 Atividade "Análise das Solicitações de Mudanças" . . . 47
3.3.3 Atividade "Identificação dos Componentes" . . . 49
3.3.4 Atividade "Especificação dos Componentes" . . . 52
3.3.5 Atividade "Desenvolvimento e Integração dos Componentes" . . . 54
3.3.6 Atividade "Validação do Sistema" . . . 57
3.4 Considerações Finais . . . 58

4 Estudo de Caso: Aplicação do Prometheus **61**
4.1 Considerações Iniciais . . . 61
4.2 Visão Geral do Sistema Maximo . . . 62
4.3 Aplicação do Prometheus no Sistema IBM Maximo . . . 64
4.3.1 Atividade "Receber Solicitações de Mudanças" . . . 68
4.3.2 Atividade "Análise das Solicitações de Mudanças" . . . 69
4.3.3 Atividade "Identificação dos Componentes" . . . 70
4.3.4 Atividade "Especificação dos Componentes" . . . 72
4.3.5 Atividade "Desenvolvimento e Integração dos Componentes" . . . 74
4.3.6 Atividade "Validação do Sistema" . . . 76
4.4 Análise do Estudo de Caso . . . 77
4.5 Considerações Finais . . . 81

5 Conclusão **83**
5.1 Caracterização da Pesquisa Realizada . . . 83
5.2 Contribuições . . . 84
5.3 Dificuldades e Limitações . . . 85
5.4 Trabalhos Futuros . . . 85

Referências **95**

Lista de Figuras

2.1 Elementos básicos de um sistema que utiliza POA (adaptado de Kiczales et al. (1997)). 16
2.2 Trecho de código em Java e AspectJ. 21
2.3 Estrutura de um conjunto de junção nomeado (adaptada de Laddad (2009)). 22
2.4 Exemplo de adendo. 22
2.5 Conjunto de junção. 23
2.6 Visão geral do processo UML Componentes (adaptado de Cheesman e Daniels (2001)). 28
2.7 Processo de especificação de um componente segundo o método UML Components (adaptado de Cheesman e Daniels (2001)). 29

3.1 Portal do processo gerado pelo EPF. 41
3.2 Visão geral das atividades do Prometheus. 43
3.3 Visão geral da participação do Analista de Sistema no Prometheus. 44
3.4 Visão geral da participação do Analista Funcional no Prometheus. 45
3.5 Visão geral da participação do Desenvolvedor no Prometheus. 45
3.6 Fluxo de tarefas da atividade "Receber Solicitacoes de Mudancas" e artefatos utilizados por cada tarefa representados no EPF. 46
3.7 Fluxo de tarefas da atividade "Análise das Solicitações de Mudanças" e artefatos utilizados por cada tarefa representados no EPF. 47
3.8 Fluxo de tarefas da atividade "Identificação dos Componentes" e artefatos utilizados por cada tarefa representados no EPF. 50
3.9 Fluxo de tarefas da atividade "Especificação dos Componentes" e artefatos utilizados por cada tarefa representados no EPF. 53
3.10 Fluxo de tarefas da atividade "Desenvolvimento e Integração dos Componentes" e artefatos utilizados por cada tarefa representados no EPF. 55
3.11 Fluxo de tarefas da atividade "Validação do Sistema" e artefatos utilizados por cada tarefa representados no EPF. 58

4.1 Visão geral do menu de aplicações do sistema IBM Maximo. 63
4.2 Parte da tela do sistema IBM Maximo mostrando a aplicação de Ativos. . 63
4.3 Tela da versão original da aplicação de "Ordem de Serviço" do sistema IBM Maximo. 66

4.4 Tela da versão customizada para a BP da aplicação de “Ordem de Serviço” do sistema IBM Maximo. 67
4.5 XML que exemplifica uma requisição feita a um dos Web Services do sistema IBM Maximo mostrando o uso dos atributos “wmssimulate” e “username”. 69
4.6 Diagrama de componentes monstrando os componentes OA projetados. . 72
4.7 Diagrama de classes referente aos componentes OA especificados. 73
4.8 Código do aspecto de integração implementado no sistema IBM Maximo. . 74
4.9 Código do aspecto de “Ordem de Serviço” implementado no sistema IBM Maximo. 75
4.10 Diagrama de Implantação mostrando os componentes OA instalados no sistema IBM Maximo. 76
4.11 Métricas extraídas da solução WMS desenvolvida pela IBM. 79
4.12 Métricas extraídas da solução desenvolvida com o Prometheus. 79

Lista de Tabelas

2.1 Tipos de manutenção de software (adaptado de Chapin et al. (2001)) . . . 11
2.2 Leis de Lehman (adaptado de Lehman et al. (1998)) 15
2.3 Terminologia básica da Programação Orientada a Aspectos (adaptado de Filman e Friedman (2000); The AspectJ Team (2003)) 18

3.1 Papéis utilizados no Prometheus e suas responsabilidades. 44

4.1 Dados gerais sobre o sistema IBM Maximo. 64
4.2 Dados da customização do sistema IBM Maximo realizada pela BP. 65
4.3 Métricas utilizadas para comparar a solução desenvolvida pela IBM com a solução desenvolvida utilizando o Prometheus. 80

Capítulo

1

Introdução

1.1 Contextualização

O CICLO de desenvolvimento de um software inicia-se na sua concepção e só termina depois que esse software deixa de ser utilizado (Sommervile, 2007). Em outras palavras, um software, após desenvolvido, entra em operação e, durante o período em que é utilizado, sofre inúmeras alterações com propósitos variados, mas que com o tempo, podem modificar drasticamente o contexto originalmente concebido para esse software. Segundo Bennett e Rajlich (2000), essas atividades de manutenção são inevitáveis, pois todo o software que é útil e bem sucedido estimula novas solicitações dos usuários por mudanças e melhorias. O problema de manutenção de software não é novo: há mais de três décadas foi caracterizado como um *iceberg* (Canning, 1972), pois a esperança é de que o imediatamente visível seja tudo o que existe, mas é sabido que uma enorme massa de informações está oculta e irá gerar problemas de prazos e custos, muitas vezes, inestimáveis. Por isso, diversos esforços foram realizados ao longo das últimas décadas na área de Manutenção de Software com o objetivo de prover ferramentas, técnicas e métodos que auxiliassem no processo de manutenção de software na indústria (Sneed, 1995).

Nesse contexto, a manutenção de software é uma área importante no ciclo de vida de um software. Uma definição de manutenção é dada pelo padrão IEEE 1219 (IEEE, 1993) como: "*A modificação de um produto de software depois de entregue para corrigir erros, melhorar a performance ou outros atributos, ou para adaptar o produto a uma*

modificação de ambiente". Essa definição pondera que quaisquer alterações realizadas no software após sua entrega fazem parte do escopo das atividades de manutenção. Uma outra definição de manutenção, dada pela norma ISO/EIC (2002), diz que o objetivo da atividade de manutenção é realizar tais mudanças e continuar preservando a integridade do software. Em outras palavras, as atividades de manutenção não devem degradar o sistema originalmente concebido. Esse cenário é ilustrado por Osborne e Chikofsky (1990): *"Um software serviu às necessidades do negócio de uma empresa por 10 ou 15 anos. Durante este tempo, foi corrigido, adaptado e aperfeiçoado várias vezes. A equipe envolvida abordou este trabalho com a melhor dos objetivos, no entanto, algumas boas práticas de engenharia de software foram deixadas de lado por vários motivos. O software agora encontra-se instável e quase toda intervenção de manutenção ocasiona efeitos colaterais indesejáveis. A conclusão é que um importante requisito não funcional acaba de ser descumprido: a manutenibilidade"*. A manutenção de sofware envolve muitos custos e já no início da década de 80 era pesquisada por Lientz e Swanson (1980) em uma pesquisa que apontou que as atividades de manutenção adaptativa e perfectiva correspondiam a 75% dos esforços, enquanto que a correção de defeitos ocupava apenas 21% desses esforços e 4% eram relativos a outros tipos de atividades de manutenção. Muitos outros estudos posteriores e mesmo os mais atuais sugerem uma magnitude similar desse problema, tais como Pressman (2010), que afirma que apenas 20% de todo o trabalho de manutenção é gasto com a manutenção corretiva e os restantes 80% são gastos adaptando sistemas existentes a modificações no seu ambiente externo, fazendo melhorias solicitadas por usuários e submetendo uma aplicação a reengenharia para uso futuro. Esses estudos mostram que a incorporação de novos requisitos do usuário é a principal dificuldade nos processos de evolução e manutenção do software. Swanson (1976) categorizou as atividades de manutenção em quatro tipos: manutenção corretiva, manutenção adaptativa, manutenção perfectiva (ou de melhoria), e manutenção preventiva ou reengenharia). Esse conjunto de atividades de melhoria e adaptações do software são chamadas comumente na indústria de software de customizações. Esse termo aplica-se, principalmente, às atividades de manutenção adaptativa que visam alterar o software em virtude de modificações do seu ambiente de execução ou do domínio do problema (Nah et al., 2001).

Em outra perspectiva, a Programação Orientada a Aspectos (do inglês, *Aspect-Oriented Programming*) ou simplesmente POA (Kiczales et al., 1997) foi proposta como uma abordagem para melhorar a separação de interesses (do inglês, *separation of concerns*, ou simplesmente SoC) em sistemas de software. Segundo Kiczales et al. (2001b), quando se diz "separação de interesses" tem-se o intuito de ilustrar a ideia de que deveria ser possível trabalhar com o projeto ou implementação de um sistema em unidades naturais de interesse, conceito, objetivo, estrutura de equipe, entre outros, ao invés de divisões im-

postas pelos métodos, técnicas, abordagens ou ferramentas em uso. Em outras palavras, a modularidade de um sistema deveria refletir a maneira que "o desenvolver pensa a respeito" dele ao invés da maneira que a linguagem de implementação ou outras ferramentas forçam a fazê-lo. Vale a pena ainda destacar que a ideia de necessidade de modularidade e separação de interesses pode ser creditada originalmente a Parnas (Parnas, 1972). A POA começou a ser difundida recentemente na comunidade de desenvolvimento (Hoffman e Eugster, 2008), introduzindo novos conceitos no desenvolvimento de software. Com esse novos conceitos, a POA oferece uma alternativa interessante aos desenvolvedores de software na divisão e estruturação da lógica da aplicação, permitindo um aumento de modularidade das aplicações e na divisão dos esforços de desenvolvimento, por meio da utilização de interesses transversais (do inglês, *crosscutting concerns*), ou seja, interesses que estão espalhados e entrelaçados com outros interesses dentro do sistema. Segundo Elrad et al. (2001), a POA veio para melhorar ainda mais a separação de interesses quando comparada com outros paradigmas de programação, como por exemplo a orientação a objetos que tem se evidenciado nos últimos anos e melhorado significativamente a capacidade de alcançar uma boa separação de interesses. Como resultado de uma boa separação de interesses, pode-se ter a redução dos custos de desenvolvimento, bem como a facilidade no desenvolvimento, na manutenção e no reúso de software (Kiczales et al., 1997; Rashid et al., 2003).

O reúso de software é reconhecido como uma área relevante de estudo na Engenharia de Software e é considerada atualmente um fator chave para a produção de software em escala e com qualidade pela indústria (Cheesman e Daniels, 2001; Pressman, 2010). Por meio de experiências positivas e comprovadas da reutilização em diversas perspectivas, incluindo artefatos e processos de software, pode-se observar a redução no tempo de desenvolvimento, no esforço e no trabalho redundante e, como consequência, pode-se alcançar aumento da produtividade, a melhora da qualidade do software e de precisão nas estimativas (McIlroy, 1969). A ideia de criar sistemas de software a partir de sistemas existentes ao invés de construir esses sistemas do zero não é nova e foi primeiramente introduzida no final dos anos 60 (McIlroy, 1969). No entanto, observa-se que o nível de abstração de artefatos reutilizados tem mudado ao longo das últimas décadas, primeiramente, reutilizando-se código fonte como o foco principal, principalmente por meio do uso de bibliotecas de software. A seguir, um nível mais elevado de artefatos começou a ser reutilizado, tais como especificações de projeto e esquemas de bancos de dados. Por fim, a preocupação com o reúso foi inserida no contexto da engenharia de requisitos e projeto da arquitetura do software. Atualmente, os benefícios da aplicação de reúso nos vários estágios do projeto e implementação de sistemas de software são amplamente reconhecidos, uma vez que sistemas complexos e de alta qualidade precisam ser construídos, muitas ve-

zes, em períodos de tempo muito curtos e demandam uma abordagem mais organizada de reúso (Pressman, 2010). É nesse contexto que surgiu a Engenharia de Software Baseada em Componentes (do inglês, *Component Based Software Engineering*, ou simplesmente CBSE) que é um processo de software que enfatiza o projeto e a construção de sistemas baseados em computador usando componentes de software reusáveis e que incorpora a filosofia de "comprar, em vez de construir", visando um aumento da qualidade e da produtividade no desenvolvimento de sistemas de software (McIlroy, 1969). Além disso, essa abordagem desloca a ênfase da programação de software para a composição de sistemas de software (Clements, 1995). Ainda segundo Clements (1995), essa estratégia está fundamentada na pressuposição de que há muitos pontos em comum entre diversos sistemas de software de grande porte, para justificar o desenvolvimento de componentes reusáveis com o objetivo de explorar e satisfazer esses pontos em comum. Alguns trabalhos têm sido propostos com o objetivo de unir os conceitos de componentes de software com o paradigma de orientação a aspectos (Almeida et al., 2005; Clemente et al., 2002a), dentre os quais, pode-se destacar o trabalho realizado por Grundy e Patel (2001) no qual é proposta uma abordagem que utiliza componentes orientados a aspectos chamada de *Aspect Oriented Component Engineering* (AOCE). Essa abordagem concentra-se no desenvolvimento de componentes cujos serviços são transversais e procura dividi-los cuidadosamente nas interfaces dos módulos orientados a aspectos, evitando técnicas tais como, combinação de código e uso de reflexão.

1.2 Motivação

No contexto da indústria, as modificações realizadas em sistemas de software pelas atividades de manutenção e a integração com outros sistemas, muitas vezes, degradam a qualidade do código, pois introduzem acoplamentos que não foram originalmente previstos o que gera maior complexidade no projeto previamente concebido. Ainda nesse contexto, nota-se que a customização de software em escala industrial não tem sido largamente investigada. Observa-se, também, uma carência de trabalhos que investiguem o uso de componentes orientados a aspectos no contexto da customização de software em escala industrial.

Além disso, segundo Hoffman e Eugster (2008), o estudo sobre reúso com base em aspectos é relativamente recente e algumas propostas podem ser encontradas, tais como a de Rashid (2008) que utiliza reúso de aspectos para implementar dependências e interações entre requisitos e a de Zhang et al. (2007) que propõem uma técnica para construção de conectores baseada em aspectos e componentes. Contudo, percebe-se que, em geral, há

muito espaço para pesquisas e experimentação na tentativa de combinar componentes e aspectos (Eler, 2006), inclusive na área da manutenção de sistemas de software.

1.3 Objetivo

Motivado pela criticidade e relevância da execução das atividades de manutenção e evolução de sistemas software, o objetivo deste trabalho é a proposta de um processo para a customização de sistemas de software utilizando componentes orientados a aspectos. Visa-se, com isso, facilitar a adaptação, diminuir o acoplamento e aumentar a coesão nos sistemas customizados. Como consequência, pretende-se melhorar a manutenção, a evolução e o reúso desses sistemas.

Com o objetivo de analisar a viabilidade do processo proposto neste trabalho, foi conduzido um estudo de caso utilizando um sistema de software comercial no qual foi aplicada a abordagem proposta. Para analisar qualitativamente esse estudo de caso foram utilizadas métricas de acoplamento no sistema customizado.

Com isso, este trabalho visa contribuir para a área de Manutenção de Software com uma abordagem que poderá contribuir com o reúso, manutenibilidade e evolução de sistemas de software.

1.4 Organização

Neste capítulo foi apresentado o contexto no qual esta proposta de mestrado se insere e a motivação para desenvolver o trabalho proposto, bem como o seu objetivo principal. O restante desta monografia está organizada da seguinte forma. No Capítulo 2 é apresentada a revisão bibliográfica que contém os conceitos mais importantes utilizados neste trabalho. A seguir, no Capítulo 3, é detalhada a abordagem proposta neste trabalho. Essa abordagem é utilizada para customizar um sistema de software comercial conforme é descrito no Capítulo 4. Por fim, no Capítulo 5 são discutidas as contribuições, dificuldades e limitações deste trabalho, bem como as perspectivas de trabalhos futuros.

Capítulo

2

Revisão Bibliográfica

2.1 Considerações Iniciais

OS SISTEMAS de software vêm se tornando a cada dia maiores e mais complexos e continuarão evoluindo independentemente de seu domínio, seu tamanho ou sua complexidade. Essa evolução é dirigida pela mudança e ocorre em diversas situações: quando erros são corrigidos, quando o software é adaptado a um novo ambiente, quando o cliente solicita novas funcionalidades ou funções, entre outros. Essas mudanças precisam continuar ocorrendo mesmo depois que o sistema entra em operação para que permaneça útil e funcional para seus clientes e seus usuários (Sommervile, 2007).

Em uma outra perspectiva, a Programação Orientada a Aspectos (do inglês, *Aspect Oriented Programming*) ou simplesmente POA surgiu no final da década de 90 (Kiczales et al., 1997) como uma opção para solucionar dificuldades encontradas nos paradigmas existentes para desenvolvimento de software, especialmente o paradigma OO (Orientado a Objetos) (Kay, 1996). Observava-se que, independentemente da técnica de programação utilizada ou do modo como o sistema fosse decomposto, alguns interesses continuavam espalhados e/ou entrelaçados nas várias unidades do código desenvolvido; portanto, não estavam sendo cumpridos os princípios de decomposição de forma organizada e localizada em módulos separados (Elrad et al., 2001).

A engenharia de software baseada em reúso está se tornando a principal abordagem de desenvolvimento para sistemas de grande porte, ou seja, sistemas que possuem cen-

tenas (até milhares) de funcionalidades implementadas em milhões de linhas código e que necessitam de grandes equipes de profissionais para desenvolvê-los e para mantê-los (Sommervile, 2007). No final da década de 60, McIlroy (1969) propôs a Engenharia de Software Baseada em Componentes que é uma abordagem de desenvolvimento de software que enfatiza o projeto e a construção de sistemas baseados em computador usando componentes de software reusáveis e que incorpora a filosofia de "comprar, em vez de construir", visando um aumento da qualidade e da produtividade no desenvolvimento de sistemas de software.

Na Seção 2.2 são discutidos os conceitos principais relacionados à manutenção de software e é dada uma visão geral a respeito do processo de evolução de software, com destaque para o conceito de customização de software. Na Seção 2.3 são descritos os conceitos fundamentais da orientação a aspectos e é apresentada uma visão geral da terminologia básica utilizada na POA, com destaque para a Seção 2.3.2 na qual são apresentadas as principais tecnologias de apoio relacionadas à POA e que são relevantes no contexto deste trabalho. Na Seção 2.4.1 são apresentados os conceitos básicos de componentes de software e são descritos os principais métodos para desenvolvimento de componentes, com destaque para o método UML Components devido à sua relevância para este trabalho. A seguir, na Seção 2.4.3, é apresentada uma visão geral dos modelos de desenvolvimento de componentes mais conhecidos e na Seção 2.4.4, são discutidos os conceitos relacionados a componentes OA e são apresentadas as metodologias de desenvolvimento mais relevantes para a construção desse tipo de componente. Finalmente, na Seção 2.5 são apresentadas as considerações finais deste capítulo.

2.2 Manutenção de Software

É comum na indústria de software as empresas priorizarem o desenvolvimento rápido de sistemas de software, mas deixar a fase de manutenção em segundo plano. Isso é um erro, pois a fase de manutenção consome mais de 60% dos recursos disponíveis em uma empresa (Canning, 1972; Harrison e Cook, 1990) e em alguns casos, chega a consumir 90% dos custos de software (Erlikh, 2000), o que demonstra a importância da atividade de manutenção na indústria de software. Ainda segundo Pressman (2010), a manutenção de software é essencial para que o software continue em operação e preserve suas características de manutenibilidade. Além disso, é comum na indústria de software a atividade de customização de sistemas com o objetivo de adaptá-los às regras de negócio ou ao ambiente operacional da empresa que o adquiriu (Nah et al., 2001).

A dinâmica da evolução de sistemas de software está intrinsecamente relacionada ao estudo das mudanças desses sistemas e também ao processo de manutenção de software

(Sommervile, 2007). A evolução de software considera tudo o que acontece a um sistema durante o seu tempo de utilização, enquanto que a manutenção de software é mais direcionada para os objetivos das atividades que alteram um sistema. Pode-se afirmar que o estudo da manutenção e da evolução de software oferecem diferentes perspectivas sobre a natureza das mudanças do software e que existe um estreito relacionamento entre essas duas áreas (Chapin et al., 2001; Mens e Demeyer, 2008).

2.2.1 Conceitos Fundamentais

Os primeiros trabalhos publicados sobre manutenção de software capturaram as preocupações principais dos profissionais envolvidos com o desenvolvimento de sistemas (Chapin et al., 2001). Na década de 70, Canning (1972) reportou que os profissionais diziam que a manutenção era limitada à correção de erros e, também, a expansão e a extensão das funcionalidades do software. Alguns profissionais também incluiram a adaptação a mudanças em sistemas ou hardware dependentes como fatores que motivavam as atividades de manutenção. Ainda na década de 70, Swanson (1976) propôs uma divisão da atividade de manutenção de software baseada na intenção ou objetivo dominante de usuários e clientes quando solicitavam o trabalho de manutenção, ou seja, o que era visto como causa ou propósito da manutenção. Essa divisão influenciou muitos pesquisadores ao longo dos anos (Swanson e Chapin, 1995) e muitos deles utilizaram os termos "corretiva", "adaptativa" e "perfectiva", mas sem o mesmo significado originalmente proposto por Swanson. Assim, nos anos seguintes, começou a existir uma divergência dos termos e definições relacionados à manutenção de software até que foi publicado um glossário pelo IEEE (1990) contendo os termos mais comuns da engenharia de software, inclusive relacionada à manutenção de software, contendo os termos propostos originalmente por Swanson (1976). Esse glossário propõem quatro categorias para a manutenção:

- **Manutenção Corretiva:** Consiste na modificação em um sistema para correção de defeitos. Esse tipo de manutenção é reativa, pois é motivado por uma solicitação de um cliente ou um usuário para correção de defeitos;

- **Manutenção Adaptativa:** Consiste na modificação de um sistema visando adaptá-lo a um novo ambiente operacional;

- **Manutenção Perfectiva:** Qualquer modificação que visa melhorar o conjunto de funcionalidades do sistema; e

- **Manutenção Preventiva:** Refere-se à modificação realizada no sistema com o propósito de antecipar as correções de defeitos antes que estes possam ser detectados pelos usuários.

É importante ressaltar também o trabalho de Chapin et al. (2001) no qual a classificação elaborada por Swanson (1976) foi estendida e tem como critérios principais a observação dos processos e tarefas relacionadas à manutenção e no alvo da atividade de manutenção, incluindo questões não relacionadas diretamente ao código fonte do sistema, tais como documentação, treinamento, consultorias, entre outros. Essa extensão fornece uma maior granularidade, pois subdivide a atividade de manutenção em doze tipos específicos agrupando-os em quatro tipos gerais denominados de *clusters*, como mostrado na Tabela 2.1. Segundo Chapin et al. (2001), essa divisão apresenta as seguintes vantagens:

- Reconhecimento dos tipos de manutenção baseado nas evidências ao invés de basear-se nas intenções da atividade de manutenção;
- Apresenta uma granularidade maior na divisão da atividade de manutenção permitindo uma melhor gestão das tarefas e alocação do tempo; e
- Lista atividades observadas para cada um dos tipos específicos propostos facilitando a identificação do profissional que possua o perfil mais adequado para executá-las.

Finalmente, é importante ressaltar que o termo customização de software é utilizado na indústria de software em situações relacionadas à atividade de manutenção de um determinado sistema (Nah et al., 2001). Em particular, a customização ocorre quando um sistema é adquirido por uma empresa e surge a necessidade da adaptação desse sistema às novas regras de negócio ou a um novo ambiente operacional que devem ser atendidos pelo sistema comprado, mas que não foram contempladas em seu projeto original. Assim, inicia-se a atividade de customização desse sistema para adequá-lo ao domínio específico da empresa que o adquiriu (Nah et al., 2001).

2.2.2 Evolução de Software

Desde o início da década de 60, o termo evolução tem sido utilizado para caracterizar o crescimento dinâmico do software. Durante quase duas décadas foram realizados trabalhos nessa área, mas só no final da década de 70 e início da década de 80, é que começaram a ser publicados os trabalhos mais significativos a respeito da evolução de software (Lehman, 1980, 1979-1980; Lehman e Belady, 1985). No início da década de 80, após uma série de análises a vários sistemas de software de grande porte, verificou-se que esses sistemas poderiam ser classificados em tipos e, com isso, foi descrita por Lehman (1980) a especificação SPE (do inglês, *Specification*, *Program* e *Evolution*). Juntamente com a especificação SPE, foram também definidas as chamadas *"Leis da Evolução de Software"* ou *"Leis de Lehman"*. Esses assuntos são expostos com mais detalhes nas seções a seguir.

Tabela 2.1: Tipos de manutenção de software (adaptado de Chapin et al. (2001))

Tipo Geral ou *Cluster*	**Tipo Específico**	**Descrição**
Regras de Negócio	Redutiva	Visa limitar ou reduzir uma ou mais regras de negócio.
	Corretiva	Visa complementar ou refinar uma determinada regra de negócio que ficou mal especificada e teve que ser revista após sua implementação.
	Extensão	Visa estender uma regra de negócio aumentando ou modificando seu escopo.
Propriedades do Sistema	Limpeza de Código	Visa realizar melhorias na organização e manutenção do código fonte sem alterar as funcionalidades do sistema.
	Preventiva	Visa antecipar defeitos ainda não percebidos pelo usuário sem alterar as funcionalidades do sistema.
	Performance	Visa otimizar diversos aspectos do sistema relacionados à sua performance, tais como armazenamento, tempo de resposta, entre outros, sem alterar as funcionalidades do sistema.
	Adaptativa	Visa adaptar o sistema a um novo ambiente operacional.
Documentação	Reformativa	Visa alterar ou refinar a documentação externa ao código fonte do sistema .
	Atualizadora	Visa atualizar a documentação externa ao código fonte do sistema.
Suporte	Treinamento	Visa a condução de aulas e sessões de treinamento a respeito do uso e funcionamento do sistema.
	Consultoria	Visa fornecer informações a clientes e usuários a respeito das condições de uso do sistema, tais como custos, limitações, boas práticas, entre outros.
	Avaliação	Envolve atividades do tipo: auditoria, pesquisa, construção de métricas, testes de regressão, entre outras.

2.2.2.1 Classificações SPE e SPE+

A maior diferença entre sistemas de hardware e de software é a de que os sistemas de software são construídos para incorporar mudanças (Pfleeger, 2004). Lehman (1980) descreveu um modo de classificar programas em termos de como eles podem ser modificados:

- **Sistemas S (do inglês, *Specification*):** Alguns sistemas são formalmente definidos e derivados de uma especificação. Nesses sistemas, um problema específico

é declarado em termos de todo o conjunto de circustâncias às quais ele se aplica. Por exemplo, pode-se pedir para construir um sistema que realiza a adição, multiplicação e inversão de matrizes de um dado conjunto de matrizes, dentro de certas restrições de desempenho. O problema está completamente definido e existem uma ou mais soluções para o problema apresentado. A solução é bem conhecida, portanto, o desenvolvedor não se preocupa em saber se a solução está correta, mas se a implementação da solução está correta. Esse tipo de sistema não se adapta facilmente a uma mudança no problema que o gerou;

- **Sistemas P (do inglês, *Problem*):** É comum a definição de problemas abstratos utilizando sistemas do tipo S e o desenvolvimento de sistemas para solucioná-los. Entretanto, nem sempre é fácil ou possível descrever um problema do mundo real. Em muitos casos, a solução teórica para um problema existe, mas ela é impraticável ou impossível de ser implementada. Em sistemas P, os requisitos têm como base aproximações; por isso, a solução depende, em parte, da interpretação do profissional que gera os requisitos e, mesmo que exista uma solução exata, a solução produzida por um sistema P é influenciada pelo ambiente em que ela deve ser produzida. Em um sistema S, a solução é aceitável se as especificações estiverem corretas. Contudo, em um sistema P, a solução é aceitável se os resultados fizeram sentido no ambiente em que o problema está incluído; e

- **Sistemas E (do inglês, *Evolution*):** Os sistemas S e os sistemas P não contemplam situações de mudança que acontecem no mundo real, pois partem do pressuposto qua a definição do problema é imutável. Entretanto, uma terceira classe de sistemas incorpora as mudanças na natureza e no mundo real. Um sistema E está embutido no mundo real e modifica-se à medida que o mundo muda, por isso, a solução tem como base um modelo que tenta representar uma determinada realidade em observação. Logo, o sistema é parte integrante do sistema do mundo que ele modela.

Na classificação SPE, quanto maior for o número de elementos com possibilidade de sofrer alterações, maior será a probabilidade de o sistema necessitar de manutenção. Em sistemas do tipo E o problema está sempre sujeito a modificações, assim, a solução de um sistema desse tipo, provavelmente, estará em constante melhoramento.

Posteriormente, Cook et al. (2006) redefiniram essa taxonomia dando origem à taxonomia SPE+. Essa atualização da especificação SPE tem como principal inovação, a substituição da definição dos sistemas do tipo P, passando o "P" a significar paradigma, ao invés de "problema". Em outras palavras, segundo a especificação SPE+, sistemas do tipo P devem representar um paradigma ao invés de um problema.

2.2.2.2 Leis de Evolução de Software

A maioria dos trabalhos na área de evolução de software foi conduzida por Lehman e Belady (1985) nas décadas de 70 e 80 (Lehman, 1980, 1974, 1979-1980) e continuaram na década de 90 quando foi investigada a importância do *feedback* em processos evolutivos (Lehman et al., 2001; Lehman, 1996; Lehman et al., 1998). Esses estudos originaram um conjunto de leis chamadas de *"Leis de Lehman"* (Lehman et al., 1997) as quais tentam explicar as mudanças de um sistema durante o seu ciclo de vida. Para a construção dessas leis, que são na realidade hipóteses, foram examinados o crescimento e a evolução de um número considerável de sistemas de software de grande porte. Essas leis podem ser vistas na Tabela 2.2. Sommervile (2007) discutiu posteriormente cada uma das *"Leis de Lehman"*:

- A primeira lei estabelece que a manutenção do sistema é um processo inevitável. À medida que o ambiente do sistema muda, novos requisitos surgem e o sistema deve ser modificado. Quando o sistema modificado é novamente introduzido no ambiente, ocorrem mais mudanças ambientais e, assim, reinicia o processo de manutenção;

- A segunda lei estabelece que, quando o sistema é alterado, sua estrutura é degradada. A única maneira de evitar isso é investir em manutenções preventivas nas quais são empregadas horas na aprimoração da estrutura do software sem adicionar novas funcionalidades. Obviamente, isso implica custos adicionais, além daqueles para implementar as mudanças necessárias no sistema;

- De acordo com a terceira lei, os sistemas de software exibem comportamentos e tendências regulares, que podem ser medidas e previstas. Essa lei é uma consequência de fatores estruturais que influenciam e restringem as mudanças do sistema, bem como de fatores organizacionais que afetam a evolução do sistema;

- A quarta lei sugere que uma mudança de recursos ou de pessoal tem efeitos imperceptíveis na evolução do sistema no longo prazo. Isso é consistente com a Lei de Auto-Regulação que sugere que a evolução do sistema é independente das decisões gerenciais. Essa lei confirma que as equipes de desenvolvimento de grandes sistemas de software são, muitas vezes, improdutivas em virtude do tempo gasto com a comunicação interna;

- A quinta lei está relacionada aos incrementos de mudanças em cada nova versão do sistema. Ao adicionar novas funcionalidades ao sistema, inevitavelmente são introduzidos novos defeitos, por isso, quanto mais funcionalidades são introduzidas, mais defeitos são inseridos no sistema. Assim, um grande incremento nas funcionalidades

de uma versão indica um trabalho proporcional na correção de defeitos. Portanto, essa lei sugere que não se deve orçar grandes incrementos de funcionalidade sem levar em conta a necessidade de correção dos defeitos;

- A sexta e a sétima leis são similares às cinco primeiras e indicam que os usuários do sistema se tornarão cada vez mais insatisfeitos com o sistema, a menos que ele seja atualizado e novas funcionalidades sejam adicionadas; e

- A última lei reflete a mais recente contribuição em processos de *feedback*, embora não fique claro como pode ser aplicada de maneira prática no processo de desenvolvimento de software.

É importante ressaltar que as Leis de Lehman representam uma tendência a respeito de como os sistemas de software evoluem (Lehman et al., 1997) e, por isso, não devem ser interpretadas de maneira literal.

2.3 Orientação a Aspectos

A POA (Kiczales et al., 1997) foi proposta como uma abordagem para melhorar a separação de interesses (do inglês, *separation of concerns*, ou simplesmente SoC) em sistemas de software. POA começou a ser difundida recentemente na comunidade de desenvolvimento de software (Hoffman e Eugster, 2008), introduzindo novos conceitos. Utilizando-se desses conceitos, a POA tem oferecido uma alternativa interessante aos desenvolvedores de software na divisão e estruturação dos sistemas de software, permitindo um aumento da modularidade e na divisão dos esforços de desenvolvimento, por meio da utilização de interesses transversais (do inglês, *crosscutting concerns*), ou seja, interesses que estão espalhados e entrelaçados com outros interesses dentro do sistema. Segundo Elrad et al. (2001), a POA veio para melhorar ainda mais a separação de interesses quando comparada com outros paradigmas de programação, como por exemplo a orientação a objetos que tem se evidenciado nos últimos anos e contribuído significativamente para a melhora da capacidade de alcançar uma boa separação de interesses e reúso de software.

2.3.1 Conceitos Fundamentais

A POA baseou-se principalmente na ideia de que sistemas computacionais são melhor programados se os diversos interesses estiverem especificados separadamente (do inglês, *separation of concerns*), ideia concebida por Dijkstra (1976) ainda na década de 70. Esses interesses podem ser funcionais, como características ou regras de negócios, ou não funcionais, como sincronização ou gerenciamento de transações (Araújo et al., 2002).

Tabela 2.2: Leis de Lehman (adaptado de Lehman et al. (1998))

Lei	**Data da Criação**	**Descrição**
Mudança Contínua	1974	Um software que está sendo utilizado em um processo de computação no mundo real deve mudar necessariamente ou tornar-se progressivamente menos útil.
Complexidade Crescente	1974	À medida que um software muda, sua estrutura tende a se tornar mais complexa. Recursos extras devem ser dedicados para preservar e simplificar essa estrutura.
Auto-Regulação	1974	A evolução de um software é um processo auto-regulável. Atributos de sistemas, tais como tamanho, tempo entre versões e número de erros reportados é quase invariável em cada versão do sistema.
Estabilidade Organizacional	1980	Durante o ciclo de vida de um software, sua taxa de desenvolvimento é quase constante e independente de recursos dedicados ao desenvolvimento do sistema.
Conservação de Familiaridade	1980	Durante o ciclo de vida de um software, mudanças incrementais em cada versão são quase constantes.
Crescimento Contínuo	1980	A funcionalidade oferecida pelos sistemas deve aumentar continuamente para manter a satisfação do usuário.
Qualidade em Declínio	1996	A qualidade de um software entrará em declínio a menos que eles sejam adaptados a mudanças em seus ambientes operacionais.
Sistema de *Feedback*	1996	Os processos de evolução incorporam sistemas de feedback com vários agentes e *loops* e devem ser tratados como sistemas de *feedback* para que sejam atingidos os aprimoramentos significativos do sistema.

Segundo Kiczales et al. (1997), o objetivo da POA é tornar possível lidar com interesses transversais (ou entrecortantes, do inglês, *crosscutting concerns*) de um sistema tão separadamente quanto possível, permitindo aos programadores expressar cada um desses interesses de forma natural e, então, combinar automaticamente esses interesses em um formato executável. Os elementos básicos de um sistema que utiliza a POA pode ser visto na Figura 2.1. A entidade utilizada para modularizar um interesse transversal é o aspecto. O processo utilizado para fazer com que os aspectos e as funcionalidades básicas do sistema (também chamado sistema base) interajam corretamente é chamado de combinação

de aspectos (do inglês, *aspect weaving*). A ferramenta utilizada para fazer a combinação é conhecida como combinador de aspectos (do inglês, *aspect weaver*). Assim, a funcionalidade básica do sistema e os interesses tranversais podem ser desenvolvidos separadamente, com as respectivas linguagens de programação que melhor adequam-se às necessidades, e o combinador fica encarregado de produzir o código em formato executável.

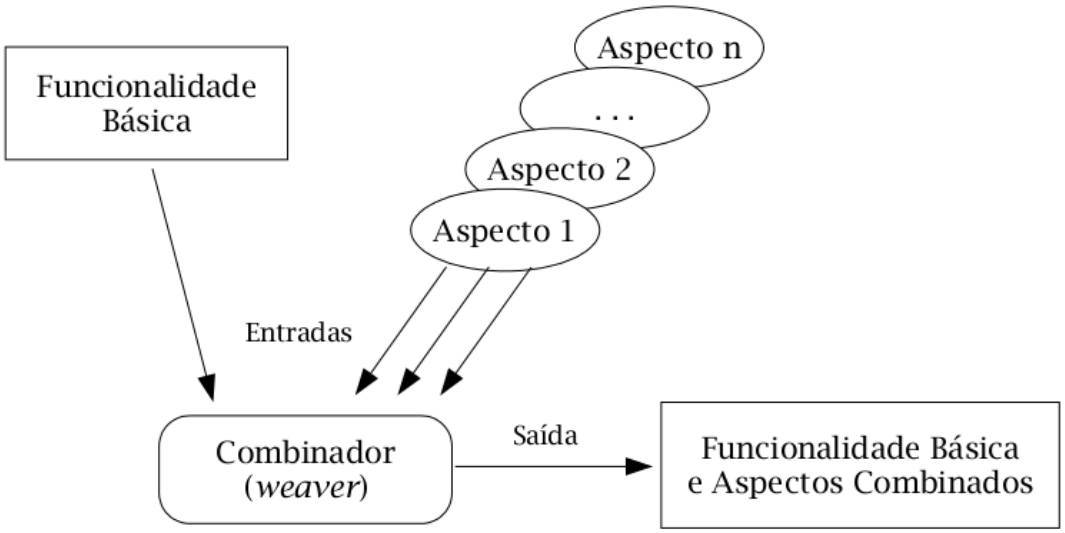

Figura 2.1: Elementos básicos de um sistema que utiliza POA (adaptado de Kiczales et al. (1997)).

Três abordagens podem ser adotadas para realizar a combinação dos aspectos com a aplicação principal que contém a funcionalidade básica do sistema (Popovici et al., 2002): (i) em tempo de compilação (*weaving* estático), no qual o código da aplicação principal é combinado com o código dos aspectos, gerando a aplicação final com os respectivos incrementos provenientes dos aspectos; (ii) em tempo de carregamento, no qual os aspectos são combinados com a aplicação compilada no momento em que essa é carregada para execução, como, por exemplo, o carregamento de classes na máquina virtual Java (do inglês, *Java Virtual Machine*, ou simplesmente JVM); e (iii) em tempo de execução (*weaving* dinâmico), no qual durante a execução da aplicação os pontos de junção (do inglês, *joint points*), que são pontos bem definidos do sistema base aos quais se pode adicionar comportamento ou alterá-lo, são identificados e os aspectos são combinados dinamicamente.

Filman e Friedman (2000) destacam que duas das mais importantes características que distinguem a POA dos demais paradigmas de programação são a quantificação e a inconsciência (do inglês, *quantification* e *obliviousness*, respectivamente). O conceito de quantificação refere-se à possibilidade de definir a execução de uma ação em um pro-

grama a qualquer momento que uma dada condição ocorrer, ou seja, pode-se definir que, dada uma condição durante a execução de um programa, uma determinada ação será executada todas as vezes que essa condição ocorrer. A inconsciência estabelece que os desenvolvedores do programa principal não precisam estar conscientes dos aspectos que as entrecortarão. Em outras palavras, os desenvolvedores não precisam preparar o código para ser alterado por aspectos.

A ideia e os conceitos propostos pela POA foi inicialmente considerada em nível de programação, mas com o tempo, o conceito de aspectos passou a ter forte relação com a maneira que os requisitos do sistema se relacionam (Jacobson e Ng, 2004). Portanto, passou a ser razoável adotar uma abordagem orientada a aspectos desde os primeiros estágios do processo de desenvolvimento de software. Assim, diversas abordagens têm sido propostas, tais como na especificação de requisitos (Araújo et al., 2002; Rashid, 2008; Rashid et al., 2003), modelagem (Guessi et al., 2011) e reúso de aspectos (Grundy e Patel, 2001; Hoffman e Eugster, 2008). Essas abordagens corroboram com a ideia de identificar os aspectos nos estágios inicias do desenvolvimento para apoiar as etapas de modelagem de requisitos e análise e projeto do sistema. Nesse contexto, as linguagens de programação orientadas a aspectos fornecem, portanto, o apoio tecnológico para implementar interesses transversais previamente projetados.

Os principais termos utilizados na POA já são consenso tanto na comunidade científica quanto na indústria de software. Em virtude de tratar-se de uma tecnologia relativamente recente, em 2004, durante o 1o Workshop Brasileiro de Desenvolvimento de Software Orientado a Aspectos (WASP'2004), buscou-se padronizar uma tradução da terminologia utilizada na POA. Assim, esses termos, destacados também por Filman et al. (2011) e pelo The AspectJ Team (2003), e suas respectivas traduções são apresentados na Tabela 2.3.

2.3.2 Tecnologias de Apoio à POA

Diversos frameworks e linguagens de propósito geral têm sido desenvolvidos para apoiar o desenvolvimento de software OA (Orientado a Aspectos) direcionadas para diferentes linguagens de programação, tais como Java, C, C++ e Smalltalk. Em geral, esses frameworks e linguagens implementam os quatro elementos básicos que uma tecnologia de apoio à POA deve ter (Elrad et al., 2001):

1. Um modelo de pontos de junção;
2. Um mecanismo de identificação dos pontos de junção;
3. Unidades que encapsulem tanto as especificações dos pontos de junção quanto os comportamentos que se deseja adicionar; e

Tabela 2.3: Terminologia básica da Programação Orientada a Aspectos (adaptado de Filman e Friedman (2000); The AspectJ Team (2003))

Termo em inglês	**Tradução**	**Descrição**
Concern	Interesse	Diferentes tópicos e questões abordados pela Engenharia de Software, podendo se referir tanto a requisitos de alto nível (tais como, o sistema deve ser gerenciável) quanto a questões de implementação (tais como, valores remotos devem ser armazenados em memória *cache*).
Crosscutting concern	Interesse entrecortante ou transversal	Interesse cuja implementação tende a estar espalhada na implementação de outros interesses.
Code tangling	Entrelaçamento de código	Ocorre quando códigos de diversos interesses estão entrelaçados. Pode ser, por exemplo, o código de um interesse transversal mesclado com os códigos dos demais interesses.
Code tangling	Espalhamento de código	Ocorre quando trechos de código similares estão distribuídos por vários módulos do programa. A modificação de um desses trechos de código torna necessária a modificação de todos os demais.
Join points	Pontos de junção	Locais bem definidos no fluxo de execução de um programa nos quais comportamentos adicionais podem ser anexados, por exemplo, por trechos de código contidos em um aspecto.
Advice	Adendo	Trecho de código que implementa um comportamento que é executado quando um ponto de junção é alcançado.
Pointcut Designator	Designador de conjunto de junção	Termo ou construção que descreve um conjunto de pontos de junção, fornecendo o mecanismo que permite a quantificação destacada por Filman e Friedman (2000).
Weaving	Combinação	É o processo de combinar, em uma unica aplicação executável, os módulos do programa principal com os aspectos como foi mostrado na Figura 2.1.
Wrapping, before and after	Empacotamento, antes e depois	Mecanismo que obriga a execução dos adendos sempre que os pontos de junção forem atingidos, podendo ser executados antes, durante ou depois dos pontos de junção.
Aspect	Aspecto	São unidades utilizadas para encapsular interesses que entrecortam as funcionalidades básicas de uma aplicação. Um aspecto contém a definição de conjuntos de junção que especificam quais os pontos de junção de interesse em uma aplicação e a definição de adendos que adicionam comportamento à aplicação caso um ponto de junção seja alcançado.

4. Um processo para combinar os aspectos com o programa principal.

Alguns desses frameworks e linguagens, tais como AspectWerkz (Bonér e Vasseur, 2005), JBoss (JBoss Inc., 2005), Spring AOP (Johnson et al., 2011), JasCo (Suvée, 2005) e AspectJ (Kiczales et al., 2001b), são apresentados a seguir. A linguagem AspectJ será descrita em mais detalhes em virtude de sua relevância para este trabalho e por ser uma das linguagens mais conhecidas e utilizadas para o desenvolvimento de sistemas OA.

2.3.2.1 POA em Java

AspectWerkz (Bonér e Vasseur, 2005), **JBoss AOP** (JBoss Inc., 2005) e **Spring e AOP** (Johnson et al., 2011) são frameworks de apoio à POA em Java. O AspectWerkz[1] permite o *weaving* em tempo de execução, sendo que os aspectos são codificados em linguagem Java ou em linguagem XML (do inglês, *eXtensible Markup Language*) (Bonér e Vasseur, 2005). O JBoss AOP[2] utiliza a linguagem Java para definir os adendos que consistem na implementação de uma interface Java padrão, enquanto que a linguagem XML é utilizada para a implementação dos conjuntos de junção (JBoss Inc., 2005). O Spring AOP[3] faz parte do Spring Framework 2[4] e permite a implementação de adendos e conjuntos de junção em classes escritas na linguagem Java, não requerendo processos especiais de compilação (Johnson et al., 2011).

JAsCo (Suvée, 2005) é uma linguagem orientada a aspectos para o desenvolvimento específico de componentes que contém elementos da POA. A linguagem busca, por um lado, incluir conceitos da programação orientada a aspectos nos componentes, fazendo com que os interesses transversais sejam modularizados adequadamente. Por outro lado, pretende incluir na programação orientada a aspectos os princípios dos componentes, com o objetivo de construir aspectos que sejam reusáveis e tenham instalação fácil e flexível.

A linguagem **AspectJ** (Kiczales et al., 2001a) é uma extensão da linguagem Java desenvolvida pelos precursores da POA como forma de dar apoio à OA em Java. Segundo seus criadores, AspectJ tem servido como base para a avaliação empírica da POA.

2.3.2.2 POA em Outras Linguagens

Para possibilitar a aplicação de uma abordagem semelhante à de AspectJ em Java, permitindo praticamente os mesmos tipos de construção, encontram-se as linguagens **AspectC++**[5] (Gal et al., 2001) e **AspectC**[6] (Coady et al., 2001), desenvolvidas para as linguagens C e C++, respectivamente.

[1] http://aspectwerkz.codehaus.org/ (acessado em 17/04/2013)
[2] http://www.jboss.org/jbossaop (acessado em 17/04/2013)
[3] http://static.springsource.org/spring/docs/2.0.x/reference/aop.html (acessado em 17/04/2013)
[4] http://www.springsource.org/ (acessado em 17/04/2013)
[5] http://www.aspectc.org/ (acessado em 17/04/2013)
[6] http://www.cs.ubc.ca/labs/spl/projects/aspectc.html (acessado em 17/04/2013)

Em Smalltalk, tem-se a linguagem **AspectS**[7] (Hirschfeld, 2001). AspectS foi implementada para o ambiente Squeak[8] e, assim como as linguagens de apoio à POA em C e C++, foi desenvolvida com uma abordagem semelhante à de AspectJ para Java (Hirschfeld, 2001).

Para possibilitar a POA em PHP, Stamey et al. (2004) desenvolveram a AOPHP. A AOPHP[9] ainda encontra-se em desenvolvimento, e a versão atual dá apoio à execução de adendos antes, em substituição e depois (*before*, *around* e *after*) da execução dos pontos de junção de um programa (Stamey et al., 2004). Além disso, a AOPHP foi desenvolvida em Java 1.5 e é executada com o servidor Web Apache[10].

2.3.2.3 AspectJ

A linguagem AspectJ foi desenvolvida no *Xerox Palo Alto Research Center*[11] e, posteriormente, foi agregada ao Projeto Eclipse[12], liderado pela empresa IBM[13]. Um programa implementado em Java pode ser normalmente compilado também com o compilador de AspectJ (The AspectJ Team, 2003).

A linguagem AspectJ (Kiczales et al., 2001b) utiliza como padrão a abordagem de combinação em tempo de compilação, embora também possibilite a combinação em tempo de carregamento. De uma forma geral, o sistema de apoio à POA pode ser utilizado para representar os três tipos de combinação existentes. Assim, o programa principal (ou programa base), em formato de código fonte ou já compilado, é combinado com os aspectos, gerando uma versão executável da aplicação adicionada dos aspectos. De uma forma geral, o sistema de apoio à POA apresentado na Figura 2.1 pode ser utilizado para representar os três tipos de combinação apresentados.

Além disso, a linguagem AspectJ permite dois tipos de implementação de interesses ou comportamentos transversais (Kiczales et al., 2001b): a dinâmica e a estática. A implementação dinâmica permite que comportamentos adicionais sejam executados quando os pontos de junção são atingidos. A implementação estática permite a inserção de novos atributos e novas operações em classes já existentes.

As construções básicas de AspectJ para permitir a implementação de interesses transversais são: conjuntos de junção, adendos, exposição de contexto e aspectos. Na Figura 2.2 é apresentado um exemplo para ilustrar a utilização de um aspecto que contém a definição de um conjunto de junção, chamado `atualiza`, a implementação de um adendo,

[7] http://map.squeak.org/package/e640e9db-2f5f-4890-a142-effebda68748 (acessado em 17/04/2013)
[8] http://www.squeak.org/ (acessado em 17/04/2013)
[9] http://www.aophp.net/ (acessado em 17/04/2013)
[10] http://httpd.apache.org/ (acessado em 17/04/2013)
[11] http://www.parc.com/ (acessado em 17/04/2013)
[12] http://www.eclipse.org/ (acessado em 17/04/2013)
[13] http://www.ibm.com/ (acessado em 17/04/2013)

linhas 50 e 51, que são utilizados para adicionar um interesse transversal, linhas 46 a 51, às classes Ponto e Linha, listadas entre as linhas 5 e 31. O código é um exemplo de aplicação escrita em AspectJ que representa uma possível implementação do interesse transversal apresentado na Figura 2.1. As quatro construções básicas são apresentadas em detalhes a seguir (The AspectJ Team, 2003):

```
1  public interface ElementoDeFigura {
2     public void move(int x,int y);
3  }
4
5  public class Ponto implements ElementoDeFigura {
6     private int x,y;
7     public Ponto(int x, int y) { this.x = x; this.y = y; }
8     public void setX(int x) { this.x = x; }
9     public void setY(int y) { this.y = y; }
10    public int getX() { return this.x; }
11    public int getY() { return this.y; }
12
13    public void move(int x, int y) {
14     this.x += x;
15     this.y += y;
16    }
17 }
18
19 public class Linha implements ElementoDeFigura {
20    private Ponto p1, p2;
21    public Linha(Ponto p1, Ponto p2) { this.p1 = p1; this.p2 = p2; }
22    public void setP1(Ponto p1) { this.p1 = p1; }
23    public void setP2(Ponto p2) { this.p2 = p2; }
24    public Ponto getP1() { return this.p1; }
25    public Ponto getP2() { return this.p2; }
26
27    public void move(int x, int y) {
28       this.p1.move(x,y);
29       this.p2.move(x,y);
30    }
31 }
32
33 public class Teste {
34    public static void main(String[] args) {
35       Ponto ponto1 = new Ponto(1,2);
36       Ponto ponto2 = new Ponto(6,7);
37       Linha linha = new Linha(ponto1,ponto2);
38       linha.move(5,5);
39    }
40 }
41
42 public class Tela {
43   public static void atualiza() { } //... comandos para atualizar a tela
44 }
45
46 public aspect AtualizaTela {
47    public pointcut atualiza():
48          call(public void ElementoDeFigura.move(int,int)) ||
49          call(public void ElementoDeFigura.set*(..));
50    after() returning: atualiza() {
51       Tela.atualiza();
52    }
53 }
```

Figura 2.2: Trecho de código em Java e AspectJ.

- **Conjuntos de Junção:** São utilizados para definir os pontos do programa principal que podem ter o comportamento alterado pelo aspecto. Os pontos de junção básicos de um programa podem ser: pontos de envio de mensagens a objetos, execução

de métodos, pontos de acesso ou definição de atributos de objetos ou pontos de instanciação de novos objetos nos quais seus construtores são executados, execução do tratador de exceções e a execução de adendos. Um ponto de junção pode ou não possuir um nome. Na Figura 2.3 é ilustrada a definição do conjunto de junção nomeado atualiza, com a respectiva identificação de cada um dos elementos que fazem parte da definição do conjunto de junção: modificador de acesso public, assinatura dos métodos que serão interceptados, o operador de composição "||" e o tipo do conjunto de junção (call);

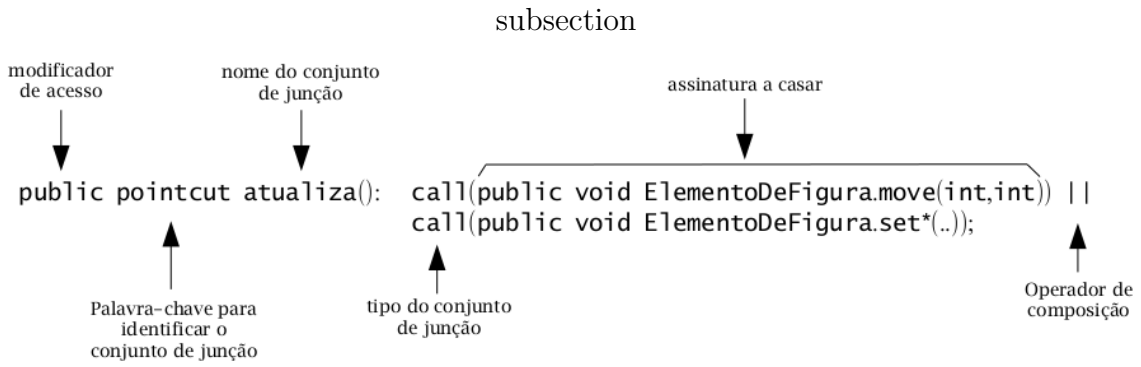

Figura 2.3: Estrutura de um conjunto de junção nomeado (adaptada de Laddad (2009)).

- **Adendo:** Quando um ponto de junção é alcançado durante a execução do programa, cada adendo que implementa o comportamento adicional do programa original pode ser executado em três momentos diferentes: antes (before), após (after) ou em substituição (around) à execução desse ponto de junção. Um exemplo de adendo implementado pelo aspecto AtualizaTela (Figura 2.2) pode ser observado na Figura 2.4. Nessa figura são mostrados os elementos básicos de um adendo: o tipo (after () returning), o nome do conjunto de junção afetado pelo adendo (atualiza) e o código a ser executado pelo adendo (Tela.atualiza);

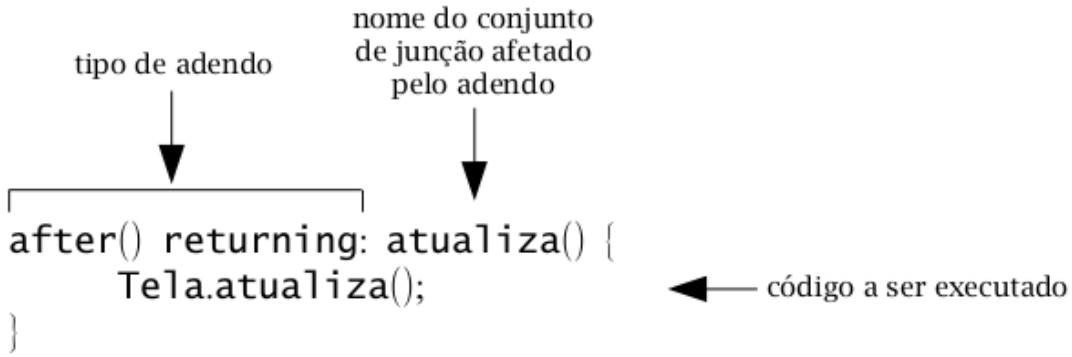

Figura 2.4: Exemplo de adendo.

- **Exposição de Contexto:** Além de identificar os pontos de junção e adicionar comportamento adicional ao programa quando esses pontos são alcançados, um conjunto de junção tem uma interface que possibilita a exposição do contexto de execução do ponto de junção. Por exemplo, o conjunto de junção mostrado na Figura 2.5 seleciona todos as chamadas a métodos públicos que recebem um valor inteiro (int) como argumento e esse argumento pode ser utilizado nos adendos que afetam esse conjunto de junção. Para isso, um adendo deve ter a mesma lista de parâmetros apresentada pelos conjuntos de junção por ele afetados; e

```
pointcut publicCall(int x):  call(public *.*(int)) && args(x);
```

Figura 2.5: Conjunto de junção.

- **Aspectos:** O padrão adotado em AspectJ é a instanciação única de aspectos. As instâncias de cada aspecto ficam disponíveis durante todo o tempo de execução da aplicação e o método estático aspectOf() é utilizado para retornar a instância de um dado aspecto. As cláusulas perthis(Pointcut), pertarget(Pointcut), percflow(Pointcut) e percflowbelow(Pointcut) são utilizadas quando existe a necessidade de uma instanciação do aspecto para cada ponto de junção selecionado pelo conjunto de junção definido em Pointcut. Um exemplo de aspecto foi mostrado na Figura 2.2 no qual foi representado o aspecto AtualizaTela utilizado para adicionar um interesse transversal às classes Ponto e Linha.

2.4 Componentes de Software

Com o crescimento de sistemas de grande porte e a complexidade de desenvolvimento gerada pelos desafios no desenvolvimento desses sistemas, novas tecnologias vêm sendo utilizadas com o objetivo de auxiliar no gerenciamento do desenvolvimento de sistemas (Pressman, 2010). Sistemas baseados em componente são um exemplo da tentativa de diminuir a complexidade no desenvolvimento de sistemas de grande porte. Nessa perspectiva, pode-se destacar as iniciativas de reúso de sistemas COTS (do inglês, *Component Of The Shelf*), sistemas compostos por outros sistemas de software, sistemas dinamicamente reconfiguráveis por usuários ou componentes, entre outros (Brown e Wallnan, 1996; Szyperski, 2002). Além disso, a reutilização sistematizada de componentes de software é facilita à medida que são utilizados métodos e modelos de desenvolvimento que auxiliem a construção de componentes de maneira que os princípios de encapsulamento, identidade e unificação de dados e funções sejam mantidos (Cheesman e Daniels, 2001). Em paralelo,

algumas propostas têm surgido com o objetivo de unir os conceitos de componentes de software com o paradigma de OA (Almeida et al., 2005; Clemente et al., 2002a), com destaque para o trabalho realizado por Grundy e Patel (2001) no qual é proposta uma abordagem que utiliza componentes orientados a aspectos chamada de *Aspect Oriented Component Engineering* (AOCE).

2.4.1 Conceitos Básicos

Componentes de software são aderentes ao princípio de dividir para conquistar com o objetivo de gerenciar complexidade, ou seja, subdividir um grande problema em pequenas partes e resolver essas pequenas partes para, então, construir uma solução mais elaboradas partindo de fundamentos simples (Pressman, 2010). Isso parece muito simples e, para muitos, soa como o desenvolvimento estruturado seguido por muitos ao longo dos anos. No entanto, a diferença principal é que componentes seguem o princípio de combinar funções e dados relacionados em uma unidade simples. Isso é divergente da abordagem estruturada que tende a focar principalmente na decomposição funcional mantendo uma forte separação entre dados e funções (Cheesman e Daniels, 2001).

Entre as diversas definições relacionadas a componentes de softwares, pode-se destacar a de Brown e Wallnan (1996), na qual um componente pode ser visto de diferentes perspectivas:

- Componente: uma parte não trivial de um sistema, praticamente independente e substituível, que preenche uma função clara no contexto de uma arquitetura bem definida;

- Componente de software em execução: um pacote dinamicamente constituído de um ou mais programas, geridos como uma unidade, ao qual se tem acesso por meio de interfaces documentadas, que podem ser descobertas durante a execução;

- Componente de software: uma unidade com dependências de contexto apenas explícita e contratualmente especificadas; e

- Componente de negócio: implementação, em software, de um conceito de negócio ou processo de negócio "autônomo".

Além dessas definições, os componentes de software podem também ser caracterizados com base no seu uso (Pressman, 2010):

- *Components Of The Shelf* (COTS): componentes comercializados e prontos para o uso;

- Componentes qualificados: avaliados por engenheiros de software para garantir que não apenas a funcionalidade, mas também o desempenho, a confiablidade, a usabilidade e outros fatores de qualidade satisfazem aos requisitos do sistema ou do produto a ser construído;

- Componentes adaptados: são encapsulados por outros componentes que controlam o acesso à sua interface para modificar características não requeridas ou indesejáveis;

- Componentes montados: integrados no estilo arquitetural[14] e interconectados com uma infra estrutura adequada para permitir que os componentes sejam coordenados e geridos efetivamente; e

- Componentes atualizados: utilizados para realizar atualizações no sistema, podendo ser substituídos por uma versão mais recente.

Por fim, os componentes de software são unidades de software estruturados de acordo com três princípios fundamentais comuns ao paradigma OO (Cheesman e Daniels, 2001; Larman, 2004):

1. Unificação de dados e funções: um componente de software consiste de valores de dados (ou estado) e das funções que processam esses dados. Essa colocação natural de dependências entre funções e dados aumenta a coesão;

2. Encapsulamento: o cliente de um componente de software desconhece a maneira como seus dados são manipulados ou como suas funções são implementadas. Em outras palavras, o cliente de um componente deve depender de sua especificação, chamada de interface, e não de sua implementação. Essa é uma separação de interesses muito importante e é fundamental para gerenciar dependências entre sistemas e reduzir o seu acoplamento; e

3. Identidade: cada componente de software tem uma identidade única, independentemente de seu estado.

2.4.2 Métodos para o Desenvolvimento de Componentes

O desenvolvimento de software baseado em componentes consiste não só no uso ou desenvolvimento de componentes em si, mas também na construção de sistemas de software por meio da composição de componentes, de acordo com um processo de desenvolvimento específico (Pressman, 2010). Encontram-se na literatura processos de desenvolvimento

[14]Entende-se por **estilo arquitetural** (ou **padrão arquitetural**) um conjunto de restrições quanto à forma e à estrutura de uma família de arquiteturas concretas ou instâncias arquiteturais (Bass et al., 2003),

baseado em componentes, entre os quais pode-se destacar o método **Catalysis** (D'Souza e Wills, 1999), o método **KobrA** (Atkinson et al., 2000) e o método **UML Components** (Cheesman e Daniels, 2001). Uma visão geral de cada um desses métodos será apresentada a seguir, com destaque para o método UML Components.

2.4.2.1 O Método Catalysis

O método Catalysis tem como foco principal fornecer um processo de desenvolvimento baseado em componentes utilizando a estratégia de combinar as características dos métodos de análise e projeto nas fases iniciais de desenvolvimento com um tratamento sistemático de refinamentos contínuos da especificação dos componentes até chegar a um projeto arquitetural bem definido e mapeá-lo para uma linguagem de programação na qual os componentes serão implementados (D'Souza e Wills, 1999).

Portanto, o método Catalysis utiliza um processo iterativo e incremental baseado na definição abstrata de conceitos e mecanismos de refinamentos da especificação e do projeto dos componentes que são aplicados no decorrer do desenvolvimento do sistema, desde os estágios iniciais até os estágios finais. Dessa forma, o desenvolvimento dos componentes sistema é visto como uma série de refinamentos dos conceitos, princípios e abstrações para níveis mais baixos de implementação, até chegar ao código.

2.4.2.2 O Método KobrA

O método KobrA foi desenvolvido no Instituto Fraunhofer[15] patrocinado pelo ministério alemão de pesquisa e tem o objetivo de dar apoio ao desenvolvimento de aplicações dirigido por modelos (do inglês, *Model Driven Application Development*) (Atkinson et al., 2000). Além disso, esse método utiliza-se da representação de componentes com a UML (OMG, 2007) para descrever a estrutura e o comportamento essencial dos componentes, independentemente da tecnologia de implementação. O método KobrA pode ser considerado ainda uma abordagem de linhas de produto para o desenvolvimento e evolução de aplicações. Isso permite que os benefícios do desenvolvimento baseado em componentes possam ser obtidos em todo o ciclo de vida do software e que a reusabilidade dos componentes possa aumentar significativamente (Atkinson et al., 2000). O desenvolvimento dirigido por modelos para a construção de componentes visa desenvolver uma arquitetura dirigida por modelos (do inglês, *Model Driven Architecture*) em que as funcionalidades principais do sistema sejam descritas independentes de plataformas específicas de implementação. Assim, os modelos desenvolvidos podem ser eficientemente transformados em componen-

[15]http://www.fraunhofer.de/index.jsp (acessado em 17/04/2013)

tes reusáveis aumentando a qualidade dos componentes e dos sistemas construídos a partir deles.

Portanto, o objetivo principal do método KobrA é fornecer um apoio para o desenvolvimento de frameworks de domínio específico baseado em componentes para, finalmente, desenvolver linhas de produto de software (Atkinson et al., 2000). Para desenvolver uma aplicação concreta, o framework genérico, que é o artefato principal produzido por esse método, é instanciado por um processo de tomada de decisões sobre o qual as funcionalidades farão parte da aplicação a ser gerada. No método KobrA, cada *Komponente*[16] do framework é descrito por um diagrama UML adequado como se fosse um sistema independente. Essa descrição é realizada em duas partes principais: a especificação, que descreve as características externas e visíveis do componente e especifica os requisitos desse componente; e a realização, que define como o componente satisfaz aos requisitos.

2.4.2.3 O Método UML Components

O método UML Components foi proposto inicialmente por Cheesman e Daniels (2001) e possui um processo bem definido para o desenvolvimento de software baseado em componentes utilizando a UML como notação para os elementos descritos no projeto. O método tem o objetivo de, a partir dos requisitos de negócio de um sistema, produzir uma arquitetura de componentes[17] com suas especificações, conforme o processo mostrado na Figura 2.6.

No processo, as caixas representam as etapas de trabalho e as setas representam os fluxos dos artefatos produzidos e consumidos por cada etapa. O processo tem como entrada os requisitos de negócio de um sistema que é usado pela etapa de Requisitos para produzir um modelo conceitual de negócio e modelos de casos de uso. Esses modelos, juntamente com os recursos existentes no sistema (sistemas legados, pacotes, base de dados) e as restrições técnicas do projeto (arquiteturas e ferramentas), formam as entradas necessárias para que a etapa de Especificação produza as especificações dos componentes, das interfaces requeridas/oferecidas e a arquitetura de componentes, mostrando como os componentes interagem entre si. Essas saídas (especificações e arquitetura) serão usadas na etapa de Provisionamento que define quais componentes serão reutilizados e quais precisam ser comprados ou construídos. A etapa de Provisionamento também inclui os testes de unidade para os componentes para que possam ser utilizados na etapa de Montagem. A seguir, a etapa de Montagem recebe como entrada os recursos existentes, os modelos de casos de uso e os modelos de componentes, para poder juntá-los e formar

[16] ***Komponente*** é o termo utilizado pelo método KobrA para se referir a um componente de software (Atkinson et al., 2000).

[17] Entende-se por **arquitetura de componentes** o conjunto de componentes de software de um sistema, suas relações estruturais e dependências de comportamento (Bass et al., 2003).

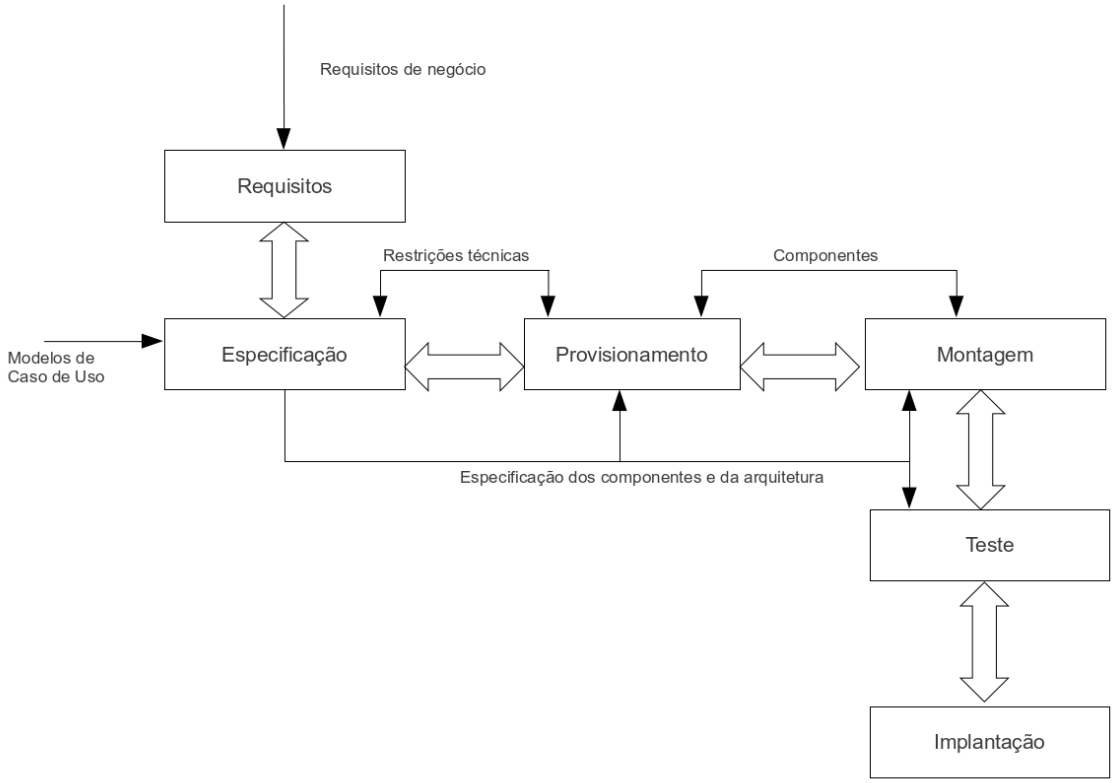

Figura 2.6: Visão geral do processo UML Componentes (adaptado de Cheesman e Daniels (2001)).

uma nova aplicação ou integrá-los em uma aplicação já existente. Finalmente, na etapa de Teste, a aplicação é testada e, em seguida, disponibilizada para entrar em operação na etapa de Implantação.

A etapa de Especificação, mostrada na Figura 2.7, é o foco principal desse método e é dividida em três sub-etapas (Cheesman e Daniels, 2001):

- **Identificação dos Componentes:** Essa etapa dá ênfase à descoberta de quais informações precisam ser gerenciadas pelos componentes, à descoberta de quais interfaces são necessárias para gerir essas informações, à descoberta de quais componentes são necessários para prover as funcionalidades e, por fim, como tudo pode ser integrado. Os objetivos principais são: identificar um conjunto inicial de interfaces de negócio para os componentes de negócio, identificar um conjunto inicial de interfaces do sistema e identificar as operações para os componentes do sistema;

- **Interação dos Componentes:** Na etapa de interação, decide-se como os componentes vão interagir entre si para implementar os requisitos do sistema. Cada operação do sistema é modelada com o uso dos diagramas de interação para descobrir as operações nas interfaces de negócio e suas assinaturas; e

Figura 2.7: Processo de especificação de um componente segundo o método UML Components (adaptado de Cheesman e Daniels (2001)).

- **Especificação dos Componentes:** O objetivo principal dessa etapa é especificar os contratos de uso e os contratos de realização dos componentes e interfaces. O contrato de uso é definido pela especificação da interface e o contrato de realização pela especificação do componente.

2.4.3 Tecnologias de Apoio ao Desenvolvimento de Componentes

Tecnologias de desenvolvimento de componentes têm se destacado ao longo do tempo como resultado da ênfase no desenvolvimento de software baseado em componentes, resultando em modelos de componentes que oferecem mecanismos com os quais os engenheiros de software podem desenvolver aplicações pela composição de componentes, definindo formas e interfaces padronizadas entre os componentes (Emmerich e Kaveh, 2001). Além disso, se os componentes de um sistema forem desenvolvidos isoladamente uns dos outros,

objetivos primários tais como, identificação e encapsulamento, são difíceis de serem alcançados, em virtude de conflitos de interfaces, formas de comunicação, divergências nas tecnologias utilizadas, entre outros. Para evitar esses conflitos, os modelos de desenvolvimento de componentes enfatizam a utilização de padrões, tais como, a forma de construir as interfaces requeridas ou oferecidas pelo componente (Pressman, 2010).

Cada tecnologia de desenvolvimento possui contêineres específicos nos quais os componentes são implantados (Sommervile, 2007). Esses contêineres implementam serviços para os componentes neles implantados, tais como registro de operações, distribuição, controle de acesso e persistência. Alguns desses serviços podem ser considerados interesses transverais do sistema, tais como persistência e controle de acesso, pois tendem a ter sua implementação espalhada pelos componentes do sistema (Rashid et al., 2003). Como os contêineres dos componentes implementam alguns interesses transversais, alguns dos problemas de entrelaçamento e espalhamento de código são amenizados, mas não são totalmente resolvidos e as chamadas aos serviços oferecidos pelos contêineres continuam espalhadas pelos componentes do sistema que utilizam esses serviços (Duclos et al., 2002).

Atualmente, os quatro modelos de componentes mais conhecidos são o COM (do inglês, *Component Object Model*) (Goswell, 1995), o EJB (do inglês, *Enterprise Java Beans*) (SUN Microsystems, 2009), o CORBA (do inglês, *Common Object Request Broker Architecture*) (OMG, 2008) e o OSGi (do inglês, *Open Source Gateway Initiative*) (OSGi, 2007), sendo que todos eles oferecem diferentes tipos de serviços para o desenvolvimento de sistemas baseados em componentes. As tecnologias COM, EJB, CORBA e OSGi oferecem os componentes COM, os *JavaBeans,* os objetos CORBA e os *bundles*, respectivamente, como componentes básicos. A comunição com os componentes é implementada por um protocolo que foi adicionado a cada modelo de componente básico, no qual o COM utiliza o DCOM (do inglês, *Distributed Component Object Model*), o EJB utiliza o RMI (do inglês, *Remote Method Invocation*), o CORBA utiliza o protocolo IIOP (do inglês, *Internet Inter-ORB Protocol*) e o OSGi utiliza os protocolos HTTP (do inglês, *HyperText Transfer Protocol*), UPnP (do inglês, *Universal Plug and Play*) e DMT (do inglês, *Discrete Multi-Tone*). A seguir, cada um desses quatro modelos de implementação de componentes será visto com mais detalhes.

2.4.3.1 Componente Object Model (COM)

A tecnologia COM foi desenvolvida primeiramente para a plataforma *Microsoft Windows*[18], mas é um padrão aberto e tem sido implementado em outras plataformas. Essa tecnologia fornece um modelo para o projeto de componentes que possuem múltiplas interfaces com vinculação dinâmica com outros componentes. Essas interfaces são binárias,

[18]http://www.microsoft.com/windows/ (acessado em 17/04/2013)

o que torna possível implementar os componentes em uma variedade de linguagens de programação, tais como C++, Visual Basic e Java. Para acessar as interfaces necessárias, um componente utiliza o modelo COM para localizar o servidor de componentes e então procurar a interface desejada. O COM+ é uma extensão do COM com tecnologias que apoiam transações, serviços de diretório, balanceamento de carga e fila de mensagens (Goswell, 1995).

O DCOM é um protocolo utilizado para tornar transparente a comunicação entre os componentes, de maneira que se tenha a impressão de que um componente que está em outro servidor esteja sendo acessado localmente. Para isso, um cliente comunica-se com um *proxy* que localiza o servidor no qual o componente requisitado está implantado e gerencia a comunicação entre eles (Goswell, 1995).

2.4.3.2 Enterprise Java Beans (EJB)

Na parte central da arquitetura JEE[19] (do inglês, *Java Enterprise Edition*) está um modelo de três camadas (do inglês, *three tier*) que é constituído pelas camadas cliente, web e servidor (SUN Microsystems, 2009). Em particular, na camada servidor encontra-se o EJB, que é um modelo de componentes que permite que o desenvolvimento de aplicações baseadas na tecnologia Java seja rápido e simples (Rubinger e Burke, 2010). O EJB possui suporte para transações e segurança e está implementado sobre um protocolo de objetos independente de plataforma o que permite que o sistema seja desenvolvido na plataforma que for mais conveniente. Além disso, o EJB é um modelo de componente portável e reusável, pois é implementado em Java e pode encapsular regras de negócio para ser utilizado em diversos contextos.

Uma arquitetura EJB padrão consiste de um servidor de aplicação que possui um contêiner de componentes EJB. Além disso, clientes EJB e outras tecnologias auxiliares dão apoio à construção de sistemas utilizando esses tipos de componentes. O contêiner EJB oferece diversos serviços que podem ser utilizados pelos componentes nele implantados, entre as quais pode-se destacar (SUN Microsystems, 2009): o JNDI (do inglês, *Java Naming Directory Interface*), JTS (do inglês, *Java Transaction Service*), JDBC (do inglês, *Java Data Base Connection*), JSP (do inglês, *Java Server Pages*), RMI (do inglês, *Remote Method Invocation*), entre outras.

O modelo de componentes EJB pode ser dividido em quatro tipos específicos, de acordo com a necessidade de sua utilização (Rubinger e Burke, 2010):

- **EJB de sessão sem estado (do inglês, *Statelless Session Bean*):** É responsável por executar as operações que não necessitam armazenar estado entre uma execução e outra;

[19] http://www.oracle.com/technetwork/java/javaee/tech/index.html (acessado em 17/04/2013)

- **EJB de sessão com estado (do inglês, *Statefull Session Bean*):** É similar ao EJB de sessão sem estado, exceto pelo fato de que armazena o estado entre uma execução e outra;
- **EJB de entidade (do inglês, *Entity Bean*):** É responsável por representar um objeto que é armazenado dentro de uma base de dados; e
- **EJB dirigido por mensagem (do inglês, *Message Driven Bean*):** É responsável por trabalhar com o gerenciamento de mensagens assíncronas utilizando gerenciamento de filas para esse processamento.

Um cliente de um componente EJB pode estar localizado no mesmo contêiner que o componente ou estar fisicamente em outro local. O modelo de componentes EJB pode ter clientes implementados em diversos tipos de tecnologias, entre as quais pode-se destacar classes Java, *Servlets*[20] ou outro componente EJB (SUN Microsystems, 2009).

2.4.3.3 Commom Object Request Broker Architecture (CORBA)

No início da década de 90, a OMG (do inglês, *Object Management Group*) criou a tecnologia CORBA com o intuito de criar um framework comum para integrar e permitir interconexão entre aplicações desenvolvidas com diferentes linguagens, implementações e plataformas, com o objetivo de alcançar reusabilidade, portabilidade e interoperabilidade de componentes de software em um ambiente distribuído (Szyperski, 2002).

O CORBA tem essencialmente três partes (OMG, 2008): um conjunto de interfaces de invocação, o ORB (do inglês, *Object Request Broker*) e um conjunto de adaptadores para objetos. Para realizar a invocação de métodos com essa tecnologia, é necessário que vinculações de implementações sejam feitas tardiamente e, para isso, as interfaces de invocação permitem vários graus de vinculação tardia. O ORB possui um registro de objetos e oferece mecanismos de transparência de localidade o que permite que as requisições possam ser feitas local ou remotamente. Os programas que precisam funcionar como objetos que oferecem serviços precisam ser registrados no ORB por meio dos adaptadores de objetos. Quando um componente precisa invocar um método de outro componente, as interfaces de invocação enfileiram os argumentos da invocação e o ORB localiza o objeto receptor da mensagem e sua máquina hospedeira. Tendo feito isso, o ORB invoca o método requisitado e transporta os argumentos da mensagem para o adaptador de objetos que recebe os argumentos e invoca o método do objeto receptor da mensagem. Para que as invocações de interfaces e os adaptadores dos objetos funcionem, é necessário que todas as interfaces dos objetos estejam descritas em uma linguagem comum e as linguagens de implementação dos objetos tenham vinculação para essa linguagem (OMG, 2008).

[20] http://www.oracle.com/technetwork/java/javaee/servlet/index.html (acessado em 17/04/2013)

Essa linguagem comum é chamada de IDL (do inglês, *Interface Definition Language*) e é utilizada para especificar as interfaces dos objetos ou componentes e gerar os esqueletos (do inglês, *stubs*) de código necessários. A tecnologia CORBA é parte da arquitetura OMA (do inglês, *Object Management Architecture*), que oferece serviços para os objetos, tais como serviço de nomeação de objetos, persistência, controle e notificação de eventos, transações, relacionamentos, segurança, concorrência, entre outros (OMG, 2008).

2.4.3.4 Open Source Gateway Initiative (OSGi)

O framework OSGi é um sistema modular e uma plataforma de serviços para a linguagem de programação Java que oferece um modelo de componentes dinâmico (OSGi, 2007). Esses componentes vêm na forma de *bundles* (que podem ser traduzidos como "pacotes") que podem ter o seu ciclo de vida remotamente controlados, ou seja, podem ser instalados, inicializados, parados, atualizados e desinstalados sem requerer uma reinicialização do sistema. O processo de registro dos componentes permite que sejam detectados a remoção ou inclusão de novos componentes com o objetivo de permitir a adaptação dinâmica do sistema (OSGi, 2007).

O OSGi é um modelo de componentes que possui diversas implementações, entre as quais pode-se destacar: Equinox [21], Apache Felix [22] e Knopflerfish [23]. É importante ressaltar que a especificação estabelecida pelo modelo OSGi requer que todas as implementações tenham o mesmo conjunto de funcionalidades básicas que são divididas em (OSGi, 2007):

- ***Bundles*:** São componentes empactados em formato jar[24] contendo um arquivo descrevendo os serviços declarados em seu conteúdo;
- **Serviços:** A camada de serviços conecta os *bundles* de uma forma dinâmica utilizando um gerenciamento de eventos relacionados com a inicialização e a busca dos *bundles*;
- **Registro de Serviços:** Corresponde à camada responsável pelo gerenciamento dos serviços e eventos relacionados a esses serviços;
- **Ciclo de Vida:** Corresponde à camada responsável por gerenciar o ciclo de vida de um *bundle*: instalação, inicialização, parada, atualização e desinstalação;
- **Módulo:** Corresponde à camada que define o encapsulamento e a declaração de dependências de cada *bundle*;

[21] http://equinoxosgi.org/ (acessado em 17/04/2013)
[22] http://felix.apache.org/site/index.html (acessado em 17/04/2013)
[23] http://www.knopflerfish.org/ (acessado em 17/04/2013)
[24] http://download.oracle.com/javase/6/docs/technotes/guides/jar/index.html (acessado em 17/04/2013)

- **Segurança:** Corresponde à camada que define as características relacionadas às restrições de acesso aos serviços contidos em um *bundle*; e

- **Ambiente de Execução:** Define quais métodos e classes estão disponíveis em uma plataforma específica.

O modelo de desenvolvimento de componentes OSGi é responsabilidade de um consórcio formado por diversas empresas chamado *OSGi Alliance*[25] (OSGi, 2007) e, por isso, visa atender às necessidades técnicas e funcionais em diversos cenários de utilização. Essa característica tem tornado esse modelo um dos mais utilizados atualmente para construção de diversos sistemas de software complexos e de grande porte, entre os quais pode-se destacar a IDE Eclipse [26] e os servidores de aplicação JEE Websphere[27] e Glassfish[28].

2.4.4 Componentes Orientados a Aspectos

A CBSE impõe um ciclo não trivial na identificação de componentes (Allen, 2001; Allen e Frost, 1998; D'Souza e Wills, 1999): os desenvolvedores devem identificar apropriadamente abstrações de componentes, alocar responsabilidades cuidadosamente para minimisar duplicações ou inconsistências, e devem tomar cuidado na combinação e configuração desses componentes. Esse tipo de abordagem dispersa os interesses tranversais por vários módulos do sistema, especialmente aqueles originados por requisitos não funcionais (Mezini e Lieberherr, 1998).

Na tentativa de solucionar os problemas originários do desenvolvimento baseado em componentes, alguns trabalhos de integração entre os conceitos de componentes e orientação a aspectos têm sido desenvolvidos (Clemente et al., 2002a; Harrison et al., 2005; Perez et al., 2006) com o objetivo de criar mecanismos para modularizar os interesses transversais do sistema e minimizar os problemas relacionados ao entrelaçamento e espalhamento de código. É importante ressaltar que a integração entre aspectos e componentes não é uma tarefa simples, pois os componentes possuem certos princípios e objetivos que devem ser respeitados nesse processo, tais como o conceito de encapsulamento do componente, certificação de qualidade e a previsibilidade de composição dos componentes (Cottenier e Elrad, 2004). Dentre esses trabalhos, vale a pena destacar as abordagens de Grundy e Patel (2001), de Clemente et al. (2002b) e de Almeida et al. (2005). Esses trabalhos foram obtidos por meio de uma Revisão Sistemática (Kitchenham e Charters, 2007) conduzida no contexto deste trabalho.

[25] http://www.osgi.org/Main/HomePage (acessado em 17/04/2013)
[26] http://www.eclipse.org/ (acessado em 17/04/2013)
[27] http://www-01.ibm.com/software/websphere/ (acessado em 17/04/2013)
[28] http://glassfish.java.net/ (acessado em 17/04/2013)

2.4.4.1 Aspect Oriented Component Engineering (AOCE)

A abordagem de Grundy e Patel (2001) para o projeto e desenvolvimento de sistemas com componentes é chamada de *Aspect Oriented Component Engineering* (AOCE). Essa abordagem concentra-se no desenvolvimento de componentes cujos serviços são transversais, oferecendo serviços a vários a outros componentes, e procura fatorá-los nas interfaces dos módulos aspectuais, evitando técnicas tais como, como combinação de código e uso de mecanismos de reflexão. O processo de desenvolvimento proposto por essa abordagem é composto pelas seguintes etapas: Engenharia de Requisitos, Especificação e Projeto, e Implementação dos Componentes.

A etapa de Engenharia de Requisitos tem o objetivo de analisar os requisitos do sistema ou dos componentes, construir diagramas de classe e modelos conceituais OO e, a partir daí, identificar possíveis componentes a serem construídos. Em seguida, são identificadas as restrições não funcionais e serviços requeridos para cada componente e identificados os aspectos e detalhes aspectuais do sistema. Os detalhes aspectuais descrevem características do componente relacionadas ao aspecto, tais como as funções requeridas e oferecidas pelo componente transversal.

A etapa de Especificação e Projeto dos Componentes inclui o refinamento dos requisitos de componentes para o projeto desses componentes, no qual os aspectos refletem os requisitos do sistema em detalhes aspectuais e propriedades dos detalhes aspectuais. Nessa etapa, são utilizados diagramas de interação e colaboração da UML para identificar funções e operações do sistema. Essa etapa é concluída com a atribuição de funções e responsabilidades de dados aos componentes.

Para a etapa de Projeto dos Componentes, Grundy e Patel (2001) propuseram diagramas que representam os componentes do sistema e seus aspectos. Além disso, os criadores dessa abordagem também propuseram extensões ao meta-modelo e à notação visual da UML para incluir componentes transversais para apoiar a modelagem com componentes e aspectos.

2.4.4.2 Componentes e Aspectos com UML

Clemente et al. (2002b) apresentaram um processo em que a POA é utilizada para descrever e implementar as dependências entre os componentes de software de um sistema. A maior contribuição desse trabalho é o uso da UML para representar os aspectos em todas as fases genéricas de desenvolvimento com componentes. Essas fases são: Especificação, Interação, Montagem e Implantação dos Componentes.

Na fase de Especificação, as interfaces oferecidas e requeridas pelos componentes são identificadas, bem como as dependências entre cada componente. Além disso, cada uma

das dependências entre os componentes são marcadas como não intrínsecas ou intrínsecas. A dependência não intrínseca é aquela em que o componente pode precisar de um serviço que é oferecido pelo contêiner ou ambiente operacional no qual o componente será usado, enquanto que a dependência intrínseca é uma dependência em que o componente necessáriamente precisa ter satisfeita para o seu correto funcionamento.

Na fase de Interação, o comportamento dos componentes é modelado utilizando diagramas de interação da UML para explicitar cada uma das operações oferecidas.

Finalmente, na fase de Montagem e Implantação é utilizado o diagrama de implantação da UML para especificar o ambiente operacional em que cada componente será implantado.

É importante ressaltar que a abordagem de Clemente et al. (2002b) apresenta o uso da UML no projeto de componentes que possuem interações aspectuais com outros componentes e/ou aspectos. Entretanto, não é apresentada um processo para o desenvolvimento de software com componentes e aspectos que explicite como os aspectos serão capturados desde as etapas iniciais de elicitação dos requisitos até a maneira como esses aspectos serão implementados em cada um dos componentes. Além disso, as etapas de montagem e implementação são essencialmente simplistas, resumindo-se a tão somente indicar o uso do diagrama de implantação da UML para documentar a implantação dos componentes. Finalmente, outro ponto a ser considerado nessa abordagem é o fato de não estar claro se a interação entre os componentes e os aspectos é feita por meio de entrecorte ou por meio de chamadas a métodos ou operações que estão implementados nos aspectos do sistema ou nos contêineres.

2.4.4.3 Desenvolvimento de Componentes de Prateleira com Aspectos

Nessa abordagem, Almeida et al. (2005) propõe a utilização de aspectos na implementação de componentes de prateleira. A ideia é implementar requisitos não funcionais, tais como registro de operações e controle de acesso, como aspectos, no qual esses aspectos atuam internamente ao componente. Cada componente de prateleira pode possuir requisitos não funcionais implementados como aspectos que podem ser ligados ou desligados pelo desenvolvedor que utiliza esses componentes.

As limitações dessa abordagem são que todos os componentes de prateleira terão que possuir vários interesses transversais implementados internamente como aspectos e interfaces a serem oferecidas para cada um deles ser ativado ou desativado no sistema. Além disso, a abordagem não deixa claro como essas interfaces funcionam. Outro problema é que não se pode prever quais requisitos não funcionais o componente deverá possuir, pois como trata-se de um componente de prateleira é difícil prever todos os requisitos não funcionais que deverão ser atendidos.

2.5 Considerações Finais

Neste capítulo, foram apresentados conceitos da área de Manutenção de Software, Orientação a Aspectos e Componentes de Software, importantes no contexto deste trabalho. Em particular, foram discutidos conceitos e classificações sobre manutenção de software e sobre o conceito de customização de software. Além disso, foram apresentados os principais pontos relacionados à evolução de software. A seguir, foram apresentados os principais conceitos relacionados à área de Orientação a Aspectos, inclusive as tecnologias de apoio, que viabilizam implementar a separação de interesses proposto pela POA como forma de diminuir o acoplamento e melhorar a manutenção de um sistema de software. Em relação às tecnologias de apoio à POA, foi dado destaque à linguagem AspectJ devido a sua relevância no contexto deste trabalho. Finalmente, foram apresentados conceitos da área de Componentes de Software, importantes no contexto deste trabalho. Em particular, foram discutidos conceitos sobre definição de componentes de software. Além disso, foram apresentados os principais métodos para o desenvolvimento de componentes, com destaque para o método UML Components. A seguir, foi dada uma visão geral a respeito das tecnologias empregadas no desenvolvimento de componentes com destaque para os métodos utilizados no desenvolvimento de componentes OA. No próximo capítulo, será apresentado o tema principal deste trabalho: o Prometheus que é um processo para customização de sistemas de software utilizando componentes OA.

Capítulo

3

O Processo Prometheus

3.1 Considerações Iniciais

MOTIVADO pela criticidade e relevância das atividades de customização de sistemas de software, a principal contribuição deste trabalho é a proposta de um processo que visa reduzir os esforços para a customização de sistemas de software. Para isso, são utilizados componentes OA no contexto de uma abordagem sistêmica na qual as atividades de customização são organizadas de maneira que os sistemas de software possam ser mais facilmente adaptados. É importante ressaltar que a indústria de software tem utilizado em larga escala processos de desenvolvimento baseados no Processo Unificado (Jacobson et al., 1999). Dessa maneira, pretende-se utilizar os conceitos do Processo Unificado aliados à experiência retirada da indústria no processo proposto neste trabalho. Como consequência, pretende-se preservar o acoplamento e a coesão dos sistemas de software afim de que suas capacidades de manutenção, evolução e reúso sejam maximizadas. Em especial, essa abordagem pode ser vantajosa para sistemas de grande porte, tais como ERP's (do inglês, *Enterprise Resource Planning*) (Leon, 2007), nos quais as modificações podem ocorrer em diversas partes do código fonte o que pode dificultar a sua manutenção, evolução e reúso futuros.

Assim, neste capítulo é apresentado o Prometheus que é um processo para customização de sistemas de software utilizando componentes OA. O Prometheus tem o objetivo de

ser simples o bastante para facilitar sua adoção na indústria de software, mas completo o suficiente para suportar a construção de um componente OA de maneira adequada.

Na Seção 3.2 é dada uma visão geral do Prometheus e são introduzidos os principais conceitos utilizados em sua concepção. A seguir, na Seção 3.3, são mostradas as atividades que compõem o Prometheus. Finalmente, na Seção 3.4 são apresentadas as considerações finais deste capítulo.

3.2 Visão Geral do Processo

O processo proposto no contexto deste trabalho foi elaborado tendo como base abordagens bem estabelicidas para construção de sistemas de software baseado em componentes, tais como o UML Components (Cheesman e Daniels, 2001), e em aspectos, tais como *Aspect Oriented Component Engineering* (Grundy e Patel, 2001), Componentes e Aspectos com UML (Clemente et al., 2002b) e Desenvolvimento de Componentes de Prateleira com Aspectos(Almeida et al., 2005). Além disso, para facilitar o entendimento e a utilização desse processo pela indústria de software foi utilizada a ferramenta *Eclipse Process Framework*[1] (ou simplesmente EPF). Dessa maneira, foram incorporados no Prometheus alguns conceitos inerentes a essa ferramenta com o objetivo de facilitar a a sua compreensão. Dentre esses conceitos, pode-se destacar:

- Produto de Trabalho: é todos o tipos de artefato ou resultado da execução de uma tarefa. Podem ser de dois tipos: (i) artefatos (do inglês, *artifacts*), que são informações utilizadas ou produzidas pelo processo; e (ii) resultados (do inglês, *outcome*), que representam o estado de um artefato utilizado por uma tarefa e que é relevante para o processo;

- Atividade: é um conjunto de uma ou mais tarefas ou atividades e, normalmente, possui um diagrama visual associado para ilustrar o fluxo de execução dos elementos envolvidos;

- Tarefa: é uma ação que deve ser desempenhada por um papel específico;

- Papel: representa uma pessoa que executará tarefas e será responsável por modificar produtos de trabalho;

- Disciplina: as atividades, papéis e produtos de trabalho são agrupados em categorias de acordo com suas semelhanças.

[1] *http://www.eclipse.org/epf/* acessado em 17/04/2013

É importante ressaltar que os conceitos citados acima são utilizados em diversos processos de desenvolvimento de software baseados no Processo Unificado, tais como EUP[2] (do inglês, *Enterprise Unified Process*), RUP[3] (do inglês, *Rational Unified Process*) e OpenUP[4] (do inglês, *The Open Unified Process*) o que facilita a utilização do Prometheus por empresas que utilizam processos baseados nesses processos. Assim, o EPF foi escolhido como ferramenta para ilustrar e divulgar o processo proposto neste trabalho que pode ser acessado em `http://143.107.231.100:8080/prometheus/`.

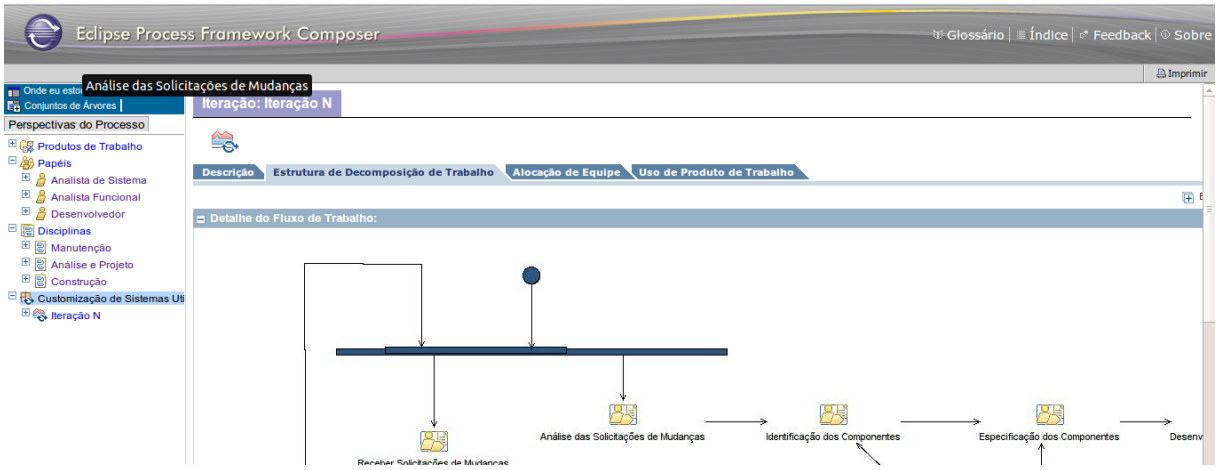

Figura 3.1: Portal do processo gerado pelo EPF.

O diagrama visual utilizado pelo EPF para ilustrar o fluxo de atividades utiliza a mesma notação que o diagrama de atividades da UML (OMG, 2007) como pode ser visto na Figura 3.2 na qual é mostrada uma visão geral das atividades do Prometheus. No diagrama mostrado na Figura 3.2, pode-se destacar 3 aspectos importantes do Prometheus:

- Seis atividades: "Receber Solicitações de Mudança", "Análise das Solicitações de Mudanças", "Identificação dos Componentes", "Especificação dos Componentes", "Desenvolvimento e Integração dos Componentes" e "Validação do Sistema". Em outras abordagens, tais como UML Components(Cheesman e Daniels, 2001) e AOCE (*Aspect Oriented Component Engineering*) (Grundy e Patel, 2001) essas atividades são chamadas de fases ou etapas. Cada atividade do Prometheus é constituída de uma ou mais tarefas que serão detalhadas nas seções a seguir;

- Desenvolvimento iterativo: a atividade "Validação do Sistema" tem como objetivo validar o sistema customizado após a implantação do componente OA e permite retornar para atividades anteriores do processo. Além disso, o ciclo se repete enquanto houverem mais solicitações de mudanças para serem incorporadas no sistema; e

[2]http://www.enterpriseunifiedprocess.com/ acessado em 17/04/2013
[3]http://www-01.ibm.com/software/awdtools/rup/ acessado em 17/04/2013
[4]http://epf.eclipse.org/wikis/openup/ acessado em 17/04/2013

- Desenvolvimento baseado em incrementos: cada componente OA é projetado segundo a abordagem "*top-down*", na qual primeiro são identificadas suas interfaces e por último são detalhados as suas operações. No Prometheus, a atividade "Identificação dos Componentes" inicia o projeto do componente OA e a atividade "Especificação dos Componentes" refina o projeto desse componente. Além disso, em cada uma dessas atividades, cada solicitação de mudança é considerada separadamente e os modelos construídos são incrementados a cada iteração.

O EPF permite que todas as partes do processo sejam especificadas e, para tal, ele permite a geração de um portal no qual é possível navegar pelo processo como pode ser visto na Figura 3.1, na qual é mostrado o portal gerado pelo EPF para o Prometheus. Nesse portal, existe um menu para navegar entre os vários elementos do Prometheus, entre os quais pode-se destacar:

- Papéis: No Prometheus, existem os seguintes papéis:
  - Analista de Sistema: responsável pelas atividades de análise e projeto do(s) componente(s) OA. Em algumas empresas, esse papel também é chamado de "Arquiteto";
  - Analista Funcional: responsável pelas atividades relacionadas com o aspecto funcional da manutenção do sistema. Em algumas empresas, esse papel também é chamado de "Analista de Requisitos"; e
  - Desenvolvedor: responsável pelas atividades de codificação do(s) componente(s) OA. Em algumas empresas, esse papel também é chamado de "Programador".
- Disciplinas: No Prometheus, existem as seguintes disciplinas:
  - Manutenção: agrupa as atividades, as tarefas e os produtos de trabalho relacionados as funcionalidades que serão alteradas ou adicionadas no sistema;
  - Análise e Projeto: agrupa as atividades, as tarefas e os produtos de trabalho relacionados com a definição dos componentes OA; e
  - Construção: agrupa as atividades, as tarefas e os produtos de trabalhos relacionados com a implementação dos componentes OA.

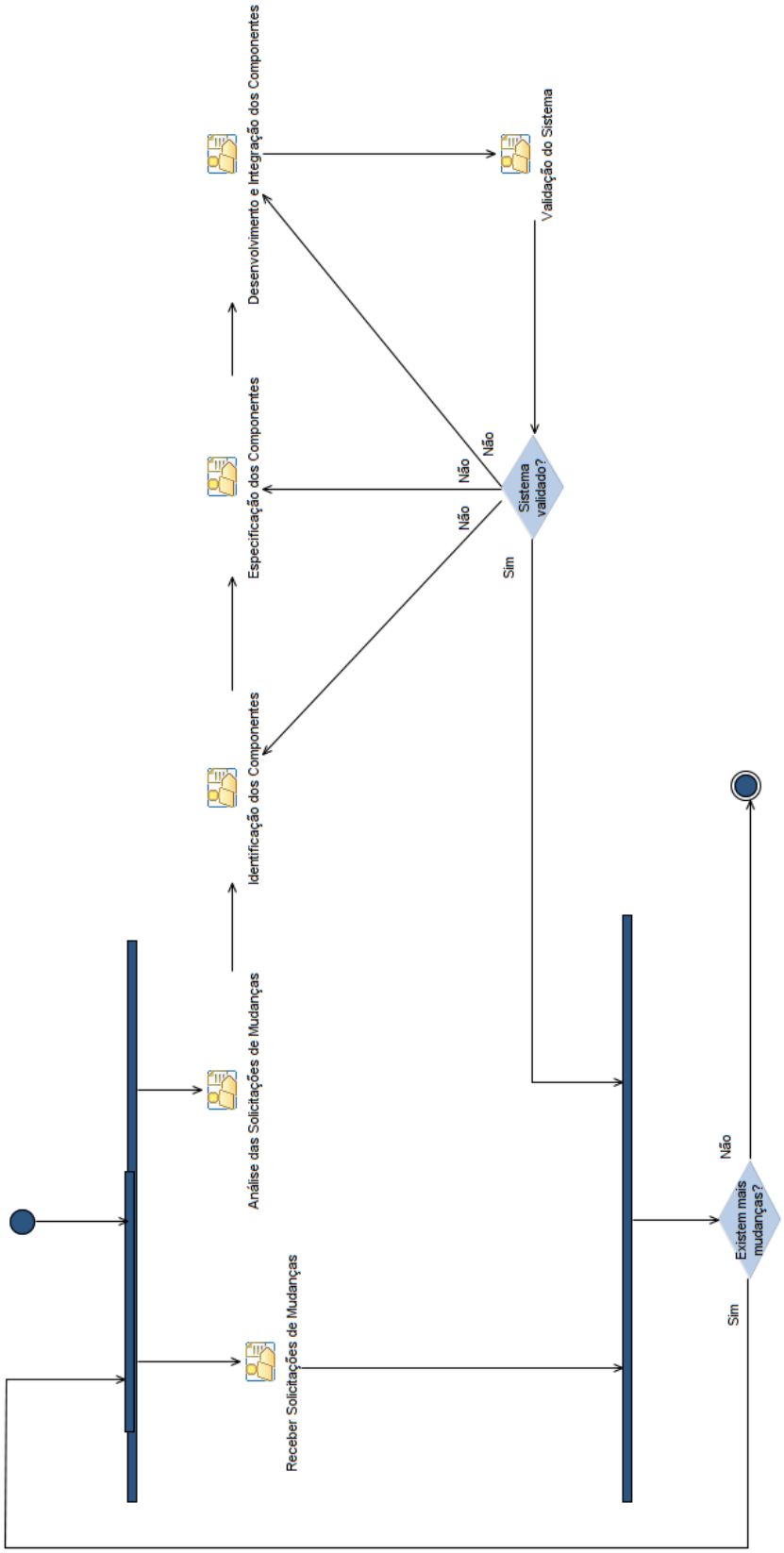

Figura 3.2: Visão geral das atividades do Prometheus.

A Tabela 3.1 mostra as tarefas desempenhadas por cada um dos papeis listados acima e os artefatos por eles alterados no Prometheus. Além disso, as Figuras 3.3, 3.4 e 3.5 são referentes aos papéis "Analista de Sistema", "Analista Funcional" e "Desenvolvedor", respectivamente, e mostram as tarefas e artefatos sob a responsabilidade de cada um desses papéis representados no EPF. Cada uma dessas atividades e tarefas será detalhada nas seções a seguir.

Papel	**Tarefas**	**Artefatos**
Analista de Sistema	Construir Arquitetura Inicial do Componente OA; Definir Modelo de Dados do Componente OA; Detalhar Operações do Componente OA; Especificar Pré e Pós Condições do Componente OA; Identificar Dependencias Entre Componentes; Identificar Partes Afetadas do Sistema; Identificar Pontos de Junção; Selecionar Componente OA para Projeto; e Selecionar Mudança.	Arquitetura dos Componentes; Detalhamento de Operações dos Componentes OA; e Diagrama de Implantação.
Analista Funcional	Analisar Solicitações de Mudança; Construir Artefatos Adicionais; Identificar Artefatos Adicionais; Receber Solicitações de Mudança; e Validar Sistema.	Lista de Artefatos Necessários; Sistema Existente; e Solicitação de Mudança.
Desenvolvedor	Implementar Componente OA; Integrar Componente OA; e Selecionar Componente OA para Implementação.	Componente OA; e Sistema Integrado.

Tabela 3.1: Papéis utilizados no Prometheus e suas responsabilidades.

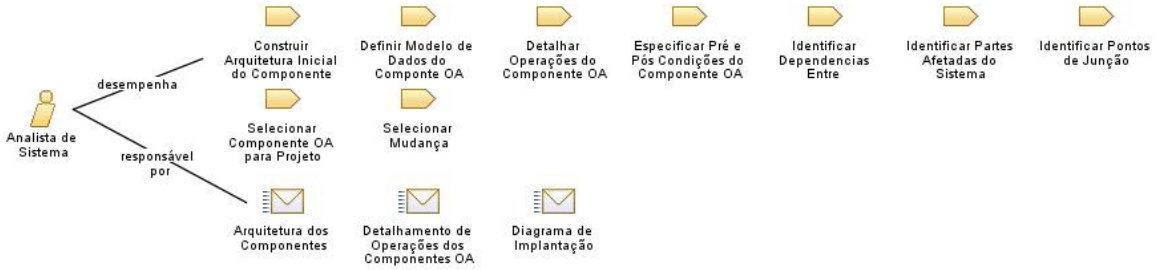

Figura 3.3: Visão geral da participação do Analista de Sistema no Prometheus.

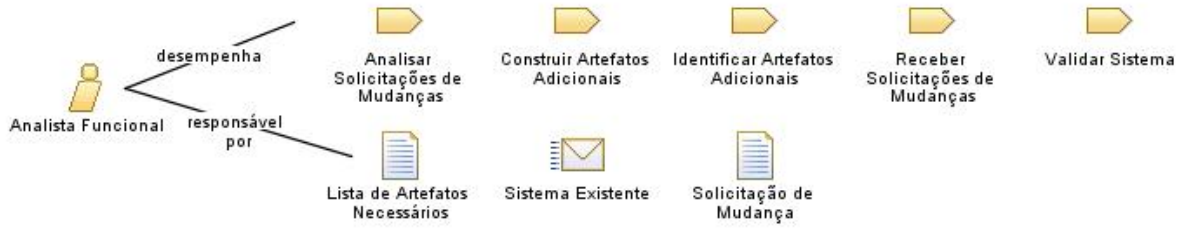

Figura 3.4: Visão geral da participação do Analista Funcional no Prometheus.

Figura 3.5: Visão geral da participação do Desenvolvedor no Prometheus.

3.3 Detalhamento das Atividades

Nesta seção, serão detalhadas as atividades do Prometheus. Cada atividade do Prometheus é composta por uma ou mais tarefas que modificam ou produzem um ou mais produtos de trabalho. Assim, cada atividade será descrita em uma seção separada e os termos importantes do Prometheus serão escritos em **negrito** e em entre "aspas". Para cada uma dessas atividades, será mostrado um diagrama desenhado no EPF detalhando o fluxo das tarefas dessa atividade. Além disso, um outro diagrama irá destacar os papéis que participam da atividade e os produtos de trabalho consumidos e produzidos por cada tarefa dessa atividade.

3.3.1 Atividade "Receber Solicitações de Mudanças"

Em alguns processos de desenvolvimento baseados no Processo Unificado, tais como o RUP, existe uma disciplina chamada de gerência de mudanças que é responsável por receber e documentar as solicitações de mudanças durante o ciclo de desenvolvimento de software (Kruchten, 2004). No contexto da manutenção de software, as solicitações de mudanças é que irão iniciar as atividades de desenvolvimento e irão desencadear uma série de alterações no sistema que vão desde os artefatos de requisitos até o código fonte desse sistema. Na indústria de software, cada empresa possui seus próprios métodos para tratar as mudanças no ciclo de desenvolvimento. No Prometheus, esse cenário foi simplificado

para apenas uma única atividade chamada de **"Receber Solicitações de Mudanças"** e que possui uma única tarefa que também é chamada de **"Receber Solicitações de Mudança"** conforme é ilustrado na Figura 3.6. Essa atividade tem o objetivo de documentar as solicitações de mudança ao longo do ciclo de desenvolvimento do componente OA para que possam ser tratadas na próxima iteração e é por isso que ela deve ocorrer paralelamente às demais atividades do processo conforme é ilustrado na Figura 3.2. O papel responsável pela tarefa de receber as solicitações de mudanças é o **"Analista Funcional"** que irá documentar as mudanças que deverão ser incorporadas ao sistema em um artefato chamado **"Solicitação de Mudança"** como pode ser visto Figura 3.6. Esse artefato tem o objetivo de detalhar quais as funcionalidades do sistema serão afetadas e como esse sistema irá funcionar após as mudanças terem sido implementadas. Na indústria de software, cada empresa possui um padrão para documentar as solicitações de mudanças e elas normalmente são documentadas e armazenadas em sistemas de rastreamento de mudanças, tais como Bugzilla [5], Clear Quest [6] e Rational Team Concert [7], entre outros.

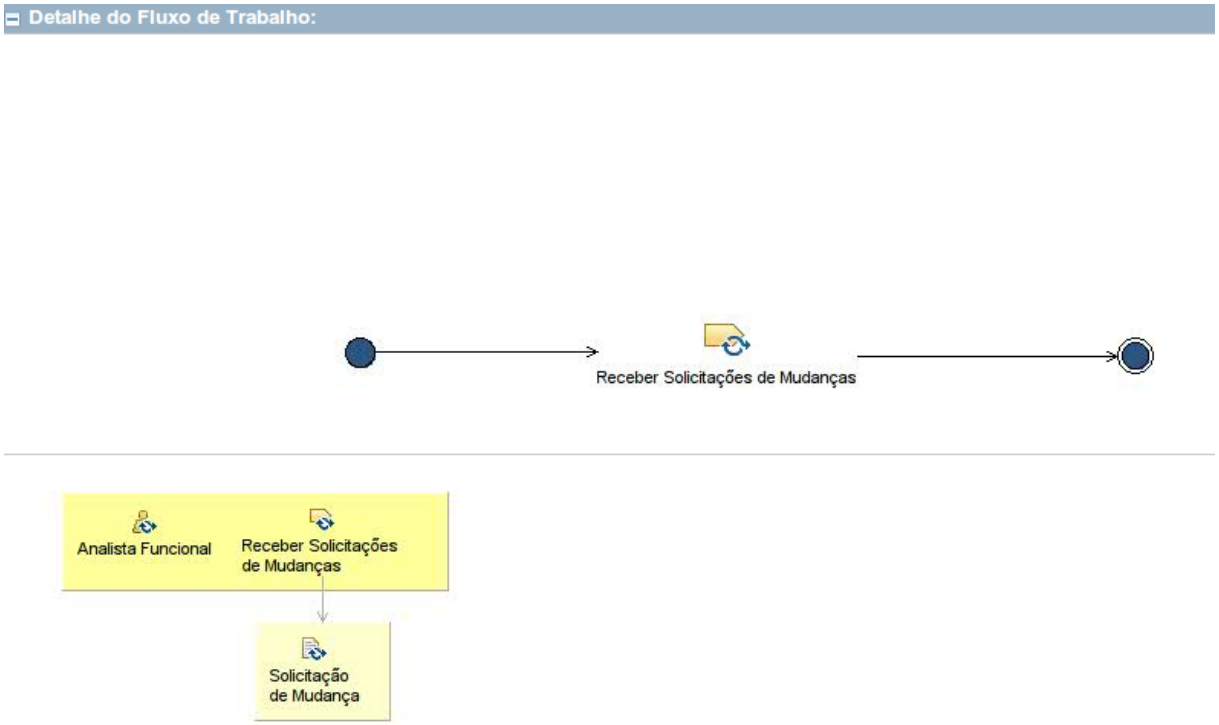

Figura 3.6: Fluxo de tarefas da atividade "Receber Solicitacoes de Mudancas" e artefatos utilizados por cada tarefa representados no EPF.

[5] http://www.bugzilla.org/
[6] http://www-01.ibm.com/software/awdtools/clearquest/
[7] https://jazz.net/products/rational-team-concert/

3.3.2 Atividade "Análise das Solicitações de Mudanças"

Como dito anteriormente, os processos de desenvolvimento baseados no Processo Unificado, tais como o RUP, possuem uma disciplina chamada de gerência de mudanças que é responsável por receber e tratar as mudanças ao longo do ciclo de desenvolvimento de software (Kruchten, 2004). Além disso, essa disciplina também é responsável por analisar o impacto que as mudanças terão em diversos artefatos relacionados ao software, tais como documento, modelos, planos, testes, código fonte, entre outros. No contexto da manutenção de software, essa disciplina tem fundamental importância, pois é responsável por manter os artefatos relacionados ao software atualizados depois que esse software entra em produção. No Prometheus, a análise de impacto foi mapeada em uma atividade chamada de **"Análise das Solicitações de Mudanças"** que tem o objetivo de analisar as solicitações de mudanças e adquirir mais informações sobre o sistema conforme é ilustrado na Figura 3.7. Essa atividade é composta por três tarefas:

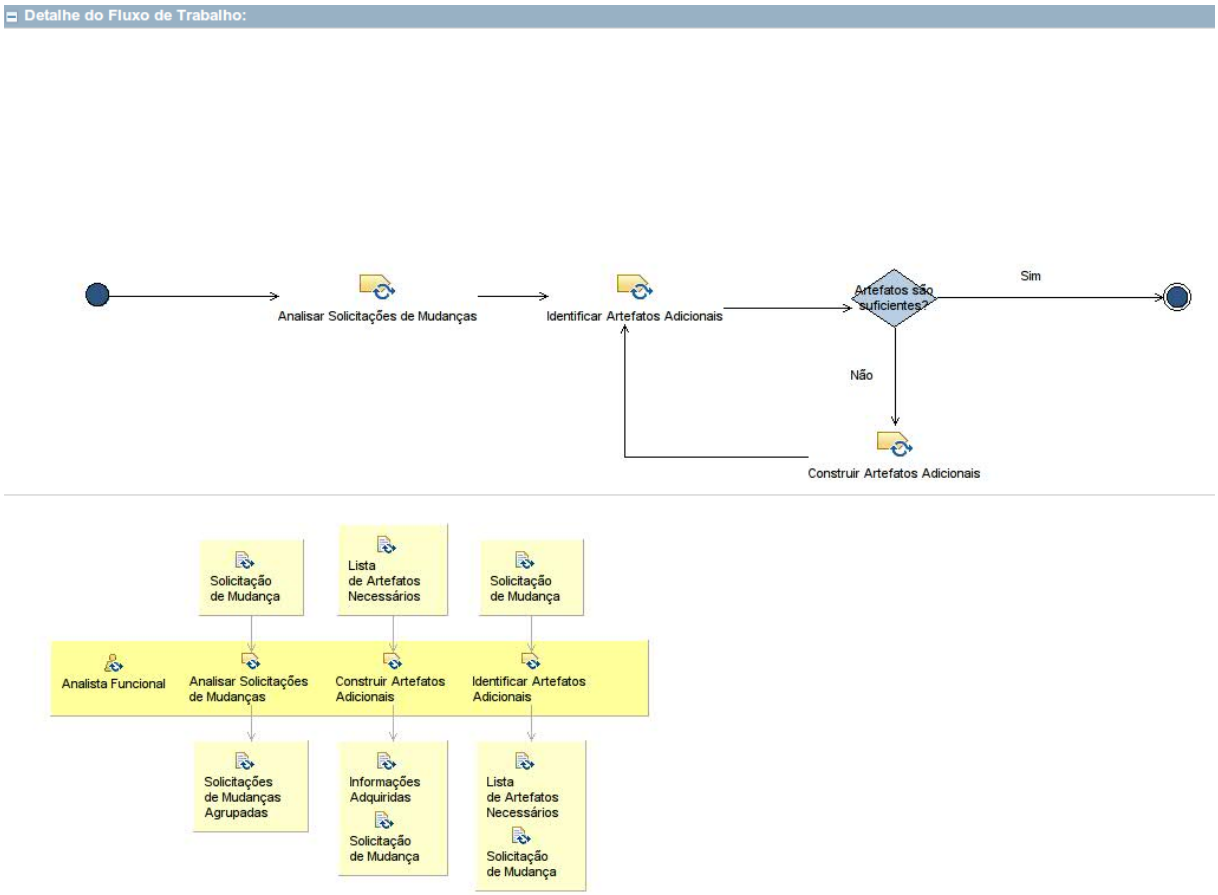

Figura 3.7: Fluxo de tarefas da atividade "Análise das Solicitações de Mudanças" e artefatos utilizados por cada tarefa representados no EPF.

- **"Analisar Solicitações de Mudanças"**: tem o objetivo de agrupar as solicitações de mudanças segundo o critério estabelecido por Chapin et al. (2001) e que foi descrito na Seção 2.2.1 deste trabalho. Esse critério divide os tipos de manutenção de software em 12 tipos específicos agrupando-os em quatro tipos mais gerais. Assim, o objetivo de agrupar as solicitações de mudanças segundo esse critério é o de facilitar a identificação de similaridades por meio da classificação das mudanças em tipos semelhantes de manutenção de software. O papel responsável por essa tarefa é o **"Analista Funcional"** que irá utilizar os artefatos de **"Solicitação de Mudança"** para agrupar as solicitações de mudanças em cada um dos tipos descritos por Chapin et al. (2001). Finalmente, as solicitações de mudanças agrupadas são representadas no Prometheus pelo produto de trabalho **"Solicitações de Mudanças Agrupadas"**;

- **"Identificar Artefatos Adicionais"**: tem o objetivo de identificar os artefatos que serão necessários para desenvolver a mudança. Dentre os artefatos necessários pode-se exemplificar: código fonte das partes afetadas, documentos de requisitos, manuais de usuário, entre outros. É importante ressaltar que em processos de desenvolvimento baseados no Processo Unificado, tais como o EUP e o RUP, é realizada uma análise de impacto com o objetivo de listar os artefatos do sistema que serão afetados pela solicitação de mudança. No Prometheus, a análise de impacto foi omitida, pois não é objetivo do processo detalhar os passos para atualizar os diversos artefatos de um sistema resultantes das atividades de manutenção. Assim, o papel responsável por essa tarefa é o **"Analista Funcional"** que irá utilizar o documento de **"Solicitação de Mudança"** para investigar e listar quais os artefatos que são necessárias para que a mudança seja incoroporada ao sistema. Adicionalmente, o papel **"Analista de Sistema"** pode auxiliar a execução dessa tarefa para identificar os artefatos que envolvam maior conhecimento técnico sobre o sistema e que serão necessários para desenvolver a mudança. Finalmente, a lista de artefatos necessários para desenvolver a mudança será documentada no artefato de **"Solicitação de Mudança"** como é representado no Prometheus pelo produto de trabalho **"Lista de Artefatos Necessários"**; e

- **"Construir Artefatos Adicionais"**: tem o objetivo de construir os artefatos que são necessários para desenvolver a mudança que, por motivos diversos, não existem ou não estão disponíveis. Nesse contexto, encaixam-se os sistemas legados e os sistemas comerciais (Pressman, 2010), nos quais a documentação do sistema foi perdida ao longo do tempo ou não é acessível pela empresa que o comprou. Especialmente no caso dos sistemas comerciais, a alteração não é feita pela empresa que o desenvolveu,

por isso existe uma carência de documentos e modelos que auxiliem na compreesão de quais partes serão afetadas pela mudança (Nah et al., 2001). Portanto, é comum serem utilizadas técnicas de Engenharia Reversa com o objetivo de recuperar informações a respeito do sistema (Pressman, 2010). No Prometheus, não é proposto nenhuma técnica de Engenharia Reversa, pois não é objtivo do processo detalhar como as informações do sistema serão recuperadas ou como os artefatos serão construídos. Assim, o papel responsável por essa tarefa é o **"Analista Funcional"** que irá utilizar o produto de trabalho **"Lista de Artefatos Necessários"** com o objetivo de construir os artefatos necessários para que a mudança seja realizada. Adicionalmente, o papel **"Analista de Sistema"** pode auxiliar a execução dessa tarefa para recuperar informações a partir do código fonte do sistema ou construir artefatos que necessitem de maior conhecimento técnico, tais como diagramas UML (OMG, 2007). Finalmente, as informações adquiridas para realizar a mudança serão documentadas no artefato de **"Solicitação de Mudança"** como é representado no Prometheus pelo produto de trabalho **"Informações Adquiridas".**

3.3.3 Atividade "Identificação dos Componentes"

O propósito da identificação de componentes é definir um conjunto inicial das interfaces e das especificações de um componente com o objetivo de criar uma visão geral da solução que será refinada e detalhada mais adiante (Cheesman e Daniels, 2001; Clemente et al., 2002b; Grundy e Patel, 2001). Os objetivos principais dessa atividade são: (i) descobrir quais informações precisam ser gerenciadas; (ii) quais intefaces são necessárias para gerenciar essas informações; (iii) quais componentes são necessários para provêr as funcionalidades; e (iv) como esses componentes serão organizados em conjunto. No Prometheus, a identificação de componentes foi mapeada em uma atividade chamada de **"Identificação dos Componentes"** que, além de identificar os componentes que serão desenvolvidos, tem o objetivo de identificar as partes do sistema que serão afetadas por esses componentes. Esses componentes irão se relacionar com o sistema utilizando o paradigma de orientação a aspectos. Mais especificamente, a interface de cada componente será mapeada em um aspecto que irá entrecortar o sistema. Assim, os componentes identificados por essa atividade são denominados de **componentes orientados a aspectos**, ou simplesmente, **componentes OA**. Finalmente, essa atividade é composta por cinco tarefas conforme é ilustrado na Figura 3.8:

- **"Selecionar Mudança"**: tem o objetivo de selecionar qual mudança será desenvolvida. Nessa tarefa as solicitações de mudanças já se encontram classificadas segundo o critério proposto por Chapin et al. (2001), uma vez que essa classificação

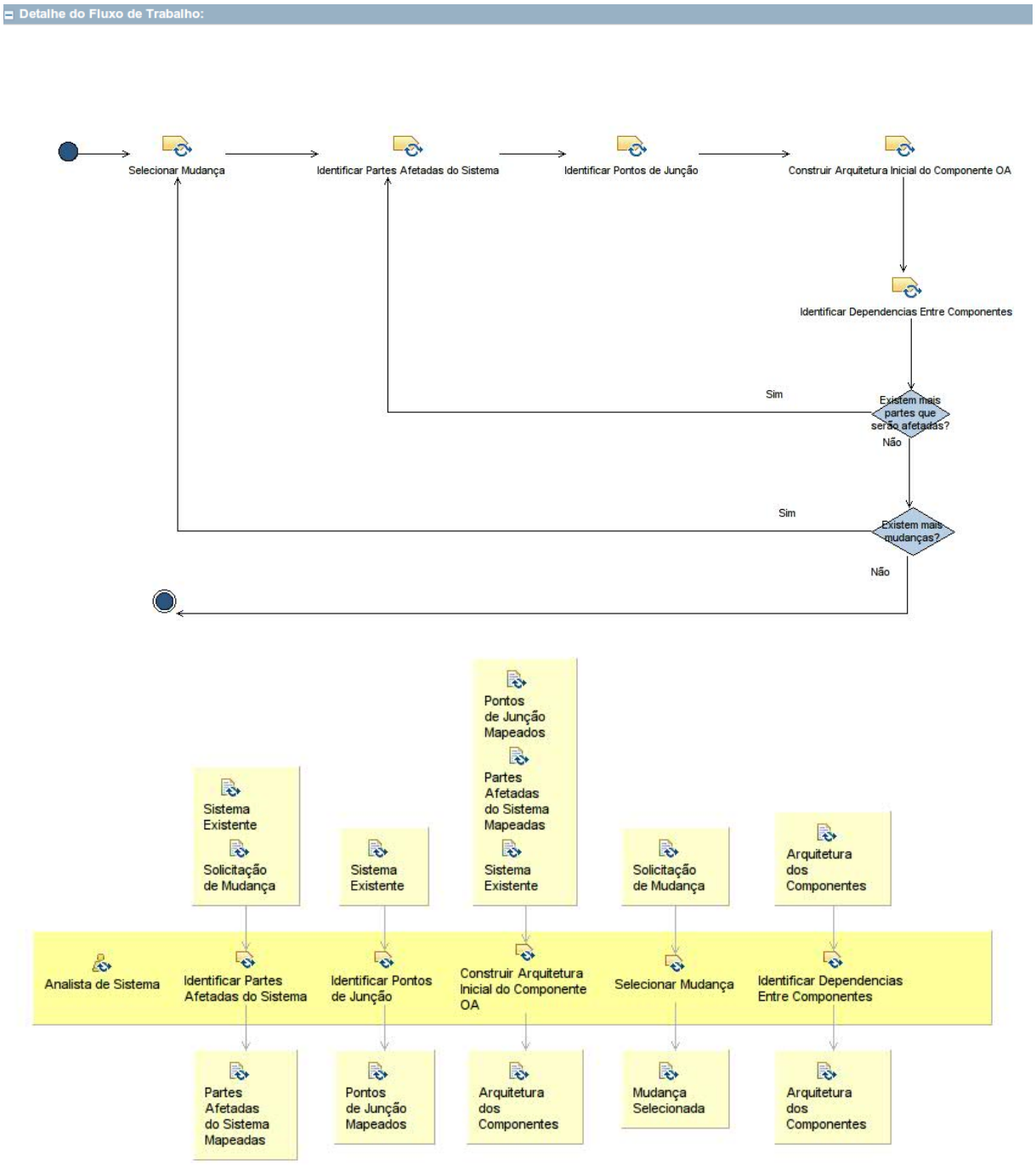

Figura 3.8: Fluxo de tarefas da atividade "Identificação dos Componentes" e artefatos utilizados por cada tarefa representados no EPF.

foi realizada na atividade **"Análise das Solicitações de Mudanças"**. Essa classificação serve para guiar a escolha da solicitação de mudança que será projetada, pois segundo o tipo de manutenção ao qual ela pertence pode-se ter uma idéia das dificuldades técnicas para que essa mudança seja incorporada ao sistema e priorizar o seu desenvolvimento adequadamente. Assim, no Prometheus o papel responsável por essa tarefa é o **"Analista de Sistema"** e o produto de trabalho **"Mudança**

Selecionada" é utilizado para representar que uma determinada solicitação de mudança foi escolhida para ser considerada nessa atividade;

- **"Identificar Partes Afetadas do Sistema"**: tem como objetivo identificar as partes do sistema que serão afetadas pelo componente OA. Mais especificamente, essa tarefa prevê que sejam identificados os trechos do código do sistema que serão afetados por essa mudança. É importante ressaltar que no contexto da customização de sistemas de software comerciais, os profissionais responsáveis por implantar uma determinada solicitação de mudança possuem conhecimento prévio de potencias partes do sistema que serão afetadas por essa mudança (Nah et al., 2001). Assim, o papel responsável por essa tarefa é o **"Analista de Sistema"** que irá utilizar como artefatos o **"Sistema Existente"** e o documento de **"Solicitação de Mudança"** para identificar as partes do sistema que serão afetadas pela solicitação de mudança e que são representadas no Prometheus pelo produto de trabalho **"Partes Afetadas do Sistema Mapeadas"**;

- **"Identificar Pontos de Junção"**: tem como objetivo definir os pontos de junção que serão utilizados por um ou mais aspectos definidos nos componentes OA para entrecortar o sistema. No Prometheus, o papel responsável por essa tarefa é o **"Analista de Sistema"** que irá utilizar produto de trabalho **"Sistema Existente"** com o objetivo de definir os pontos de junção de cada componente OA e que são representados no Prometheus pelo produto de trabalho **"Pontos de Junção Mapeados"**. É importante ressaltar que, no contexto da customização de sistemas de grande porte, é muito comum os profissionais responsáveis por customizar um determinado sistema realizarem alguma técnica de Engenharia Reveresa (Pressman, 2010) com o intuito de descobrir o local exato que precisa ser alterado nesse sistema (Coady et al., 2001; Nah et al., 2001). No Prometheus, entende-se que essa atividade será utilizada para identificar os pontos de junção dos componentes OA que serão desenvolvidos;

- **"Construir Arquitetura Inicial do Componente OA"**: tem como objetivo definir quais os componentes OA que serão necessários para atender a uma determinada solicitação de mudança solicitação de mudança. Esses componentes devem ser identificados de maneira que suas interfaces sejam expressas pelos aspectos e pontos de junção identificados anteriormente. No Prometheus, o papel responsável por essa tarefa é o **"Analista de Sistema"** que irá utilizar os produtos de trabalho **"Pontos de Junção Mapeados"**, **"Partes Afetadas do Sistema Mapeadas"** e **"Sistema Existente"** com o objetivo de construir um diagrama de componentes (OMG, 2007) que é representado no Prometheus pelo produto de trabalho "**Arquitetura**

dos Componentes". Esse diagrama deve representar os elementos da POA utilizados na modelagem dos componentes OA, tais como métodos interceptados pelo componente OA e pontos de junção identificados. Nesse contexto, é importante ressaltar que existem diversas formas de representar os elementos da POA em diagramas UML, não existindo, portanto, um consenso sobre qual a melhor maneira de realizar essa representação (Guessi et al., 2011); e

- **"Identificar Dependências Entre Componentes"**: tem como objetivo atualizar o diagrama de componentes definido na tarefa anterior para apontar as dependências entre os componentes considerados, se existirem. É importante ressaltar que as dependências entre os componentes OA e o sistema que será customizado não devem ser representadas nesse diagrama, pois essas dependências serão consideradas mais adiante com mais detalhes. Assim, no Prometheus o papel responsável por essa tarefa é o **"Analista de Sistema"** que irá atualizar o diagrama de componentes representado no Prometheus pelo produto de trabalho "**Arquitetura dos Componentes"**.

3.3.4 Atividade "Especificação dos Componentes"

O objetivo da especificação de componentes é especificar as operações e interfaces dos componentes (Cheesman e Daniels, 2001; Clemente et al., 2002b; Grundy e Patel, 2001). No Prometheus, a especificação de componentes foi mapeada em uma atividade chamada de **"Especificação dos Componentes"** que, além de especificar as operações de cada componente OA, tem o objetivo de explicitar as dependências desses componentes com o sistema que será customizado. Assim, essa atividade é composta por quatro tarefas conforme é ilustrado na Figura 3.9:

- **"Selecionar Componente OA para Projeto"**: tem o objetivo de selecionar qual o componente OA será projetado. Nessa tarefa, os componentes OA identificados na atividade **"Identificação dos Componentes"** serão selecionados com base na classificação realizada na atividade **"Análise das Solicitações de Mudanças"**. Além disso, na atividade **"Identificação dos Componentes"** também foram analisadas as partes afetadas do sistema por cada componente OA e essas informações podem ser utilizadas para selecionar o componente OA que será projetado. Assim, no Prometheus o papel responsável por essa tarefa é o **"Analista de Sistema"** que vai utilizar o produto de trabalho **"Arquitetura dos Componentes"** com o objetivo de analisar as informações contidas nesse produto de trabalho para selecionar o componente OA que será projetado. Finalmente, o resultado dessa seleção é repre-

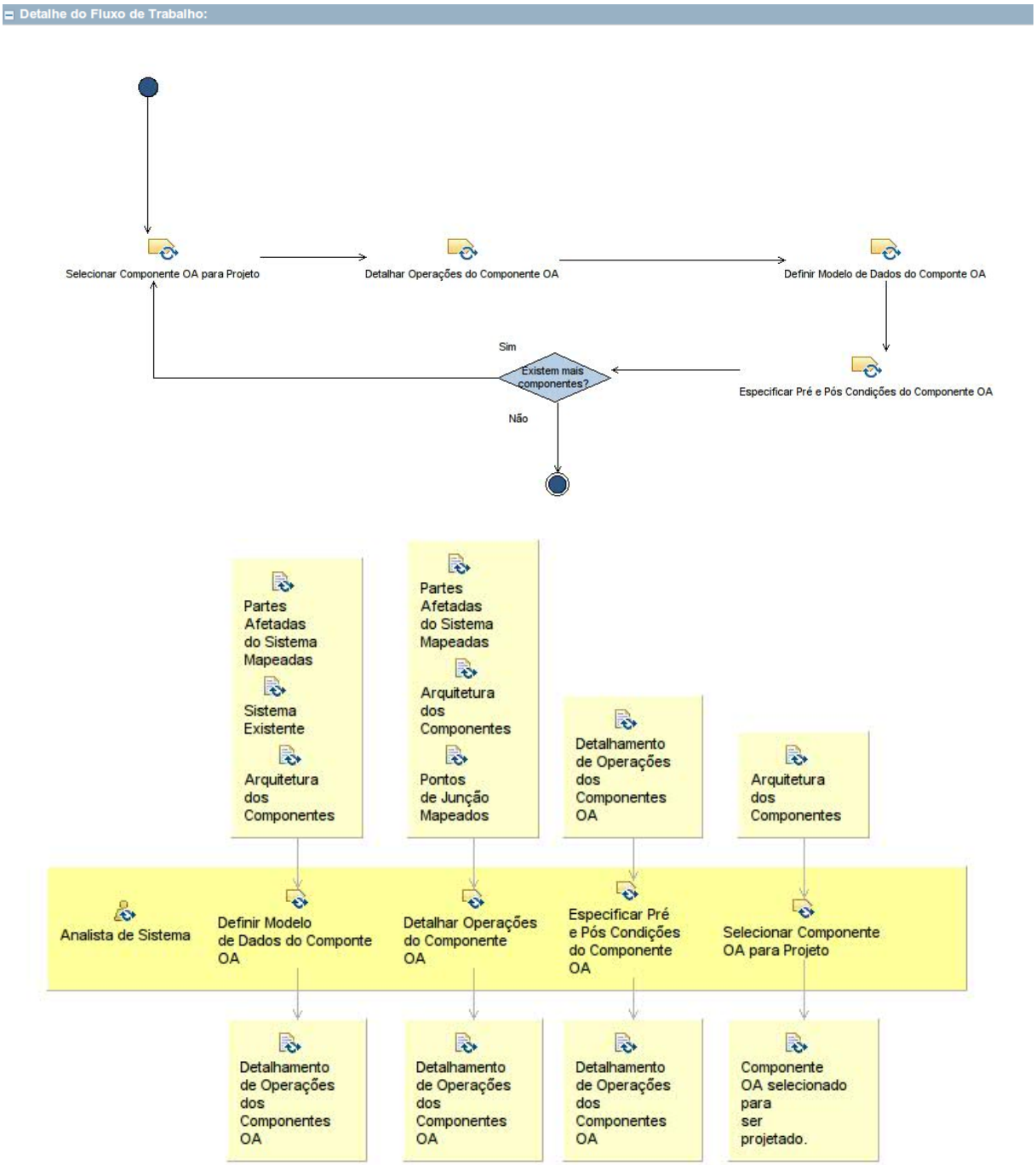

Figura 3.9: Fluxo de tarefas da atividade "Especificação dos Componentes" e artefatos utilizados por cada tarefa representados no EPF.

sentado no Prometheus pelo produto de trabalho **"Componente OA selecionado para ser projetado"**;

- **"Detalhar Operações do Componente OA"**: tem o objetivo de definir as operações que irão compor o componente OA . Nessa tarefa, todos os elementos da POA envolvidos no projeto do componente OA devem estar detalhados: adendos,

pontos de junção e aspectos. No Prometheus, o papel responsável por essa tarefa é o **"Analista de Sistema"** que irá utilizar os produtos de trabalho **"Partes Afetadas do Sistema Mapeadas"**, **"Arquitetura dos Componentes"** e **"Pontos de Junção Mapeados"** para construir um diagrama de classes (OMG, 2007) que é representado no Prometheus pelo produto de trabalho **"Detalhamento de Operações dos Componentes OA"**. Esse diagrama de classes deve conter a definição de todos os elementos do componente OA: classes, métodos, adendos, pontos de junção, aspectos e associações. Nesse contexto, é importante ressaltar que existem diversas formas de representar os elementos da POA em diagramas UML, não existindo, portanto, um consenso sobre qual a melhor maneira de realizar essa representação (Guessi et al., 2011);

- **"Definir Modelo de Dados do Componente OA"**: tem o objetivo de refinar o diagrama de classes construído na tarefa anterior para indicar as dependências de cada elemento desse diagrama com o sistema que será customizado. No Prometheus, o papel responsável por essa tarefa é o **"Analista de Sistema"** que vai utilizar os produtos de trabalho **"Partes Afetadas do Sistema Mapeadas"**, **"Sistema Existente"** e **"Arquitetura dos Componentes"** para indicar no diagrama de classes as dependências de cada elemento com o sistema que será customizado. Por fim, o diagrama de classes atualizado é representado no Prometheus pelo produto de trabalho **"Detalhamento de Operações dos Componentes OA"**;

- **"Especificar Pré e Pós Condições do Componente OA"**: tem o objetivo de especificar as pré e pós condições necessárias para a execução de cada interface. Nessa tarefa, as pré e pós condições para execução de cada adendo devem ser detalhadas, no entanto, isso só é necessário se esse adendo modificar as pré e pós condições dos métodos que ele interceptar, pois, do contrário, essa informação torna-se irrelevante para a customização do sistema. Assim, no Prometheus, o papel responsável por essa tarefa é o **"Analista de Sistema"** que vai atualizar, se necessário, o diagrama de classes representado no Prometheus pelo produto de trabalho **"Detalhamento de Operações dos Componentes OA"**.

3.3.5 Atividade "Desenvolvimento e Integração dos Componentes"

O objetivo do desenvolvimento e integração dos componentes é implementar os componentes especificados e integrá-los ao sistema (Cheesman e Daniels, 2001; Clemente et al., 2002b; Grundy e Patel, 2001). No Prometheus, o desenvolvimento e a integração dos componentes é mapeado em uma atividade chamada de **"Desenvolvimento e Integração dos Componentes"** que tem como objetivo implementar os componentes OA projetados

nas atividades anteriores e integrá-los ao sistema que será customizado. Portanto, essa atividade é composta por três tarefas conforme é ilustrado na Figura 3.10:

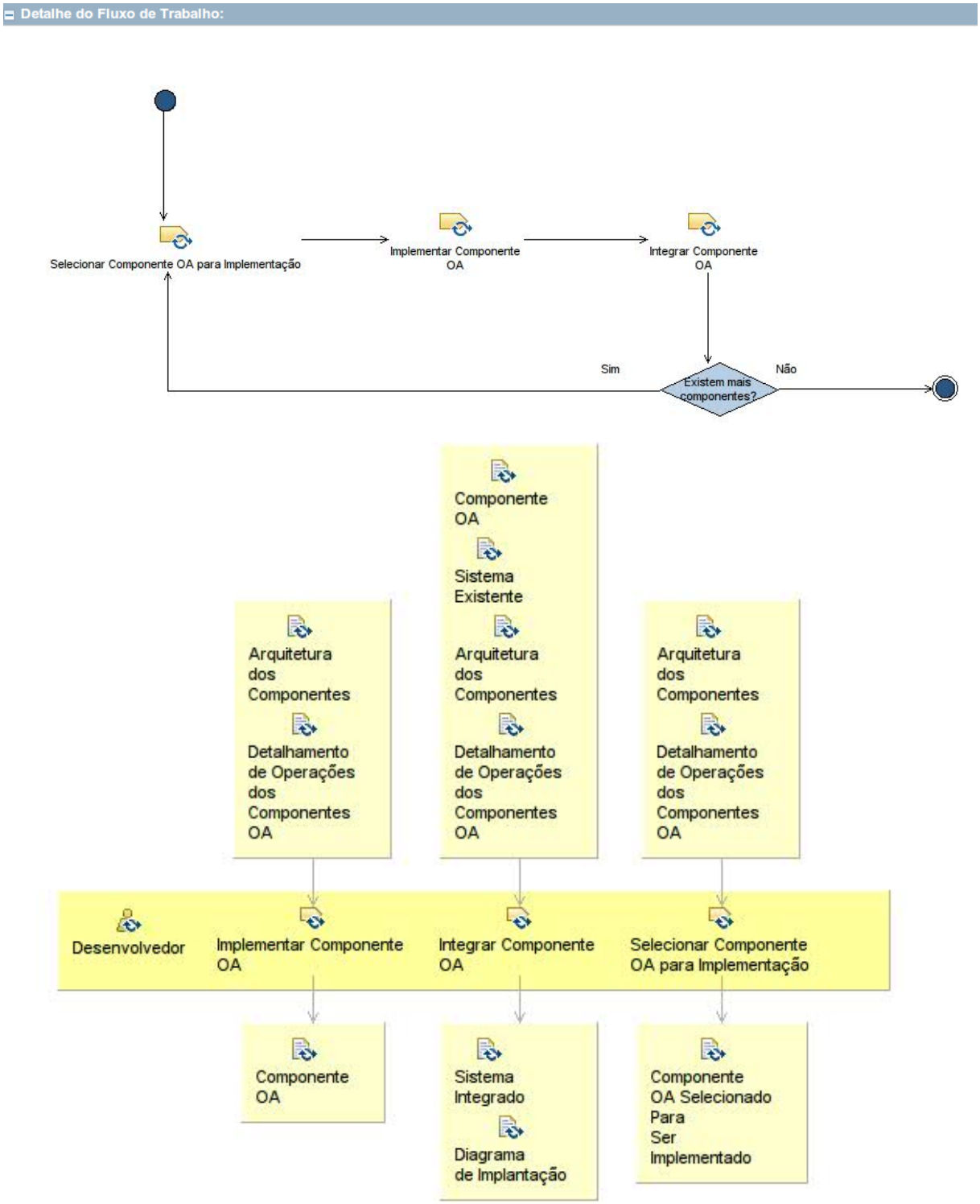

Figura 3.10: Fluxo de tarefas da atividade "Desenvolvimento e Integração dos Componentes" e artefatos utilizados por cada tarefa representados no EPF.

- **"Selecionar Componente OA para Implementação"**: tem o objetivo de selecionar qual o componente OA será implementado. Nessa tarefa, os componentes OA projetados na atividade **"Especificação dos Componentes"** serão selecionados com base na classificação realizada na atividade **"Análise das Solicitações de Mudanças"** e nas informações sobre as partes afetadas no sistema adquiridas na atividade **"Identificação dos Componentes"**. Além disso, na atividade "Especificação dos Componentes" foi construído um diagrama de classes contendo as dependências dos componentes OA com o sistema que será customizado e que será utilizado como fonte de informação nessa tarefa. Assim, no Prometheus o papel responsável por essa tarefa é o **"Desenvolvedor"** que vai utilizar os produtos de trabalho **"Arquitetura dos Componentes"** e **"Detalhamento de Operações dos Componentes OA"** com o objetivo de analisar as informações contidas nesses produtos de trabalho para selecionar o componente OA que será implementado. Finalmente, o resultado dessa seleção é representado no Prometheus pelo produto de trabalho **"Componente OA Selecionado Para Ser Implementado"**;

- **"Implementar Componente OA"**: tem o objetivo de implementar o componente OA. Nessa tarefa, o componente OA será implementado de acordo com as tecnologias utilizadas pelo sistema que será customizado. É importante ressaltar que existem diversas tecnologias disponíveis para implementar componentes OA tais como AspectWerkz (Bonér e Vasseur, 2005), JAsCO (Suvée, 2005), AspectJ (Kiczales et al., 2001b), AspectC++(Gal et al., 2001), AspectC(Coady et al., 2001), entre outras. Dessa maneira, o Prometheus pode ser aplicado em uma grande quantidade de sistemas. Assim, no Prometheus o papel responsável por essa tarefa é o **"Desenvolvedor"** que vai utilizar como fonte de informação os diagramas de componentes e de classes representados no Prometheus pelos produtos de trabalho **"Arquitetura dos Componentes"** e **"Detalhamento de Operações dos Componentes OA"**, respectivamente. Por fim, no Prometheus o produto de trabalho **"Componente OA"** representa a conclusão dessa tarefa; e

- **"Integrar Componente OA"**: tem o objetivo de integrar o componente OA ao sistema. Nessa tarefa, o componente OA será integrado ao sistema com o objetivo de customizar esse sistema conforme previsto nas solicitações de mudanças. No Prometheus, o papel no responsável por essa tarefa é o **"Desenvolvedor"** que vai utilizar como fonte de informação os diagramas de componentes e de classes representados no Prometheus pelos produtos de trabalho **"Arquitetura dos Componentes"** e **"Detalhamento de Operações dos Componentes OA"**, respectivamente. Além disso, também serão utilizados o componente OA e o sistema que será

customizado os quais são representados no Prometheus pelos produtos de trabalho **"Componente OA"** e **"Sistema Existente"**, respectivamente. Dessa maneira, todas essas informações serão utilizadas para construir um diagrama de implantação (OMG, 2007) com o objetivo de representar como o componente OA será integrado ao sistema. Esse diagrama é representado no Prometheus pelo produto de trabalho **"Diagrama de Implantação"**. Nesse contexto, é importante ressaltar que o modo como o componente OA será integrado ao sistema deve seguir os padrões e tecnologias utilizados por esse sistema. Finalmente, após concluída essa tarefa o sistema encontra-se customizado e é representado no Prometheus pelo produto de trabalho **"Sistema Integrado"**.

3.3.6 Atividade "Validação do Sistema"

O objetivo principal da verificação de um sistema é avaliar se os requisitos inicialmente concebidos encontram-se corretamente implementados nesse sistema (Pressman, 2010). Nesse contexto, o objetivo é realizar testes funcionais para verificar se as entradas e saídas do sistema são coerentes com os requisitos propostos para esse sistema (Delamaro et al., 2007). No Prometheus, a necessidade de validar a implementação no sistema customizado é expressa pela atividade **"Validação do Sistema"** que possui uma única tarefa chamada de **"Validar Sistema"** conforme é mostrado na Figura 3.11. Essa atividade tem o objetivo de avaliar se os requisitos contidos nas solicitações de mudanças estão corretamente implementados no sistema customizado. Nesse caso, a proposta do Prometheus é que sejam realizados testes funcionais para verificar se o sistema foi customizado como descrito nas solicitações de mudanças. Assim, o papel responsável pela execução da tarefa de validar o sistema é o **"Analista Funcional"** que irá verificar o sistema customizado utilizando cada artefato de **"Solicitação de Mudança"** com o objetivo de validar as customizações realizadas nesse sistema. Finalmente, após finalizados os testes, o sistema encontra-se validado e no Prometheus é representado pelo produto de trabalho **"Sistema Validado"**. No entanto, se algum problema fôr encontrado por essa tarefa, o Prometheus prevê que o processo retorne a uma dessa três atividades: **"Desenvolvimento e Integração dos Componentes"**, **"Identificação dos Componentes"** e **"Especificação dos Componentes"**. Esse desvio pode ser visto na visão geral do processo, conforme foi ilustrado na Figura 3.2, e é importante, pois contempla eventuais retrabalhos decorrentes de defeitos encontrados nos testes funcionais. Finalmente, é importante ressaltar que o Prometheus não propõe nenhuma técnica ou método de testes, pois existem diversas técnicas e métodos que podem ser utilizados para validar um sistema (Sommervile, 2007) e, além disso, é comum que cada empresa aplique as técnicas e métodos de testes que costuma utilizar.

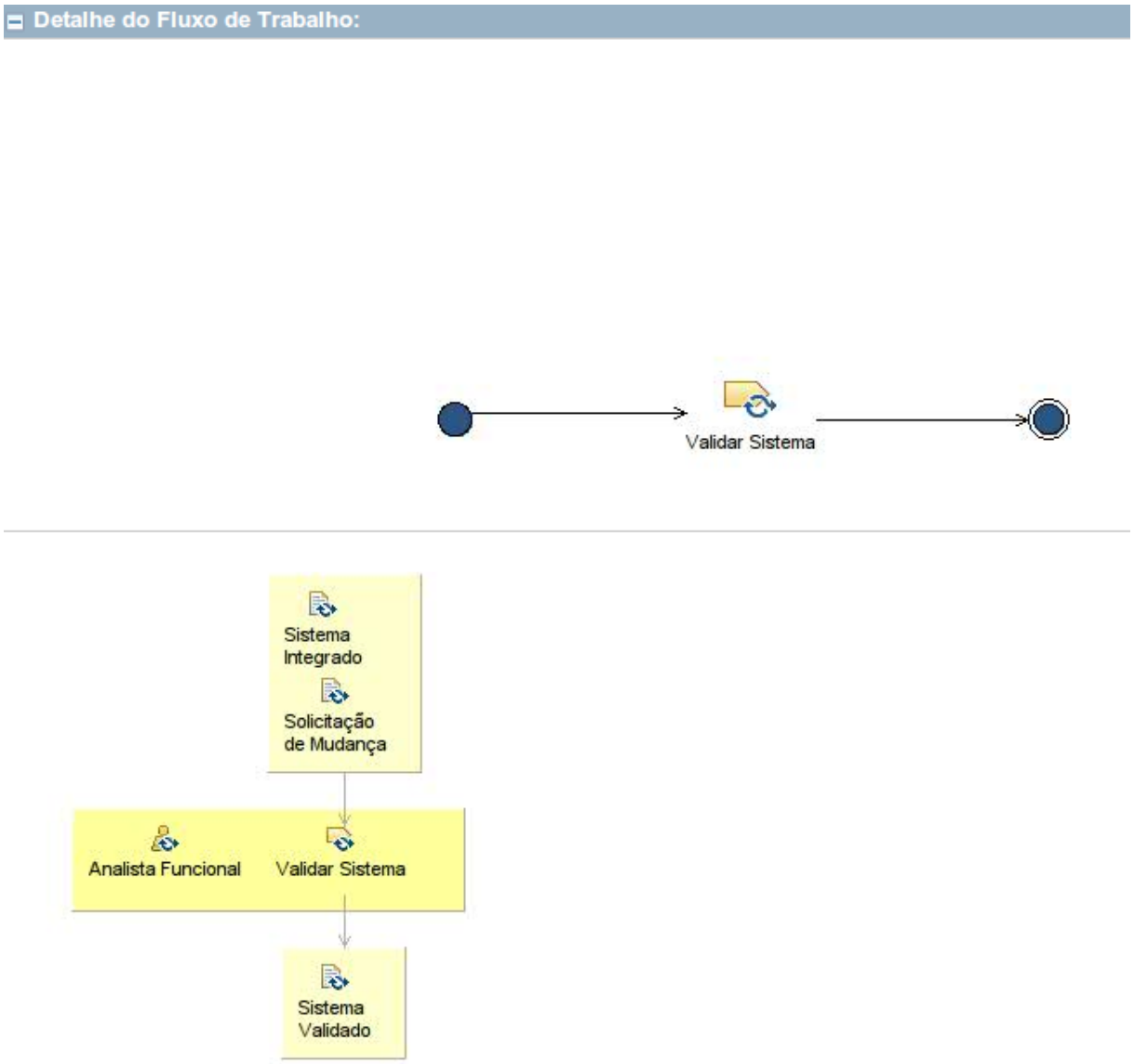

Figura 3.11: Fluxo de tarefas da atividade "Validação do Sistema" e artefatos utilizados por cada tarefa representados no EPF.

3.4 Considerações Finais

Neste capítulo foi apresentado o Prometheus que é um processo para a customização de sistemas de software utilizando componentes OA. Esse processo é constituído de seis atividades que são compostas de uma ou mais tarefas e que têm o objetivo de sistematizar a utilização de componentes OA na customização de sistemas de software. Primeiramente, foi dada uma visão geral do Prometheus na qual foram explicados os conceitos principais necessários para a sua compreensão. A seguir, foram descritas cada uma das atividades do Prometheus contextualizando-as no cenário da customização de sistemas de software. Nesse contexto, é importante ressaltar a utilização da ferramenta EPF que facilitou a

organização e a apresentação dos principais conceitos envolvidos no Prometheus. Por fim, com o objetivo de avaliar a aplicação do Prometheus, no próximo capítulo, será apresentado um estudo de caso no qual o Prometheus é utilizado na customização de um sistema de software.

Capítulo

4

Estudo de Caso: Aplicação do Prometheus

4.1 Considerações Iniciais

COM o objetivo de avaliar a aplicação do Prometheus, neste capítulo é apresentado um estudo de caso na customização do sistema IBM Maximo[1]. Esse sistema é um tipo específico de ERP chamado de EAM (do inglês, *Enterprise Asset Management*) (Brower et al., 2008) que se caracteriza por ser um sistema de software concebido para auxiliar a gestão de equipamentos de grande porte, tais como caminhões, turbinas, tubulações, entre outros. Tal qual os demais ERP's, o sistema IBM Maximo precisa ser customizado para que possa atender às necessidades das empresas que o utilizam (Nah et al., 2001). É nesse contexto que a IBM customizou o sistema IBM Maximo para a *British Petroleum*[2] com o objetivo de tornar o sistema aderente às necessidades dessa empresa. Dessa maneira, os dados dessa customização foram utilizados como base nesse estudo de caso com o objetivo de avaliar os benefícios da aplicação do Prometheus em relação à solução desenvolvida pela IBM. Por fim, acredita-se esse estudo de caso seja suficiente para mostrar a viabilidade da utilização do Prometheus em outros contextos.

[1] http://www-01.ibm.com/software/tivoli/products/maximo-asset-mgmt/ acessado em 17/04/2013
[2] http://www.bp.com/bodycopyarticle.do?categoryId=1&contentId=7052055 acessado em 17/04/2013

Na Seção 4.2 é dada uma visão geral do sistema IBM Maximo para contextualizá-lo com este trabalho. Na Seção 4.3 é descrita a aplicação do Prometheus para customizar o sistema IBM Maximo na qual são mostrados todos os artefatos construídos nesse estudo de caso. A seguir, na Seção 4.4, é feita uma análise dos resultados obtidos nesse estudo de caso. Finalmente, na Seção 4.5 são apresentadas as considerações finais deste capítulo.

4.2 Visão Geral do Sistema Maximo

O sistema escolhido para aplicar o Prometheus foi o IBM Maximo Asset Management, ou simplesmente IBM Maximo, que é um sistema de gerenciamento de ativos (do inglês, *Enterprise Asset Management* ou simplesmente EAM) (Brower et al., 2008) com funcionalidades que auxiliam diversas atividades operacionais de uma indústria, tais como, gerenciamento de manutenção, gerenciamento de estoque, gerenciamento de compras, entre outros. Por isso, o sistema IBM Maximo pode ser considerado um EAM, pois permite que equipamentos valiosos para a operação de uma corporação sejam gerenciados com o objetivo de melhorar sua utilização e aumentar sua performance (Redbooks, 2008).

O sistema IBM Maximo é desenvolvido na plataforma JEE(SUN Microsystems, 2009) e possui uma arquitetura orientada a serviços (do inglês, *Service Oriented Architecture*, ou simplesmente SOA) (Erl, 2008) que permite trocar informações com diversos outros sistemas utilizando *web services (Erl, 2004)* como interface de comunicação. Além disso, o sistema IBM Maximo é baseado em um framework que abstrai as operações que são compartilhadas entre as várias partes do sistema. Do ponto de vista funcional, o sistema IBM Maximo possui diversas aplicações como é mostrado na Figura 4.1. Cada aplicação contém um conjunto de abas e diversos campos relacionadas a um assunto específico como é mostrado na Figura 4.2, na qual pode ser vista a tela principal da aplicação de "Ativos" (do inglês, *Assets*). Essas aplicações são divididas em diversos modulos dentre os quais pode-se destacar:

- Ativos: conjunto de aplicações que auxiliam no gerenciamento e configuração de equipamentos;

- Ordem de Serviço: conjunto de aplicações utilizadas para planejar, mensurar e acompanhar as atividades operacionais;

- Serviço: conjunto de aplicações utilizadas para gerenciar incidentes;

- Contrato: conjunto de aplicações utilizadas para gerenciar contratos e cadastro de fornecedores;

- Materiais: conjunto de aplicações utilizadas para realizar controle de estoque e inventário de materiais utilizados nas atividades operacionais; e

- Gestão de Compras: conjunto de aplicações utilizadas para gerenciar as atividades de compra de equipamentos e materiais.

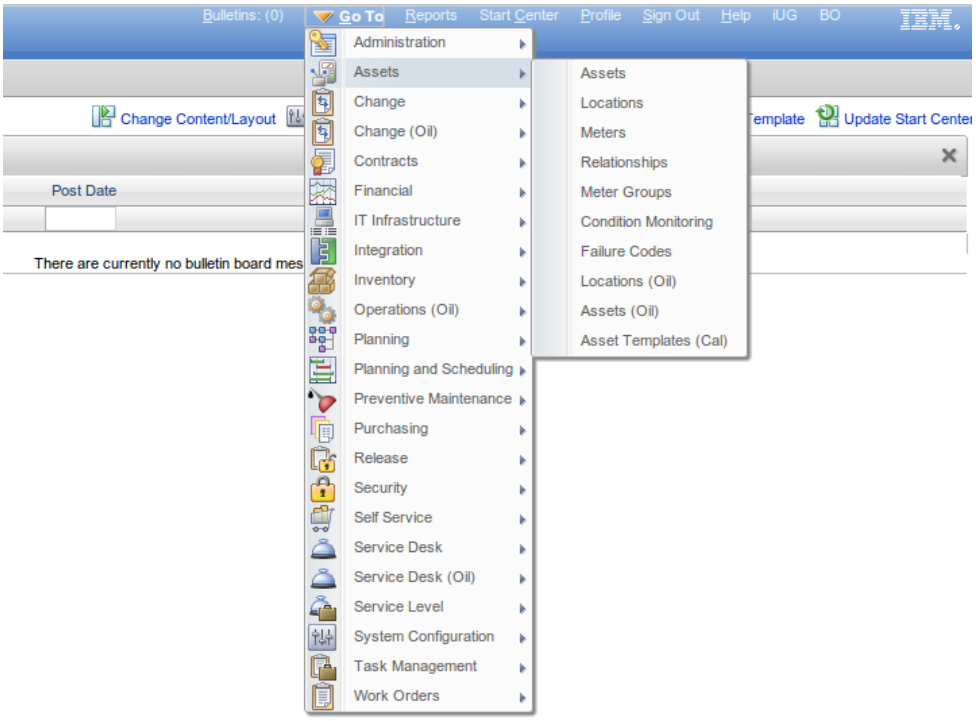

Figura 4.1: Visão geral do menu de aplicações do sistema IBM Maximo.

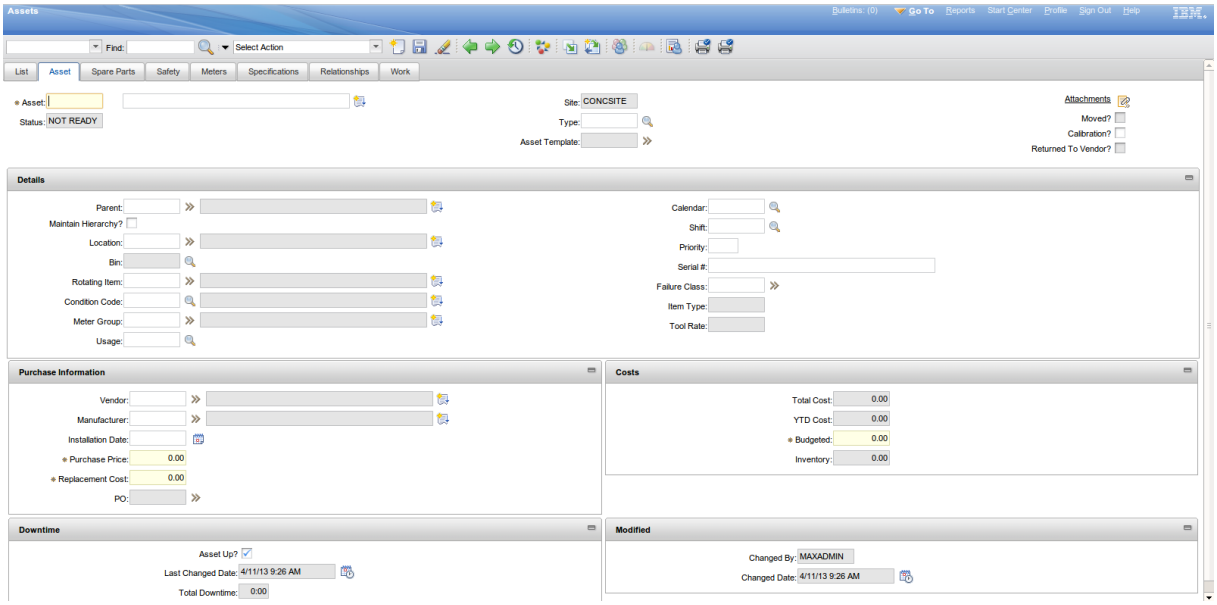

Figura 4.2: Parte da tela do sistema IBM Maximo mostrando a aplicação de Ativos.

Além de possuir diversas aplicações que auxiliam nas atividades operacionais de uma indústria, o sistema IBM Maximo possui também diversas especializações para segmentos específicos da indústria e que podem ser utilizados em conjunto com o objetivo de combinar suas funcionalidades. Dentre essas especializações pode-se destacar:

- Indústria Petrolífera e de Gás Natural[3]: possui aplicações específicas para o ramo de exploração de petróleo e extração de gás natural;
- Indústria de Transportes[4]: possui aplicações específicas para gerenciar equipamentos utilizados pela indústria de transportes, tais como ônibus, caminhões, entre outros; e
- Indústria Hospitalar[5]: possui aplicações específicas para a gestão de equipamentos e atividades hospitalres.

Assim, é possível dizer que o sistema IBM Maximo pode ser considerado um sistema de grande porte, pois possui uma quantidade grande de funcionalidades e além disso, é utilizado em larga escala, pois é utilizado por grandes empresas, tais como Banco Itaú[6], Vale do Rio Doce[7], entre outros. A Tabela 4.1 mostra algumas informações que dão idéia do tamanho do sistema IBM Maximo.

Item	**Valor**
Linhas de Código (sem comentários e espaços)	1.202.342
Número de Classes	12968
Número de Aplicações	199
Número de Relatórios	329
Número de Tabelas no Banco de Dados	1430
Tamanho do Diretório no Sistema de Arquivos	2,4 Gigabytes
Número de Arquivos no Diretório	86.621

Tabela 4.1: Dados gerais sobre o sistema IBM Maximo.

4.3 Aplicação do Prometheus no Sistema IBM Maximo

O sistema IBM Maximo foi escolhido para a aplicação do Prometheus conforme foi descrito na seção anterior. Nesta seção, será descrito como o Prometheus foi aplicado para

[3]http://www-01.ibm.com/software/tivoli/products/maximo-oil-gas/ acessado em 17/04/2013
[4]http://www-01.ibm.com/software/tivoli/products/maximo-transportation/ acessado em 17/04/2013
[5]http://www-01.ibm.com/software/tivoli/products/maximo-life-sciences/ acessado em 17/04/2013
[6]http://www.itau.com.br/ acessado em 17/04/2013
[7]http://www.vale.com/EN/Pages/Landing.aspx acessado em 17/04/2013

customizar o sistema IBM Maximo baseando-se em um estudo de caso real extraído de um dos maiores usuários do sistema IBM Maximo: a *British Petroleum*.

A *British Petroleum*, ou simplesmente BP, é uma empresa do setor de extração e produção de combustíveis que possui sua matriz na Inglaterra, mas que tem pontos de exploração de petróleo e gás natural em diversos pontos do planeta, tais como Canadá, Índia, Estados e Brasil. Com o objetivo de padronizar a operação da empresa, foi adotada uma solução baseada na utilização de diversos sistemas de software chamada de *Work Management System*, ou simplesmente, WMS (Holland et al., 2005). Holland et al. (2005) descreveu a solução WMS como uma solução que utiliza diversos sistemas que cooperam entre si para auxiliar os processos operacionais da BP e que, juntos, trabalham como um único sistema ou um SoS (do inglês, *System of Systems*) (Jamshidi, 2009). No entanto, esses sistemas precisam sofrer várias alterações para que possam atender às necessidades da BP e para que possam interagir entre si. Além disso, cada um dos pontos de exploração da BP possui regras de negócio específicas impostas pela legislação do país no qual se encontram e por fatores associados à logística empregada nesse local. Assim, o sistema IBM Maximo, como parte da solução WMS, precisa ser customizado para que as regras de negócio da empresa possam ser incorporadas ao seu funcionamento. Por isso, ao longo dos últimos anos, o sistema IBM Maximo tem sido modificado intensamente como pode ser visto nas informações mostradas na Tabela 4.2 que contém alguns dados relativos à quantidade de alterações realizadas no sistema IBM Maximo pela BP. Outro exemplo da quantidade de alterações realizadas na solução WMS pode ser visto na Figura 4.3 que mostra a aplicação de "Ordem de Serviço" (do inglês, *Work Order*) na sua versão original do sistema IBM Maximo; a seguir, a versão da mesma aplicação modificada pela BP é mostrada na Figura 4.4.

Item	**Valor**
Linhas de Código (sem comentários e espaços)	39.643
Número de Classes	725
Número de Aplicações Alteradas	42
Número de Relatórios Alterados	33
Número de Arquivos no Diretório	1.669

Tabela 4.2: Dados da customização do sistema IBM Maximo realizada pela BP.

Nas seções a seguir, será descrito o resultado da aplicação de cada uma das seis atividades do Prometheus na customização do sistema IBM Maximo. É importante ressaltar que as informações utilizadas nas seções a seguir foram extraídas do projeto desenvolvido pela IBM para a BP no desenvolvimento da solução WMS. Dessa maneira, será possível comparar a implementação desenvolvida com a utilização de componentes OA com a abordagem orientada a objetos adotada pela IBM. Devido à grande quantidade de al-

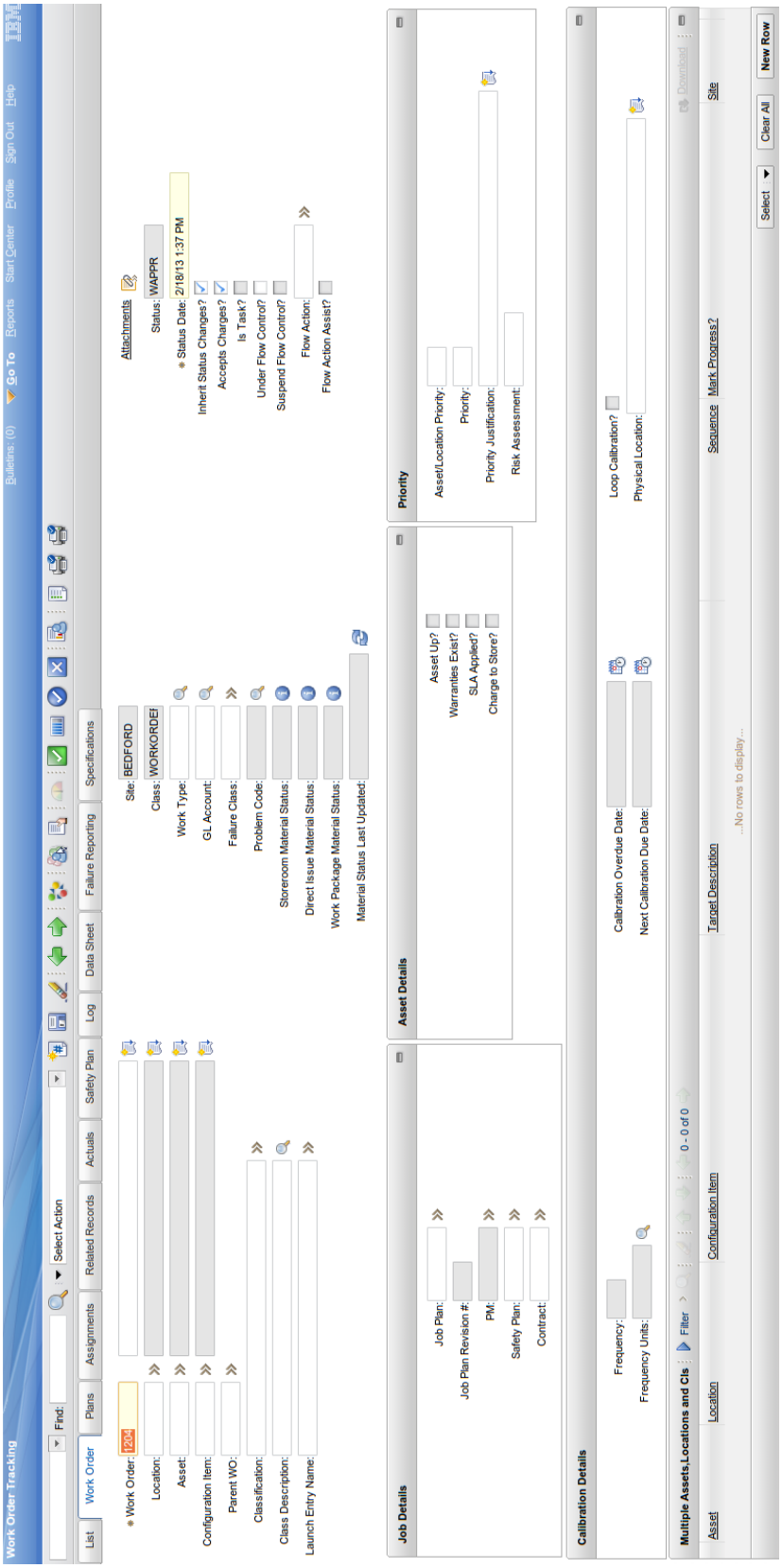

Figura 4.3: Tela da versão original da aplicação de "Ordem de Serviço" do sistema IBM Maximo.

Figura 4.4: Tela da versão customizada para a BP da aplicação de "Ordem de Serviço" do sistema IBM Maximo.

terações requisitadas pela BP no sistema IBM Maximo, o Prometheus será aplicado em duas solicitações de mudanças ao invés de considerar toda customização desenvolvida pela IBM. Isso será feito com o objetivo de facilitar a ilustrução do Prometheus neste trabalho. Finalmente, por se tratar de informação confidencial, algumas informações serão resumidas; no entanto, isso não afetará a compreensão da aplicação do Prometheus nesse estudo de caso.

4.3.1 Atividade "Receber Solicitações de Mudanças"

De acordo com o Prometheus, nessa atividade, as solicitações de mudanças devem ser propriamente documentadas para que possam ser implementadas nas atividades posteriores. Para documentar cada uma dessas solicitações de mudanças, a BP possui um documento específico que é armazenado no RTC[8] (do inglês, *Rational Team Concert*). No entanto, por se tratar de um documento confidencial as solicitações de mudanças serão descritas de maneira resumida. Assim, no contexto desse trabalho, foram escolhidas duas customizações realizadas no sistema IBM Maximo para adaptá-lo às necessidades da BP:

1. **Solicitação de Mudança 1**: alteração do mecanismo de processamento das requisições provenientes de outros sistemas. Essa solicitação de mudança consiste em modificar a maneira como o sistema IBM Maximo processa transações recebidas pelos seus web services vindas de outros sistemas que fazem parte da solução WMS. Por padrão, o sistema IBM Maximo executa todas as transações como um mesmo usuário que é configurável, mas tem que ser o mesmo para todas as transações. Na solução WMS, cada requisição precisa ser processada como o usuário que a originou, ou seja, cada requisição deverá ser processada com um usuário diferente. Assim, no xml da requisição, o usuário que será utilizado para processar a requisição deve ser identificado pelo atributo "username" como é mostrado na Figura 4.5. Além disso, faz parte dessa solicitação de mudança prover a capacidade de simular uma requisição sem que ela seja efetivada no sistema. Nesse caso, será utilizado o atributo "wmssimulate" que deverá conter o valor 0 se a simulação estiver desabilitada ou 1 se a simulação estiver habilitada como é mostrado na Figura 4.5; e

2. **Solicitação de Mudança 2**: alteração do cálculo da data de finalização de uma "ordem de serviço". No sistema IBM Maximo uma ordem de serviço contém todas as informações necessárias para que uma intervenção seja executada em um determinado equipamento: equipe de profissionais envolvida, caracterização do equipamento que será alvo da intervenção, materiais utilizados, serviços terceirizados que serão

[8]https://jazz.net/products/rational-team-concert/ acessado em 17/04/2013

necessários, atividades que serão desempenhadas, entre outros. Além disso, outro conceito relacionado é o de "plano de trabalho" que é um modelo de uma ordem de serviço que pode ser aplicado a uma ordem de serviço com o objetivo de atribuir um conjunto pré-definido de informações relacionadas à execução dessa ordem de serviço. Dessa maneira, essa solicitação de mudança consiste em substituir a maneira como a data de finalização de uma ordem de serviço é calculada pelo critério de cálculo utilizado pela BP. Segundo esse critério, a data de finalização de uma ordem de serviço deverá ser igual à data de sua criação ignorando-se o campo de estimativa de esforço da ordem de serviço; ou ser igual à estimativa de esforço do plano de trabalho se esse estiver associado à ordem de serviço. Na versão original do sistema IBM Maximo, o plano de trabalho não é considerado no cálculo da ordem de serviço e, além disso, o campo de estimativa de esforço presente na ordem de serviço é considerado no cálculo da data de finalização da ordem de serviço.

```
<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope" xmlns:max="http://www.ibm.com/maximo">
   <soap:Header/>
   <soap:Body>
        <ns1:SyncWMSPO creationDateTime="09/06/2009 05/19/32" messageID="messageId"
          xmlns:ns1="http://www.ibm.com/maximo" username="USUARIO" wmssimulate="0">
                <ns1:WMSPOSet>
                        <ns1:PO action="AddChange">
                                <ns1:CURRENCYCODE>CURRENCYCODE</ns1:CURRENCYCODE>
                                <ns1:ORDERDATE>2009-06-09T11:19:30</ns1:ORDERDATE>
                                <ns1:PONUM>PONUM</ns1:PONUM>
                                <ns1:DESCRIPTION>TESTE</ns1:DESCRIPTION>
                                <ns1:SITEID>SITEID</ns1:SITEID>
                        </ns1:PO>
                </ns1:WMSPOSet>
        </ns1:SyncWMSPO>
   </soap:Body>
</soap:Envelope>
```

Figura 4.5: XML que exemplifica uma requisição feita a um dos Web Services do sistema IBM Maximo mostrando o uso dos atributos "wmssimulate" e "username".

4.3.2 Atividade "Análise das Solicitações de Mudanças"

Seguindo o Prometheus, essa atividade tem o objetivo de classificar as solicitações de mudanças consideradas para implementação e adquirir as informações necessárias para que as solicitações de mudanças sejam incorporadas ao sistema. Ainda de acordo com o Prometheus, a classificação das solicitações de mudanças deverá seguir o critério estabelecido por Chapin et al. (2001) e que foi descrito na Seção 2.2.1 deste trabalho. Assim, as solicitações de mudanças consideradas nesse estudo de caso podem ser classificadas em:

1. **Solicitação de Mudança 1**: por se tratar de uma alteração que modificou o sistema para adequá-lo com o ambiente operacional, decidiu-se classificar essa solicitação de mudança como:

   (a) Tipo Geral: Propriedades do Sistema; e

   (b) Tipo Específico: Adaptativa.

2. **Solicitação de Mudança 2**: por se tratar de uma alteração que extendeu as regras de negócio existentes no sistema, decidiu-se classificar essa solicitação de mudança como:

   (a) Tipo Geral: Regras de Negócio; e

   (b) Tipo Específico: Extensão.

Conforme o Prometheus, após classificadas as solicitações de mudanças devem ser considerados os artefatos necessários para a sua implementação. Assim, para desenvolver as solicitações de mudanças utilizadas nesse estudo de caso, foram considerados os seguintes artefatos paro o seu desenvolvimento:

- Manual do Usuário do sistema IBM Maximo (IBM, 2012): contém a descrição das funcionalidades do sistema IBM Maximo e como utilizá-las no contexto operacional do cliente;

- Código Fonte do sistema IBM Maximo: a maioria dos clientes e empresas terceiras obtêm o código fonte do sistema IBM Maximo através da decompilação das classes Java, pois não possuem acesso ao código fonte e nem direitos autoriais sobre o produto. No entanto, isso não precisou ser feito neste trabalho, pois foi a IBM que implementou as customizações no sistema IBM Maximo requisitadas pela BP e que servem de base para esse estudo de caso. Portanto, no estudo de caso foi utilizado o código fonte original do sistema IBM Maximo.

Finalmente, o Prometheus contempla a construção de artefatos adicionais para apoiar o desenvolvimento das solicitações de mudanças. No entanto, não foi necessário construir nenhum artefato adicional, pois os artefatos listados acima foram considerados suficientes para apoiar as atividades de desenvolvimento.

4.3.3 Atividade "Identificação dos Componentes"

Seguindo o Prometheus, essa atividade tem o objetivo de identificar os componentes OA que serão desenvolvidos para que as solicitações de mudanças possam ser incorporadas ao sistema. Assim, com o objetivo de implantar as solicitações de mudanças consideradas nesse estudo de caso, foram projetados dois componentes OA:

- Componente **"WMSIntegrationAspect"**: esse componente OA contém o aspecto que será responsável pela implementação da **Solicitação de Mudança 1** e que pode agrupar todos os interesses transversais relacionados com as modificações da

camada de integração do sistema IBM Maximo. Nesse componente é definido um aspecto que intercepta o método processExternalDataAsync da classe psdi.iface.gateway. MEAGatewayBean afim de que o sistema IBM Maximo seja customizado adequadamente. Portanto, o ponto de junção para a definição desse aspecto é definido como byte [] psdi.. MEAGatewayBean.processExternalDataAsync (byte [], String, String). Finalmente, o método que implementa o comportamento do aspecto é denominado do prefixo "call" seguido pelo nome do método do sistema que é interceptado e, consequentemente, é chamado de callProcessExternalDataAsync;

- Componente **"WMSWorkOrderAspect"**: esse componente OA contém o aspecto que será responsável pela implementação da **Solicitação de Mudança 2** e que pode agrupar todos os interesses transversais relacionados com "ordens de serviço" do sistema IBM Maximo. Nesse componente é definido um aspecto que intercepta o método calculateTargetComplete da classe psdi.app.workorder.WO afim de que o sistema IBM Maximo seja customizado adequadamente. Portanto, o ponto de junção para a definição desse aspecto é definido como void psdi..*WO.calculateTargetComplete(). Finalmente, o método que implementa o comportamento do aspecto é denominado do prefixo "call" seguido pelo nome do método do sistema que é interceptado e, consequentemente, é chamado de callCalculateTargetComplete;

Ainda seguindo o Prometheus, uma representação gráfica desses componentes é recomendada. É importante ressaltar que existem diversas formas de representar os elementos da POA em diagaramas UML não existindo, portanto, um consenso sobre qual a melhor maneira de realizar essa representação (Guessi et al., 2011). Por isso, para esse estudo de caso foi considerada a proposta feita por Barra et al. (2004) na qual é utilizado um diagrama de componentes (OMG, 2007) e os interesses tranversais são representados pelas interfaces dos componentes. Assim, foi construído um diagrama de componentes utilizando-se a ferramenta "Astah UML"[9] que contém os componentes OA projetados e as suas interfaces nomeadas pelos métodos que implementam os comportamentos dos aspectos conforme é mostrado na Figura 4.6. Esses componentes serão disponibilizados sob a forma de módulos JAR (do inglês, *Java ARchive*) (SUN Microsystems, 2009) para que possam ser integrados ao sistema customizado. Além disso, no diagrama de componentes também são representadas as classes do sistema IBM Maximo que são interceptadas pelos componentes OA e os métodosessas classes que são representados na forma de interfaces. Finalmente, os pontos de junção são representados pelas notas associadas às conexões entre as interfaces dos componente OA e os métodos que elas interceptam.

[9]http://astah.net/editions/uml acessado em 17/04/2013

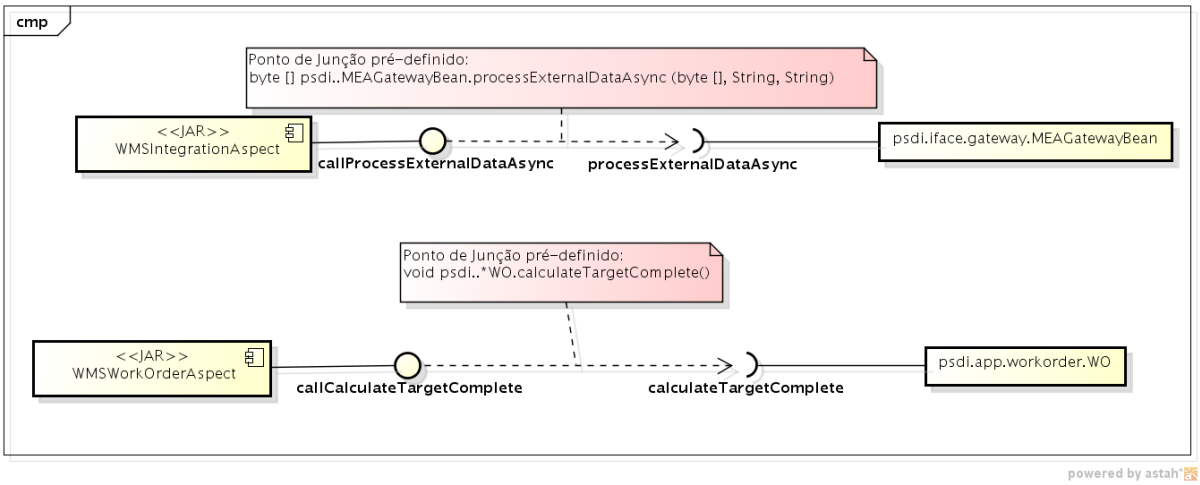

Figura 4.6: Diagrama de componentes monstrando os componentes OA projetados.

4.3.4 Atividade "Especificação dos Componentes"

Seguindo o Prometheus, essa atividade tem o objetivo de especificar os componentes OA que serão desenvolvidos. Nesse estudo de caso, os componentes OA projetados na atividade "Identificação dos Componentes" foram especificados de maneira que os métodos, classes, aspectos e pontos de junção fossem detalhados. Além disso, todas as dependências entre os elementos dos componentes OA também foram indicadas. Novamente, é importante ressaltar que existem diversas formas de representar os elementos da POA em diagaramas UML não existindo, portanto, um consenso sobre qual a melhor maneira de realizar essa representação (Guessi et al., 2011). Por isso, foi construído um diagrama de classes (OMG, 2007) que considera a proposta feita por Merson (2005) e que é mostrado na Figura 4.7. Esse diagrama contém os aspectos, os adendos, os pontos de junção e os métodos dos componentes OA definidos e as dependências desses componentes com as classes do sistema IBM Maximo. Ainda seguindo o Prometheus, nessa atividade poderiam ser descritas as pré e pós condições dos adendos dos componentes OA, caso esses adendos alterassem as pré e pós condições dos métodos que eles interceptam. No entanto, isso não foi necessário, pois os adendos considerados nessa solução não modificam as pré e pós condições dos métodos interceptados por eles. É importante destacar que os componentes OA projetados não apresentam nenhuma dependência com as aplicação do sistema IBM Maximo, mas dependem apenas do *framework* do sistema. Em outras palavras, esses componentes poderiam ser reutilizados em diferentes partes do sistema IBM Maximo em uso pela BP ou até mesmo em outro cliente.

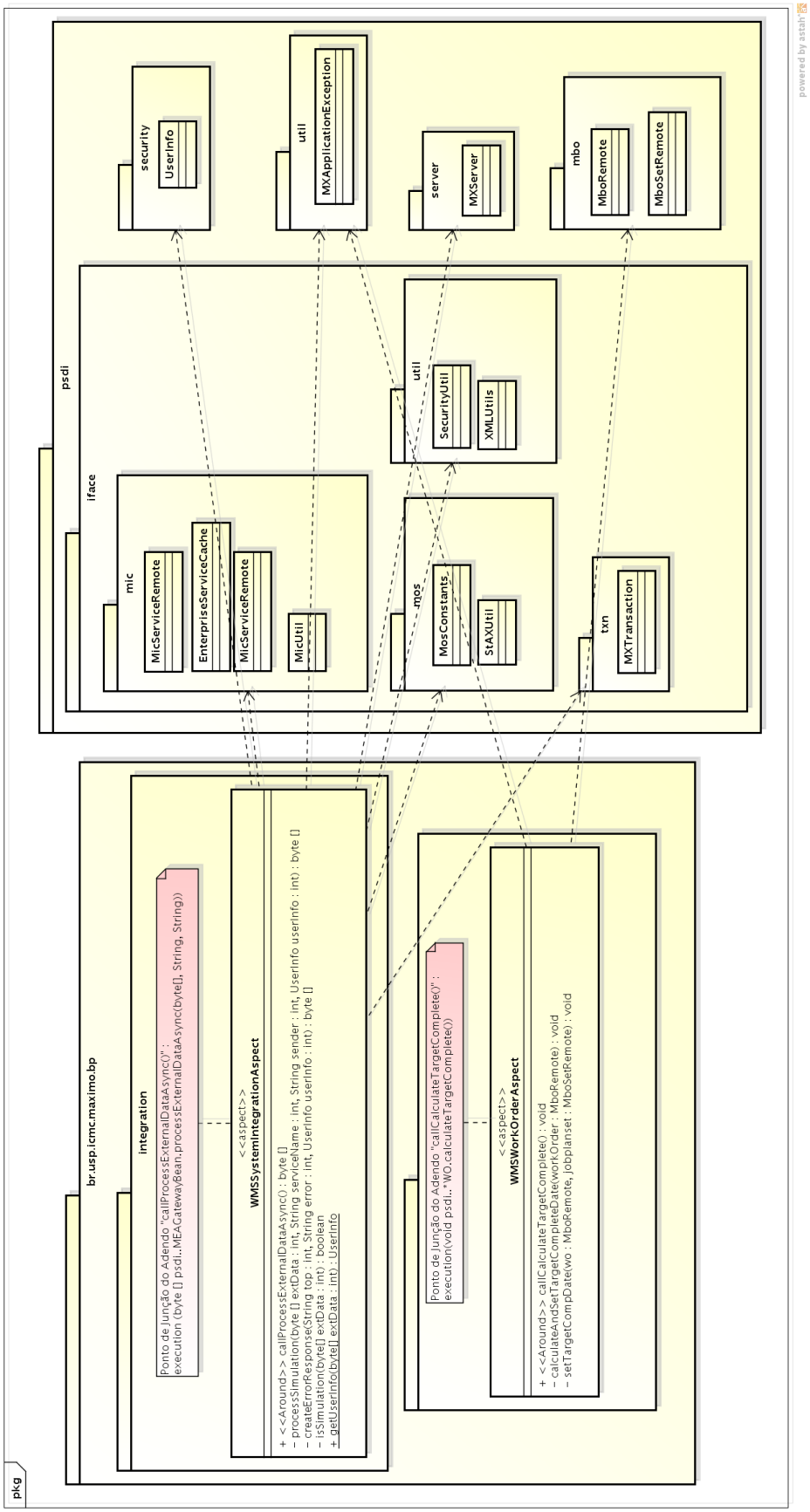

Figura 4.7: Diagrama de classes referente aos componentes OA especificados.

4.3.5 Atividade "Desenvolvimento e Integração dos Componentes"

Seguindo o Prometheus, essa atividade tem o objetivo de implementar os componentes OA e integrá-los ao sistema que está sendo customizado. Assim, os componentes OA projetados nesse estudo de caso foram implementados e integrados ao sistema IBM Maximo. Para implementar esses componentes foi escolhido o AspectJ[10] por se tratar de uma tecnologia de referência para o desenvolvimento de aplicações orientadas a aspectos utilizando-se a linguagem Java. Portanto, parte do código dos aspectos **"WMSSystemIntegrationAspect"** e **"WMSWorkOrderAspect"** podem ser vistos nas Figuras 4.8 e 4.9, respectivamente. Em ambas as figuras pode-se notar a utilização da anotação "@Around" que, além de indicar o adendo utilizado, contém a declaração do ponto de junção utilizado em cada aspecto. É importante ressaltar que o código fonte utilizado na customização desses aspectos foi extraído da solução WMS e, por isso, o código fonte dos métodos que contêm a parte principal da implementação de cada aspecto foi escondido, pois é muito similar ao código fonte em uso na BP, tratando-se, portanto, de informação confidencial.

```
WMSSystemIntegrationAspect.java
1   package br.usp.icmc.maximo.bp.integration;
2
3⊕  import java.io.ByteArrayOutputStream;
30
31  @Aspect
32  public class WMSSystemIntegrationAspect {
33
34      private static final MXLogger integrationLogger = MicUtil.INTEGRATIONLOGGER;
35
36      @Around("execution (byte [] psdi..MEAGatewayBean.processExternalDataAsync(byte[], String, String))")
37      public byte[] callProcessExternalDataAsync(ProceedingJoinPoint jointPoint) throws Throwable {
38
39          Object[] arguments = jointPoint.getArgs();
40
41          byte[] extData = (byte[]) arguments[0];
42          String serviceName = (String) arguments[1];
43          String sender = (String) arguments[2];
44
45          if (isSimulation(extData)) {
46              return processSimulation(extData, serviceName, sender, getUserInfo(extData));
47          } else
48              return (byte[]) jointPoint.proceed();
49
50      }
51
52⊕     private byte[] processSimulation(byte[] extData, String serviceName, String sender, UserInfo userInfo) {
103
104⊕    private byte[] createErrorResponse(String top, String error, UserInfo userInfo) throws MXException {
132
133⊕    private boolean isSimulation(byte[] extData) {
150
151⊕    public static UserInfo getUserInfo(byte[] extData) {
173 }
174
```

Figura 4.8: Código do aspecto de integração implementado no sistema IBM Maximo.

Além disso, nessa atividade foram utilizadas diversas ferramentas que apoiaram o desenvolvimento, tais como:

[10]http://www.eclipse.org/aspectj/ acessado em 17/04/2013

```
WMSWorkOrderAspect.java
1  package br.usp.icmc.maximo.bp.workorder;
2
3  import java.rmi.RemoteException;
13
14 @Aspect
15 public class WMSWorkOrderAspect {
16
17     @Around("execution(void psdi..*WO.calculateTargetComplete())")
18     public void callCalculateTargetComplete(JoinPoint jointPoint) throws MXException, RemoteException {
19         MboRemote workOrder = ((MboRemote) jointPoint.getThis());
20
21         calculateAndSetTargetCompleteDate(workOrder);
22     }
23
24     private void calculateAndSetTargetCompleteDate(MboRemote workOrder) throws RemoteException, MXException {
53
54     private void setTargetCompDate(MboRemote wo, MboSetRemote jobplanset) throws MXException, RemoteException {
81
82 }
```

Figura 4.9: Código do aspecto de "Ordem de Serviço" implementado no sistema IBM Maximo.

- IDE (do inglês *Integrated Developed Environment*), Eclipse[11]: foi utilizada para a implementação dos componentes OA e foi escolhida por ser a ferramenta de referência da IBM para desenvolvimento do sistema IBM Maximo;

- Plugin AJDT (do inglês, *AspjectJ Development Tools*[12]): o plugin AJDT da IDE Eclipse foi utilizado para facilitar a implementação dos componentes OA, pois possui funcionalidades que auxiliam a identificação dos pontos de junção e dos métodos afetados do sistema IBM Maximo;

- Servidor IBM WebSphere[13]: foi utilizado como servidor de aplicação JEE para executar o sistema IBM Maximo e é utilizado pela BP para executar a solução WMS; e

- Banco de Dados Oracle[14]: foi utilizado como sistema gerenciador de base dados para executar o sistema IBM Maximo e é utilizado pela BP para executar a solução WMS.

Ainda seguindo o Prometheus, nessa atividade os componentes OA devem ser integrados ao sistema customizado. É importante ressaltar que o sistema IBM Maximo utiliza a plataforma JEE como base tecnológica de sua arquitetura e, por isso, a integração e empacotamento dos componentes OA que foram implementados devem seguir os padrões definidos pela especificação dessa plataforma. Portanto, após concluída a implementação dos aspectos de cada componente OA, esses componentes foram empacotados em um

[11] http://www.eclipse.org/ acessado em 17/04/2013
[12] http://www.eclipse.org/ajdt/ acessado em 17/04/2013
[13] http://www-01.ibm.com/software/websphere/ acessado em 17/04/2013
[14] http://www.oracle.com/technetwork/database/enterprise-edition/overview/index.html acessado em 17/04/2013

módulo JAR (do inglês, *Java ARchive*) (SUN Microsystems, 2009) e ambos foram adicionados ao módulo EAR (do inglês, *Enterprise Application Archive*) (SUN Microsystems, 2009) do sistema IBM Maximo como é mostrado no diagrama de implantação (OMG, 2007) apresentado na Figura 4.10. Finalmente, o módulo EAR contendo os componentes OA foi instalado no servidor de aplicação JEE para poder ser executado.

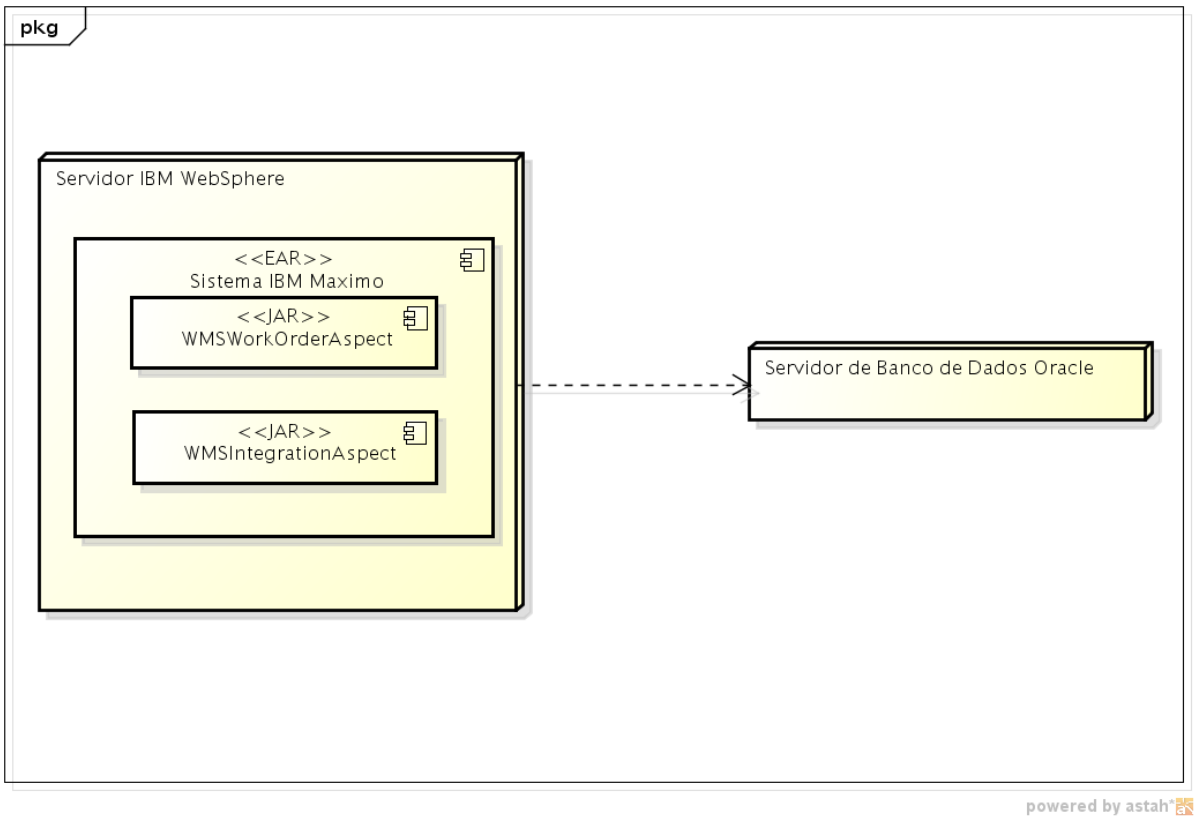

Figura 4.10: Diagrama de Implantação mostrando os componentes OA instalados no sistema IBM Maximo.

4.3.6 Atividade "Validação do Sistema"

Seguindo o Prometheus, é necessário validar as customizações incorporadaso ao sistema com o objetivo de garantir que os requisitos inicias encontram-se corretamente implementados. Nesse estudo de caso, achou-se por bem utilizar os casos de testes escritos pela IBM para realizar os testes funcionais na solução WMS, uma vez as solicitações de mudanças consideradas foram extraídas do sistema IBM Maximo customizado para a BP. Embora esses documentos sejam confidenciais, é possível descrever resumidamente o conteúdo dos testes funcionais realizados para validar cada solicitação de mudança considerada nesse estudo de caso:

1. **Solicitação de Mudança 1**: os casos de testes utilizados para verificar a implementação dessa solicitação de mudança tiveram como objetivo principal avaliar o uso dos atributos "wmssimulate" e "username" em requisições originadas de outros sistemas para o sistema IBM Maximo. Assim, foram executados testes que simulavam a requisição de um sistema externo utilizando dados válidos e inválidos, avaliando-se a resposta do sistema IBM Maximo diante de cada um desses cenários. Para simular essas requisições foi utilizada a ferramenta SoapUI[15] que é a ferramenta utilizada pela equipe da IBM para simular a integração do sistema IBM Maximo com outros sistemas. Dessa maneira, foram feitas requisições SOAP para criar, alterar e excluir ativos do sistema IBM Maximo e, ao final de cada requisição, eram avaliados os dados alterados no sistema. Ao final, o resultado dos testes realizados foram satisfatórios e o sistema integrado com o componente OA desenvolvido se comportou conforme o esperado; e

2. **Solicitação de Mudança 2**: os casos de testes utilizados para verificar a implementação dessa solicitação de mudança tiveram como objetivo principal avaliar o cálculo da data de finalização da ordem de serviço no sistema IBM Maximo considerando os critérios estabelecidos pela BP. Assim, foram executados testes manuais no sistema IBM Maximo considerando os planos de trabalho utilizados pela BP contendo dados reais de estimativa de esforço. Feito isso, foram criadas ordens de serviço que utilizassem os planos de trabalho considerados e foram avaliadas as datas de finalização calculadas pelo componente OA. Além disso, os mesmos testes foram realizados com diferentes configurações de fuso horário, pois a BP possui locais de trabalho em diversos fusos horários e os cálculos de data e hora devem considerar essas situações. Ao final, o resultado dos testes realizados foram satisfatórios e o sistema integrado com o componente OA desenvolvido se comportou conforme o esperado.

É importante ressaltar que não foram notadas alterações na performance do sistema customizado com os componentes OA durante a execução dos testes funcionais. Esse comportamento era previsto, uma vez que o uso de aspectos em um sistema não compromete sua performance ou escalabilidade (Singh e Kiczales, 2007).

4.4 Análise do Estudo de Caso

A aplicação do Prometheus no sistema IBM Maximo trouxe resultados importantes não só pelo fato de aplicá-lo em um sistema comercial de grande porte, mas também por

[15]http://www.soapui.org/ acessado em 17/04/2013

utilizar dados extraídos da customização desse sistema para a BP. Nesse contexto, é possível comparar os paradigmas orientados a objetos e orientados a aspectos, pois a customização realizada pela IBM não utilizou POA para alterar o sistema IBM Maximo enquanto que a customização realizada nesse estudo de caso fez uso da POA para alterar esse mesmo sistema. Assim, para comparar essas duas abordagens foram utilizadas as métricas propostas por Chidamber e Kemerer (1994) e que foram avaliadas por Dhanoa et al. (2011). As métricas utilizadas no contexto deste trabalho são:

- *Weighted Methods per Class* (WMC): essa métrica é a soma das complexidades dos métodos definidos em uma classe;

- *Coupling Between Objects* (CBO): essa métrica indica a dependência dessa classe com o restante do sistema. Um valor igual a 0 indica nenhuma relacionamento com nenhuma outra classe do sistema, valores entre 1 e 4 indicam um acoplamento baixo, por fim, valores superiores a 4 indicam um alto acoplamento dessa classe com o sistema; e

- *Depth of Inheritance Tree* (DIT): essa métrica indica a profundidade de uma classe na hierarquia.

Para dar suporte à avaliação da aplicação do Prometheus, foi utilizado o plugin Metrics[16] da IDE Eclipse com o objetivo de extrair as métricas citadas acima da solução WMS desenvolvida pela IBM e da solução desenvolvida pela aplicação do Prometheus conforme é ilustrado nas Figuras 4.11 e 4.12, respectivamente. Nessas figuras é possível observar diversas informações relacionadas a ambas as soluções, tais como, número de classes (do inglês, *number of classes*), contagem de linhas de código (do inglês, *total lines of code*), número de métodos (do inglês, *number of methods*), entre outros. É importante destacar que alguns valores apresentaram diminuição na solução desenvolvida utilizando-se o Prometheus em comparação com a solução desenvolvida pela IBM, tais como, número de métodos sobreescritos (do inglês, *number of overridden methods*) e índice de especialização (do inglês, *specialization index*).

O resultado da aplicação dessas métricas pode ser visto na Tabela 4.3 que destaca os valores médios das métricas extraídas de ambas as abordagens. A partir dos dados mostrados nessa tabela é possível observar que as métricas WMC e CBO ficaram praticamente inalteradas. Isso pode ser explicado pelo fato do código fonte utilizado na aplicação do Prometheus ser muito similar ao código fonte desenvolvido pela IBM para a BP. Dessa maneira, a métrica WMC mostra que as complexidades dos métodos utilizados nas duas

[16]http://metrics.sourceforge.net/ acessado em 17/04/2013

Problems | @ Javadoc | Declaration | Progress | Metrics - wmslbm | Console | Cross References

Metric	Total	Mean	Std. De	Maximum	Resource causing Maximum	Method
Number of Static Attributes (avg/max per type)	2	1	1	2	/wmslbm/src/bp/wms/iface/gateway/WMSMEAGatewayBean.java	
Number of Parameters (avg/max per method)		1.7778	1.2273	4	/wmslbm/src/bp/wms/iface/gateway/WMSMEAGatewayBean.java	processSim
Number of Classes (avg/max per packageFragment)	2	1	0	1	/wmslbm/src/bp/wms/app/workorder	
Specialization Index (avg/max per type)		0.625	0.625	1.25	/wmslbm/src/bp/wms/app/workorder/WMSWO.java	
Efferent Coupling (avg/max per packageFragment)		1	0	1	/wmslbm/src/bp/wms/app/workorder	
Number of Attributes (avg/max per type)	0	0	0	0	/wmslbm/src/bp/wms/app/workorder/WMSWO.java	
Abstractness (avg/max per packageFragment)		0	0	0	/wmslbm/src/bp/wms/app/workorder	
Number of Interfaces (avg/max per packageFragment)	0	0	0	0	/wmslbm/src/bp/wms/app/workorder	
Number of Static Methods (avg/max per type)	1	0.5	0.5	1	/wmslbm/src/bp/wms/iface/gateway/WMSMEAGatewayBean.java	
Normalized Distance (avg/max per packageFragment)		0	0	0	/wmslbm/src/bp/wms/app/workorder	
Total Lines of Code	193					
Number of Methods (avg/max per type)	8	4	0	4	/wmslbm/src/bp/wms/app/workorder/WMSWO.java	
Weighted methods per Class (avg/max per type)	39	19.5	6.5	26	/wmslbm/src/bp/wms/iface/gateway/WMSMEAGatewayBean.java	
Instability (avg/max per packageFragment)		1	0	1	/wmslbm/src/bp/wms/app/workorder	
Number of Packages	2					
Depth of Inheritance Tree (avg/max per type)		3.5	1.5	5	/wmslbm/src/bp/wms/app/workorder/WMSWO.java	
Nested Block Depth (avg/max per method)		2.5556	1.0657	4	/wmslbm/src/bp/wms/iface/gateway/WMSMEAGatewayBean.java	processSim
McCabe Cyclomatic Complexity (avg/max per method)		4.3333	2.4944	8	/wmslbm/src/bp/wms/iface/gateway/WMSMEAGatewayBean.java	processSim
Method Lines of Code (avg/max per method)	135	15	12.0646	41	/wmslbm/src/bp/wms/iface/gateway/WMSMEAGatewayBean.java	processSim
Lack of Cohesion of Methods (avg/max per type)		0	0	0	/wmslbm/src/bp/wms/app/workorder/WMSWO.java	
Number of Overridden Methods (avg/max per type)	1	0.5	0.5	1	/wmslbm/src/bp/wms/app/workorder/WMSWO.java	
Afferent Coupling (avg/max per packageFragment)		0	0	0	/wmslbm/src/bp/wms/app/workorder	
Number of Children (avg/max per type)	0	0	0	0	/wmslbm/src/bp/wms/app/workorder/WMSWO.java	

Figura 4.11: Métricas extraídas da solução WMS desenvolvida pela IBM.

Problems | @ Javadoc | Declaration | Progress | Metrics - wmsPrometheus | Console | Cross References

Metric	Total	Mean	Std. De	Maximum	Resource causing Maximum	Method
Number of Static Attributes (avg/max per type)	1	0.5	0.5	1	/wmsPrometheus/src/br/usp/icmc/maximo/bp/integration/WMSSystem	
Number of Parameters (avg/max per method)		1.75	1.0897	4	/wmsPrometheus/src/br/usp/icmc/maximo/bp/integration/WMSSystem	processSim
Number of Classes (avg/max per packageFragment)	2	1	0	1	/wmsPrometheus/src/br/usp/icmc/maximo/bp/integration	
Specialization Index (avg/max per type)		0	0	0	/wmsPrometheus/src/br/usp/icmc/maximo/bp/integration/WMSSystem	
Efferent Coupling (avg/max per packageFragment)		1	0	1	/wmsPrometheus/src/br/usp/icmc/maximo/bp/integration	
Number of Attributes (avg/max per type)	0	0	0	0	/wmsPrometheus/src/br/usp/icmc/maximo/bp/integration/WMSSystem	
Abstractness (avg/max per packageFragment)		0	0	0	/wmsPrometheus/src/br/usp/icmc/maximo/bp/integration	
Number of Interfaces (avg/max per packageFragment)	0	0	0	0	/wmsPrometheus/src/br/usp/icmc/maximo/bp/integration	
Number of Static Methods (avg/max per type)	1	0.5	0.5	1	/wmsPrometheus/src/br/usp/icmc/maximo/bp/integration/WMSSystem	
Normalized Distance (avg/max per packageFragment)		0	0	0	/wmsPrometheus/src/br/usp/icmc/maximo/bp/integration	
Total Lines of Code	198					
Number of Methods (avg/max per type)	7	3.5	0.5	4	/wmsPrometheus/src/br/usp/icmc/maximo/bp/integration/WMSSystem	
Weighted methods per Class (avg/max per type)	38	19	7	26	/wmsPrometheus/src/br/usp/icmc/maximo/bp/integration/WMSSystem	
Instability (avg/max per packageFragment)		1	0	1	/wmsPrometheus/src/br/usp/icmc/maximo/bp/integration	
Number of Packages	2					
Depth of Inheritance Tree (avg/max per type)		1	0	1	/wmsPrometheus/src/br/usp/icmc/maximo/bp/integration/WMSSystem	
Nested Block Depth (avg/max per method)		2.75	0.9682	4	/wmsPrometheus/src/br/usp/icmc/maximo/bp/integration/WMSSystem	processSim
McCabe Cyclomatic Complexity (avg/max per method)		4.75	2.3318	8	/wmsPrometheus/src/br/usp/icmc/maximo/bp/integration/WMSSystem	processSim
Method Lines of Code (avg/max per method)	139	17.375	11.0106	41	/wmsPrometheus/src/br/usp/icmc/maximo/bp/integration/WMSSystem	processSim
Lack of Cohesion of Methods (avg/max per type)		0	0	0	/wmsPrometheus/src/br/usp/icmc/maximo/bp/integration/WMSSystem	
Number of Overridden Methods (avg/max per type)	0	0	0	0	/wmsPrometheus/src/br/usp/icmc/maximo/bp/integration/WMSSystem	
Afferent Coupling (avg/max per packageFragment)		0	0	0	/wmsPrometheus/src/br/usp/icmc/maximo/bp/integration	
Number of Children (avg/max per type)	0	0	0	0	/wmsPrometheus/src/br/usp/icmc/maximo/bp/integration/WMSSystem	

Figura 4.12: Métricas extraídas da solução desenvolvida com o Prometheus.

soluções foram semelhantes. Com relação às dependências das duas abordagens com o *framework* do sistema IBM Maximo, também pode-se observar que a métrica CBO indicou que não houveram alterações nesse sentido.

No entanto, o valor da métrica DIT foi bem menor na solução desenvolvida com a utilização do Prometheus em comparação com a solução WMS desenvolvida pela IBM. Essa diferença de valores pode ser explicada pelo fato de que o componente OA não precisa estender nenhuma classe do sistema para alterar o seu comportamento. Essa observação é importante, pois a solução desenvolvida pela IBM para customizar o sistema IBM Maximo utiliza o recurso de herança o que torna necessário alterar a configuração do sistema em

arquivos e no banco de dados com o objetivo de que a classe herdada seja executada no lugar da classe original. Em contrapartida, essas alterações não foram necessárias na solução desenvolvida nesse estudo de caso, pois as classes originais do sistema IBM Maximo foram entrecortadas pelos aspectos dos componentes OA desenvolvidos. É importante frisar que essas alterações precisam ser empacotadas e distribuídas nos diversos pontos de exploração da BP para serem implantadas nos ambientes operacionais nos quais o sistema IBM Maximo é utilizado. Considerando o contexto de todas as alterações realizadas pela IBM para a BP, se o Prometheus fosse utilizado para customizar o sistema IBM MAXIMO, poderiam ter sido evitadas alterações no banco de dados para customizar 367 classes relacionadas com o comportamento de objetos e atributos; também poderiam ter sido evitadas alterações em cinco arquivos xml.

Métrica	**IBM**	**Prometheus**
Weighted Methods per Class (WMC)	19.5	19
Coupling Between Objects (CBO)	1	1
Depth of Inheritance Tree (DIT)	3.5	1

Tabela 4.3: Métricas utilizadas para comparar a solução desenvolvida pela IBM com a solução desenvolvida utilizando o Prometheus.

Por outro lado, é necessário observar que, embora os componentes OA desenvolvidos no contexto desse estudo possam ser reutilizados em outros contextos, podem haver algumas restrições quanto ao seu reúso em outros sistemas e em outros domínios. Isso deve ao fato de que os componentes OA desenvolvidos nesse estudo de caso apresentam dependências com o *framework* do sistema IBM Maximo e, além disso, eles foram desenvolvidos no domínio específico das solicitações de mudanças consideradas. No entanto, acredita-se que esses componentes possam ser reutilizados por outros clientes do sistema IBM Maximo que possuam necessidades semelhantes às da BP. Portanto, esse processo poderia ser aplicado em diferentes domínios e sistemas com o objetivo de comparar os resultados obtidos nesses contextos com aqueles obtidos neste trabalho.

Dentre as dificuldades encontradas para a realização desse estudo de caso pode-se destacar a pouca variedade de ferramentas para o desenvolvimento dos componentes OA. Essa dificuldade é expressa pelo pouco suporte ferramental a IDE's conhecidas, tais como Eclipse e NetBeans[17]. Dessa maneira, este trabalho pretende incentivar a melhoria das ferramentas disponíveis para o desenvolvimento de componentes OA. Além disso, a falta de padronização para representar os elementos da POA em diagramas UML pode dificultar a sua utilização em grades equipes de desenvolvimento, pois as ferramentas de modelagem UML existentes permitem diversas maneiras de representação. Por isso, este

[17]https://netbeans.org/ acessado em 17/04/2013

trabalho também visa estimular o desenvolvimento de ferramentas de modelagem UML que contenham formas padronizadas de representar os elementos da POA.

4.5 Considerações Finais

Neste capítulo foi apresentado o estudo de caso que contém a aplicação da proposta deste trabalho. Primeiramente, foi dada uma visão geral do sistema IBM Maximo. A seguir, foi descrita a aplicação das atividades do Prometheus na customização do sistema IBM Maximo tendo como base dados provenientes do uso desse sistema na empresa *British Petroleum*. É importante ressaltar que esse estudo de caso mostrou a viabilidade da utilização do Prometheus para customizar sistemas de software comerciais utilizando componentes OA. Além disso, o Prometheus poderia ser aplicado em outros sistemas e domínios com o objetivo de avaliar sua validade. Dentre as limitações e dificuldades encontradas nesse estudo de caso, destacam-se a ausência de um padrão para representar os elementos da POA na UML e a pouca variedade de ferramentas para o desenvolvimento de componentes OA. Finalmente, no próximo capítulo, será discutido em mais detalhes a conclusão deste trabalho.

Capítulo

5

Conclusão

NESTE capítulo são retomados brevemente o contexto no qual esse trabalho se insere e suas principais contribuições. São apresentadas também as dificuldades encontradas, limitações referentes a esta pesquisa e perspectivas de trabalhos futuros.

5.1 Caracterização da Pesquisa Realizada

O ciclo de vida de um software inicia-se na sua concepção e só termina depois que esse software deixa de ser utilizado. Em outras palavras, um software, após desenvolvido, entra em operação e, durante o período em que é utilizado, sofre inúmeras alterações com propósitos variados, mas que com o tempo, podem inclusive modificar drasticamente o contexto originalmente concebido para esse software. A atividades de manutenção é inevitável, pois todo o software que é útil e bem sucedido estimula novas solicitações dos usuários por mudanças e melhorias (Bennett e Rajlich, 2000). Nesse contexto, as modificações realizadas pela atividade de manutenção, muitas vezes, degradam a qualidade do software, introduzindo acoplamentos que não foram originalmente previstos e resultando em sistemas difíceis de manter e evoluir.

A atividade de manutenção de software tem se mostrado crítica no contexto do desenvolvimento de software na indústria (Chapin et al., 2001; Pressman, 2010). Diversos esforços têm sido realizados ao longo das últimas décadas para identificar, quantificar e qualificar os motivos e consequências das alterações que um software sofre ao longo de sua

vida útil (Chapin et al., 2001; Lehman et al., 1997; Lientz e Swanson, 1980). Esses esforços têm como objetivo principal reduzir os custos e minimizar os esforços para a manutenção dos sistemas de software, visto que o processo de evolução do software é inevitável (Bennett e Rajlich, 2000). Em particular, a atividade de customização de software é um tipo de manutenção que tem por objetivo a adaptação de um sistema de software a um domínio diferente do qual foi originalmente projetado (Nah et al., 2001). Vale destacar que a customização de um software requer considerável esforço e tempo, uma vez que as alterações necessárias estão, na maioria dos casos, espalhadas em diversas partes do sistema. Contudo, a customização é uma atividade bastante comum realizada na indústria de software. Dessa forma, pesquisas que proponham facilidades para a customização de software são bastante relevantes, uma vez que, a customização não tem sido largamente investigada, inclusive abordando a essa atividade em escala industrial.

5.2 Contribuições

A principal contribuição deste trabalho é o estabelecimento de uma abordagem sistematizada para a customização de sistemas de software utilizando componentes orientados a aspectos chamada de Prometheus. É importante destacar que, até onde os pesquisadores envolvidos neste trabalho têm conhecimento, o processo proposto nesse contexto é inédito. Além disso, um sistema de software de grande porte foi utilizado para ilustrar a aplicação do Prometheus em um estudo de caso real e acredita-se que esse estudo de caso dê suporte para a aplicação do Prometheus em outros sistemas.

No contexto da indústria de software, nota-se que as atividades de customização de sistemas contribuem para a degradação da qualidade da solução originalmente concebida, pois, na maioria das vezes, são realizadas de maneira informal e, além disso, são realizadas por profissionais que não apresentam nenhum vínculo com a empresa que desenvolveu o sistema ou até mesmo que não possuem formação adequada para desempenhar essas funções. Portanto, entende-se que o Prometheus vai contribuir para a área de Manutenção de Software com o objetivo de melhorar os aspectos relacionados com a manutenção dos sistemas de software tais como, reúso, acoplamento e coesão, uma vez que o processo possui um conjunto de atividades, artefatos e papéis concebidos com o intuito de que as atividades de customização causem o menor impacto possível nos sistemas em que ele fôr utilizado.

Finalmente, esta pesquisa contribuiu para a aproximação entre universidade e a indústria, uma vez que os produtos e tecnologias da IBM serviram para dar suporte à utilização da abordagem proposta neste trabalho. Além disso, os cenários utilizados no estudo de caso foram extraídos da BP o que ajudou a dar relevância para este trabalho. Por fim,

o Prometheus foi apresentado para profissionais da BP e da IBM despertando grande interesse nessas empresas inclusive com a possibilidade de utilizá-lo no futuro próximo.

5.3 Dificuldades e Limitações

Uma das dificuldades encontradas neste trabalho foi a falta de padronização das ferramentas de modelagem para representar os elementos da POA em diagramas UML. Essa dificuldade pode servir de impecílio para a utilização dos modelos UML em grandes times, pois mais de uma representação dos elementos da POA poderia ser utilizada dificultando a comunicação das informações contidas nesses modelos. Além disso, o trabalho com sistemas de software de grande porte demanda muito esforço computacional para atividades habituais tais como, compilação, empacotamento e implantação. Outra dificuldade encontrada, foi a pouca variedade de ferramentas para trabalhar com a POA.

A principal limitação deste trabalho é ter aplicado o Prometheus apenas no sistema IBM Maximo. Isso foi feito, pois não existiu tempo hábil aplicar o Prometheus em outros cenários no contexto desta pequisa. Portanto, este trabalho pode ser utilizado como base para realizar uma análise quantitativa considerando os resultados de mais estudos de caso. Além disso, outras métricas podem ser avaliadas para analisar a aplicabilidade do Prometheus. Com relação ao estudo de caso, ferramentas que automatizassem a aplicação do Prometheus poderiam ter sido desenvolvidas.

5.4 Trabalhos Futuros

Visando dar continuidade ao trabalho desenvolvido nesta pesquisa, a seguir são listadas as principais perspectivas de trabalhos futuros relacionadas a esse assunto:

- **Incentivar o desenvolvimento de ferramentas de modelagem UML para representação dos elementos da POA**: com o objetivo de padronizar a utilização da UML para projetos que envolvam elementos da POA pode-se desenvolver uma ferramenta que padronize a utilização da POA ou que possibilite a utilização de *profiles* da UML desenvolvidos para esse objetivo. Mais especificamente, será de extrema relevância se essas ferramentas puderem ser utilizadas pela indústria de software no desenvolvimento de novos sistemas;

- **Melhorar a performance das IDE's para dar suporte ao desenvolvimento de grandes sistemas**: visando diminuir o tempo e esforço computacional utilizados para trabalhar com sistemas de grande porte, pesquisas poderão ser conduzidas para otimizar tarefas habituais do desenvolvimento de sistema tais como, compilação,

empacotamento e implantação. Nesse sentido, a indústria de desenvolvimento de software pode contribuir para a melhoria das IDE's de maneira que sejam mais adequadas ao desenvolvimento desse tipo de sistema e que estejam disponíveis para utilização pela própria indústria de software;

- **Realizar uma análise quantitativa do Prometheus**: com o objetivo de validar a abordagem proposta neste trabalho, uma análise quantitativa poderá ser realizada. Essa análise pode envolver a aplicação do Prometheus na customização de outros sistemas de software e fornecer dados para uma análise quantitativa utilizando um conjunto mais abrangente de métricas e, portanto, mais ampla acerca da aplicabilidade desse processo;

- **Utilizar o Prometheus para customização de sistemas desenvolvidos pela IBM**: com o intuito de melhorar a manutenibilidade dos sistemas de software desenvolvidos pela IBM, o Prometheus poderá ser utilizado como base para construção de um modelo corporativo para a customização das soluções desenvolvidas pela empresa; e

- **Desenvolver ferramentas que automatizem a utilização do Prometheus**: como uma forma de diminuir ainda mais o tempo de customização nos sistemas de software, ferramentas poderão ser desenvolvidas para automatizar a utilização do Prometheus, especialmente se essas ferramentas forem compatíveis com IDE's amplamente conhecidas, tais como Eclipse e NetBeans.

Referências

Allen, P. R. *Realising E-business with Components.* Addison-Wesley, 2001.

Allen, P. R.; Frost, S. *Component-Based Development for Enterprise Systems: Apply the Select Perspective.* SIGS Books/Cambridge University Press, 1998.

Almeida, H.; Loureiro, E.; Perkusich, A.; Costa, E. Usando Aspectos para Personalizar a Execução de Aplicações Baseadas em Componentes de Prateleira. In: *Proceedings of the 2nd Workshop Brasileiro de Desenvolvimento de Software Orientado a Aspectos*, Uberlândia, MG, Brasil, 2005.

Araújo, J.; Moreira, A.; Brito, I.; Rashid, A. Aspect-Oriented Requirements with UML. In: *UML '02: Workshop on Aspect-Oriented Modeling with UML*, Dresden, Germany, 2002.

Atkinson, C.; Bayer, J.; Laitenberger, O.; Zettel, J. Component-Based Software Engineering: The KobrA Approach. In: *ICSE '00: Proceedings of the 22nd International Conference on Software Engineering, International Workshop on Component-Based Software Engineering*, Limerick, Ireland, 2000, p. 5–6.

Barra, E.; Génova, G.; Llorens, J. An approach to aspect modelling with uml 2.0. In: *Aspect-Oriented Modeling Workshop, AOM*, 2004.

Bass, L.; Clements, P.; Kazman, R. *Software Architecture in Practice.* Boston, MA, USA: Addison-Wesley, 2003.

Bennett, K. H.; Rajlich, V. T. Software Maintenance and Evolution: a Roadmap. In: *ICSE '00: Proceedings of the Conference on The Future of Software Engineering*, New York, NY, USA: ACM, 2000, p. 73–87.

Bonér, J.; Vasseur, A. AspectWerkz - Plain Java AOP. Online, (acessado em 26/02/2011), 2005.
Disponível em `http://aspectwerkz.codehaus.org/`

Brower, C.; Landau-Hubbard, G. M.; Morgan, S. R.; Whalley, M. Enterprise Asset Management. US Patent App. 12/188,090, 2008.

Brown, A. W.; Wallnan, K. C. Engineering of Component-Based Systems. In: *ICECCS '96: Proceedings of the 2nd IEEE International Conference on Engineering of Complex Computer Systems*, Washington, DC, USA: IEEE Computer Society, 1996, p. 414.

Canning, R. That Maintenance 'Iceberg'. *EDP Analyzer*, v. 10, n. 10, p. 1–14, 1972.

Chapin, N.; Hale, J. E.; Kham, K. M.; Ramil, J. F.; Tan, W.-G. Types of Software Evolution and Software Maintenance. *Journal of Software Maintenance: Research and Practice*, v. 13, p. 3–30, 2001.

Cheesman, J.; Daniels, J. *UML Components: A Simple Process for Specifying Component-based Software Systems.* Addison-Wesley, 2001.

Chidamber, S. R.; Kemerer, C. F. A Metrics Suite for Object Oriented Design. *IEEE Trans. Softw. Eng.*, v. 20, n. 6, p. 476–493, 1994.
Disponível em `http://dx.doi.org/10.1109/32.295895`

Clemente, P. J.; Hernandez, J.; Murillo, J. M.; Perez, M. A.; Sanchez, F. AspectCCM: an Aspect-oriented Extension of the CORBA Component Model. In: *Proceedings of the 28th Euromicro Conference*, 2002a, p. 10–16.

Clemente, P. J.; Sanchez, F.; Perez, M. A. Modeling with UML Component-based and Aspect Oriented Programming Systems. In: *ECOOP/WCOP '02: Proceedings of the 7th International Workshop on Component-Oriented Programming at European Conference on Object Oriented Programming*, Malaga, Spain, 2002b, p. 44–54.

Clements, P. C. From Subroutines to Subsystems: Component Based Software Development. *American Programmer*, v. 8, n. 11, p. 1–8, 1995.

Coady, Y.; Kiczales, G.; Feeley, M.; Smolyn, G. Using AspectC to improve the modularity of path-specific customization in operating system code. In: *Proceedings of 9th ACM SIGSOFT Internation Symposium on the Foundations of Software Engineering (FSE-9)*, 2001.

Cook, S.; Harrison, R.; Lehman, M. M.; Wernick, P. Evolution in software systems: foundations of the SPE classification scheme: Research Articles. *Journal of Software Maintenance and Evolution: Research and Practice*, v. 18, p. 1–35, 2006.

Cottenier, T.; Elrad, T. Validation of context-dependent aspect-oriented adaptations to components. In: *ECOOP '04: Proceedings of the 9th International Workshop on Component-Oriented Programming*, Oslo, Norway, 2004, p. 14–18.

Delamaro, M. E.; Maldonado, J. C.; Jino, M. *Introdução ao Teste de Software*. Elsevier, 2007.

Dhanoa, I. S.; Salaria, E. D. S.; Johal, H. S. Comparative Analysis of Java and AspectJ on the Basis of Various Metrics. In: *Proceedings of the 2011 10th IEEE/ACIS International Conference on Computer and Information Science*, ICIS '11, Washington, DC, USA: IEEE Computer Society, 2011, p. 270–275 (*ICIS '11*,).
Disponível em `http://dx.doi.org/10.1109/ICIS.2011.50`

Dijkstra, E. W. *A Discipline of Programming*. Prentice Hall, Inc., 1976.
Disponível em `http://www.amazon.com/exec/obidos/redirect?tag=citeulike07-20&path=ASIN/013215871X`

D'Souza, D. F.; Wills, A. C. *Objects, components, and frameworks with UML: the catalysis approach*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1999.

Duclos, F.; Estublier, J.; Morat, P. Describing and using non functional aspects in component based applications. In: *AOSD '02: Proceedings of the 1st International Conference on Aspect-oriented Software Development*, New York, NY, USA: ACM, 2002, p. 65–75.

Eler, M. M. *Um Método para o Desenvolvimento de Software Baseado em Componentes e Aspectos*. Dissertação de Mestrado, ICMC/USP, São Carlos, SP, 2006.

Elrad, T.; Filman, R. E.; Bader, A. Aspect-oriented Programming. v. 44, n. 10, p. 29–32, 2001.

Emmerich, W.; Kaveh, N. Component Technologies: Java beans, COM, Corba, RMI, EJB and The Corba Component Model. In: *SIGSOFT Software Engeneering Notes*, 2001, p. 311–312.

Erl, T. *Service-Oriented Architecture: A Field Guide to Integrating XML and Web Services*. Upper Saddle River, NJ, USA: Prentice Hall PTR, 2004.

Erl, T. *SOA: Principles of Service Design*, v. 1. Prentice Hall, 2008.

Erlikh, L. Leveraging Legacy System Dollars for E-Business. *IT Professional*, v. 2, p. 17–23, 2000.

Filman, R. E.; Elrad, T.; Clarke, S.; Aksit, M. *Aspect-Oriented Software Development*. Addison-Wesley Professional, 2011.

Filman, R. E.; Friedman, D. P. Aspect-Oriented Programming is Quantification and Obliviousness. In: *Workshop on Advanced Separation of Concerns (OOPSLA)*, OOPSLA '00, Minneapolis, USA, 2000 (*OOPSLA '00*,).

Gal, A.; Schroder-Preikschat, W.; Spinczyk, O. AspectC++: Language proposal and prototype implementation. In: *OOPSLA '01:Procceding of Workshop on Advanced Separation of Concerns in Object-Oriented Systems*, Tampa Bay, Florida, USA, 2001.

Goswell, C. The COM Programmer's Cookbook. Online, Microsoft Office Product Unit, (acessado em 26/02/2011), 1995.
Disponível em `http://msdn.microsoft.com/en-us/library/ms809982.aspx`

Grundy, J.; Patel, R. Developing Software Components with the UML, Enterprise Java Beans and Aspects. *Australian Software Engineering Conference*, v. 0, p. 0–127, 2001.

Guessi, M.; Oliveira, L. B. R.; Nakagawa, E. Y. Extensions of UML to Model Aspect-oriented Software Systems. *CLEI Electronic Journal*, v. 14, n. 1, p. 3–3, 2011.

Harrison, W.; Cook, C. Insights on improving the maintenance process through software measurement. In: *Proceedings of the Conference on Software Maintenance*, San Diego, CA: IEEE Computer Society Press, Los Alamitos, CA, 1990, p. 37–46.

Harrison, W.; Ossher, H.; Sutton, S.; Tarr, P. Supporting aspect-oriented software development with the Concern Manipulation Environment. *IBM Systems Journal*, v. 44, n. 2, p. 309–318, 2005.

Hirschfeld, R. AspectS - AOP with Squeak. In: *OOPSLA '01: Workshop on Advanced Separation of Concerns in Object-Oriented Systems*, Tampa Bay, Florida, USA, 2001.

Hoffman, K.; Eugster, P. Towards Reusable Components with Aspects: an Empirical Study on Modularity and Obliviousness. In: *ICSE '08: Proceedings of the 30th International Conference on Software Engineering*, New York, NY, USA: ACM, 2008, p. 91–100.

Holland, C. P.; Shaw, D. R.; Kawalek, P. BP's Multi-enterprise Asset Management System. *Inf. Softw. Technol.*, v. 47, n. 15, p. 999–1007, 2005.
Disponível em `http://dx.doi.org/10.1016/j.infsof.2005.09.006`

IBM IBM Maximo Asset Management, Version 7.5. Online, (acessado em 26/03/2013), 2012.
Disponível em `http://pic.dhe.ibm.com/infocenter/tivihelp/v49r1/index.jsp?topic=/com.ibm.mam.doc/welcome.html`

IEEE *Standard Glossary of Software Engineering Terminology.* Relatório Técnico, IEEE Institute, 1990.

IEEE *Standard for Software Maintenance (Std 1219).* Relatório Técnico, IEEE Institute, 1993.

ISO/EIC *12207: Software Life Cycle Processes.* International Standard 12207 Amendment 2, International Organization for Standardization, 2002.

Jacobson, I.; Booch, G.; Rumbaugh, J. *The Unified Software Development Process.* Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1999.

Jacobson, I.; Ng, P.-W. *Aspect-Oriented Software Development with Use Cases.* Addison-Wesley Professional, 2004.

Jamshidi, M. Introduction to System of Systems. *System of Systems Engineering*, v. 1, p. 1–20, 2009.

JBoss Inc. The JBoss 4 Application Server Guide. Online, (acessado em 26/02/2011), 2005.
Disponível em `http://onjava.com/pub/a/onjava/2003/05/28/aop_jboss.html`

Johnson, R.; Hoeller, J.; Arendsen, A.; Sampaleanu, C.; Harrop, R.; Risberg, T.; Davison, D.; Kopylenko, D.; Pollack, M.; Templier, T.; Vervaet, E. Spring - Java/J2EE Application Framework. Online, versão 1.2.9 (acessado em 26/02/2011), 2011.
Disponível em `http://static.springsource.org/spring/docs/1.2.9/reference/index.html`

Kay, A. C. History of Programming Languages—II. cáp. The early history of Smalltalk, New York, NY, USA: ACM, p. 511–598, 1996.

Kiczales, G.; Hilsdale, E.; Hugunin, J.; Kersten, M.; Palm, J.; Griswold, W. Getting Started with AspectJ. *Communications of the ACM*, v. 44, n. 10, p. 59–65, 2001a.

Kiczales, G.; Hilsdale, E.; Hugunin, J.; Kersten, M.; Palm, J.; Griswold, W. G. An Overview of AspectJ. In: *ECOOP '01: Proceedings of the 15th European Conference on Object-Oriented Programming*, London, UK, UK: Springer-Verlag, 2001b, p. 327–353.

Kiczales, G.; Irwin, J.; Lamping, J.; Loingtier, J. M.; Lopes, C. V.; Maeda, C.; Mendhekar, A. Aspect-Oriented Programming. In: *Proceedings of the European Conference on Object-Oriented Programming (ECOOP)*, Springer-Verlag, 1997, p. 220–242.

Kitchenham, B. A.; Charters, S. *Guidelines for Performing Systematic Literature Reviews in Software Engineering.* Relatório técnico EBSE 2007-001, Keele University and Durham University Joint Report, 2007.

Kruchten, P. *The Rational Unified Process: An Introduction.* 3 ed. Boston, MA, USA: Addison-Wesley Professional, 2004.

Laddad, R. *AspectJ in Action, Second Edition.* Manning Publications, 2009.

Larman, C. *Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development (3rd Edition).* Upper Saddle River, NJ, USA: Prentice Hall PTR, 2004.

Lehman, M. Programs, life cycles, and laws of software evolution. *Proceedings of the IEEE*, v. 68, n. 9, p. 1060–1076, 1980.

Lehman, M.; Ramil, J. F.; Sandler, U. An Approach to Modelling Long-Term Growth Trends in Software Systems. In: *ICSM '01*, Washington, DC, USA: IEEE Computer Society, 2001, p. 219–228.
Disponível em `http://portal.acm.org/citation.cfm?id=846228.848651`

Lehman, M. M. *Programs, Cities, Students, Limits to Growth?* Relatório Técnico, Imperial College of Science and Technology, London, UK, inaugural Lecture Series (Vol. 9, 1970, 1974, pp. 211-229), 1974.

Lehman, M. M. On understanding laws, evolution, and conservation in the large-program life cycle. *Journal of Systems and Software*, v. 1, p. 213–221, 1979-1980.
Disponível em `http://www.sciencedirect.com/science/article/B6V0N-48TD28P-53/2/51df5317a72a296bb7819c211555db83`

Lehman, M. M. Laws of Software Evolution Revisited. In: *EWSPT '96: Proceedings of the 5th European Workshop on Software Process Technology*, London, UK: Springer-Verlag, 1996, p. 108–124.

Lehman, M. M.; Belady, L. A. *Program evolution: processes of software change.* San Diego, CA, USA: Academic Press Professional, Inc., 1985.

Lehman, M. M.; Ramil, J. F.; Perry, D. E. On Evidence Supporting the FEAST Hypothesis and the Laws of Software Evolution. In: *METRICS '98*, Washington, DC, USA: IEEE Computer Society, 1998, p. 84–88.
Disponível em `http://portal.acm.org/citation.cfm?id=823455.823936`

Lehman, M. M.; Ramil, J. F.; Wernick, P. D.; Perry, D. E.; Turski, W. M. Metrics and Laws of Software Evolution - The Nineties View. In: *METRICS '97*, Washington, DC, USA: IEEE Computer Society, 1997, p. 20–32.
Disponível em `http://portal.acm.org/citation.cfm?id=823454.823901`

Leon, A. *Enterprise Resource Planning.* Tata McGraw-Hill Education, 2007.

Lientz, B. P.; Swanson, E. B. *Software Maintenance Management.* Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1980.

McIlroy, D. Mass-produced Software Components. In: Buxton, J. M.; Naur, P.; Randell, B., eds. *ICSE '69: Proceedings of the 1st International Conference on Software Engineering*, Garmisch Pattenkirchen, Germany: NATO Science Committee, 1969, p. 138–155.

Mens, T.; Demeyer, S. *Software Evolution.* Springer Publishing Company, 2008.

Merson, P. Representing Aspects in the Ssoftware Architecture - Practical Considerations. In: *Early Aspects Workshop*, Citeseer, 2005.

Mezini, M.; Lieberherr, K. Adaptive plug-and-play components for evolutionary software development. *SIGPLAN Not.*, v. 33, p. 97–116, 1998.

Nah, F. F.-H.; Faja, S.; Cata, T. Characteristics of ERP Software Maintenance: a Multiple Case Study. *Journal of Software Maintenance and Evolution: Research and Practice*, v. 13, n. 6, p. 399–414, 2001.

OMG *Object Management Group Unified Modeling Language (OMG UML), Infrastructure, V2.1.2.* Relatório Técnico, Object Management Group, 2007.
Disponível em `http://www.omg.org/spec/UML/2.1.2/Infrastructure/PDF`

OMG Common Object Request Broker Architecture (CORBA) Specification. Online, Object Management Group, versão 3.1 (acessado em 26/02/2011), 2008.
Disponível em `http://www.omg.org/spec/CORBA/3.1/`

Osborne, W. M.; Chikofsky, E. J. Guest Editors' Introduction: Fitting Pieces to the Maintenance Puzzle. *IEEE Software*, v. 7, n. 1, p. 11–12, 1990.

OSGi OSGi Service Platform Release 4. Online, Open Source Gateway Initiative, versão 4.1 (acessado em 26/02/2011), 2007.
Disponível em `http://www.osgi.org/Download/Release4V42`

Parnas, D. L. On the Criteria to be Used in Decomposing Systems into Modules. *Commun. ACM*, v. 15, n. 12, p. 1053 – 1058, 1972.

Perez, J.; Navarro, E.; Letelier, P.; Ramos, I. A modelling proposal for aspect-oriented software architectures. In: *ECBS '06: Proceedings of the 13th Annual IEEE International Symposium and Workshop on Engineering of Computer Based Systems*, 2006, p. 10 pp.–41.

Pfleeger, S. L. *Engenharia de Software: Teoria e Prática*. 2 ed. Sao Paulo, SP, Brasil: Prentice Hall, 2004.

Popovici, A.; Gross, T.; Alonso, G. Dynamic Weaving for Aspect-Oriented Programming. In: *AOSD '02: Proceedings of the 1st International Conference on Aspect-oriented Software Development*, New York, NY, USA: ACM, 2002, p. 141–147.

Pressman, R. *Software Engineering: A Practitioner's Approach*. 7 ed. New York, NY, USA: McGraw-Hill, Inc., 2010.

Rashid, A. Aspect-Oriented Requirements Engineering: An Introduction. In: *RE '08: Proceedings of the 16th International Requirements Engineering*, Los Alamitos, CA, USA: IEEE Computer Society, 2008, p. 306–309.

Rashid, A.; Moreira, A.; Araújo, J. Modularisation and Composition of Aspectual Requirements. In: *AOSD '03: Proceedings of the 2nd International Conference on Aspect-Oriented Software Development*, New York, NY, USA: ACM, 2003, p. 11–20.
Disponível em `http://doi.acm.org/10.1145/643603.643605`

Redbooks, I. *Maximo Asset Management Essentials V7. 1 Implementer's Guide*. Vervante, 2008.

Rubinger, A. L.; Burke, B. *Enterprise JavaBeans 3.1*. O'Reilly Media, 2010.

Singh, A.; Kiczales, G. The scalability of AspectJ. In: *Proceedings of the 2007 conference of the center for advanced studies on Collaborative research*, CASCON '07, Riverton, NJ, USA: IBM Corp., 2007, p. 203–214 (*CASCON '07*,).
Disponível em `http://dx.doi.org/10.1145/1321211.1321233`

Sneed, H. M. Planning the Reengineering of Legacy Systems. *IEEE Software*, v. 12, p. 24–34, 1995.

Sommervile, I. *Engenharia de Software*. 8 ed. Sao Paulo, SP, Brasil: Pearson Education - Br, 2007.

Stamey, J.; Saunders, B.; Cameron, M. Aspect-Oriented PHP. Online, (acessado em 26/02/2011), 2004.
Disponível em `http://www.aophp.net/`

SUN Microsystems Java Platform Enterprise Edition (Java EE) Specification. Online, Microsoft Office Product Unit, versão 6 (acessado em 26/02/2011), 2009.
Disponível em `http://jcp.org/aboutJava/communityprocess/final/jsr316/index.html`

Suvée, D. JAsCo language reference 0.8.7. Online, (acessado em 26/02/2011), 2005.
Disponível em `http://ssel.vub.ac.be/jasco/lib/exe/fetch49de.pdf?cache=cache&media=documentation%3Ajasco.pdf`

Swanson, E. B. The Dimensions of Maintenance. In: *ICSE '76: Proceedings of the 2nd International Conference on Software Engineering*, Los Alamitos, CA, USA: IEEE Computer Society Press, 1976, p. 492–497.

Swanson, E. B.; Chapin, N. Interview with E. Burton Swanson. *Journal of Software Maintenance and Evolution: Research and Practice*, v. 7, p. 303–315, 1995.

Szyperski, C. *Component Software: Beyond Object-Oriented Programming*. Addison-Wesley / ACM Press, 2002.

The AspectJ Team The AspectJ Programming Guide. Online, (acessado em 26/02/2011), 2003.
Disponível em `http://eclipse.org/aspectj/doc/released/progguide/index.html`

Zhang, J.; Li, H.; Li, F. Research on Aspect Connectors for Software Architecture Adaptation. In: *SEW '07: Proceedings of the 31st IEEE Software Engineering Workshop*, Washington, DC, USA: IEEE Computer Society, 2007, p. 63–66.