Universidade Estadual de Maringá
Centro de Tecnologia - Departamento de Informática
Especialização em Desenvolvimento de Sistemas para *Web*

Integração de Aspectos e Serviços Web no Desenvolvimento de Sistemas Distribuídos

Gislaine Camila Lapasini Leal

Prof. Dra. Elisa Hatsue Moriya Huzita
Orientadora

Maringá, 2007.

Universidade Estadual de Maringá
Centro de Tecnologia - Departamento de Informática
Especialização em Desenvolvimento de Sistemas para *Web*

Gislaine Camila Lapasini Leal

Integração de Aspectos e Serviços Web no Desenvolvimento de Sistemas Distribuídos

Trabalho submetido à Universidade Estadual de Maringá como requisito para obtenção do título de Especialista de Desenvolvimento de Sistemas para *Web*.

Universidade Estadual de Maringá
Centro de Tecnologia - Departamento de Informática
Especialização em Desenvolvimento de Sistemas para *Web*

Gislaine Camila Lapasini Leal

Integração de Aspectos e Serviços Web no Desenvolvimento de Sistemas Distribuídos

Profº. Dr. Wesley Romão Ass.:_________________________
Profª. Dra. Elisa Hatsue Moriya Huzita (orientadora) Ass.:_________________________
Profª. Maria Madalena Dias Ass.:_________________________
Profª. Thelma Elita Colanzi Lopes Ass.:_________________________

Dedico este trabalho
aos meus pais.

AGRADECIMENTOS

A Deus, pela força que sempre esteve comigo neste desafio.

Meus sinceros agradecimentos a Prof.ª Elisa Hatsue Moriya Huzita, pela orientação e ajuda fundamental para a conclusão deste trabalho.

Aos meus pais, Noeli Terezinha Lapasini dos Santos e Gervásio Santana Leal, que sempre me incentivaram a seguir adiante, acreditando em mim e no meu trabalho.

Ao professor Paulo César Masiero e seus alunos do mestrado (Marcelo Medeiros Eler e Antonielly Garcia Rodrigues) pelas contribuições no início deste trabalho.

As professoras Maria Madalena Dias e Thelma Elita Colanzi Lopes, pela presença na banca examinadora.

Aos amigos José Roberto, Márcia e Rafael pelos conselhos e contribuições.

Aos demais amigos aqui não citados que de alguma forma me ajudaram e incentivaram.

RESUMO

Este trabalho tem por objetivo aplicar uma metodologia para a integração da tecnologia de aspectos e serviços web num ambiente que oferece suporte ao desenvolvimento distribuído de software, o DiSEN. A análise da integração dessas duas tecnologias é motivada pelos benefícios oferecidos pela Programação Orientada a Aspectos (POA) e pela necessidade de separar interesses não-funcionais também presentes na Computação Orientada a Serviços (COS). Para que o objetivo deste trabalho seja cumprido são analisadas algumas metodologias que integram essas tecnologias e o modelo SPC. Por fim, o trabalho apresenta a modelagem do SPC seguindo uma das abordagens de integração analisadas.

ABSTRACT

This work has for objective to apply a methodology for the integration of the technology of aspects and services web in an environment that offers support to the distributed development of software, the DiSEN. The analysis of the integration of these two technologies is motivates for the benefits offered for the Aspect Oriented Programming (AOP) and for the necessity to also separate to interests not-functionaries gifts in the Computation Oriented Service (COS). So that the objective of this fulfilled work if either is analyzed some methodologies that integrate these technologies and model SPC. Finally the work presents the modeling of SPC following one of the analyzed boardings of integration.

LISTA DE ILUSTRAÇÕES

Figura 2.1: Processo de combinação de aspectos e componentes......................................24
Figura 3.1: Pápeis nos serviços web...27
Figura 3.2: Pápeis e operações básicas nos serviços web..28
Figura 3.3: Pilha de Protocolo dos Serviços Web...28
Figura 3.4: Elementos de uma mensagem SOAP...31
Figura 3.5: Elementos de um documento WSDL..32
Figura 3.6: Estrutura do UDDI..34
Figura 3.7: UDDI na publicação e descoberta de serviço web...34
Figura 4.1: Relacionamento entre os modelos da MDA...41
Figura 4.2: Definição da notação *Aspect-Oriented UML Profile*.......................................43
Figura 5.1: Arquitetura do DiSEN...47
Figura 5.2: Representação do gerenciado de workspace..49
Figura 5.3: Proposta do Modelo SPC apresentada por Pozza...50
Figura 5.4: Diagrama de Atividades do SPC..51
Figura 5.5: Diagrama de Classe do *framework* de cooperação...53
Figura 6.1: Representação do modelo SPC na arquitetura dos serviços web.....................57
Figura 6.2: Diagrama com a distribuição dos métodos entre as classes.............................58
Figura 6.3: Diagrama com a identificação dos possíveis aspectos....................................58
Figura 6.4: *Workspace* e seus aspectos...61
Figura 6.5: Percepção e seus aspectos...62
Figura 6.6: Sincronização e seus aspectos...63
Figura 6.7: Comunicação e seus aspectos. 64
Figura A1: PIM do Modelo SPC utilizando serviços web e aspectos................................68

LISTA DE QUADROS

Quadro 4.1: Comparação entre PSM, PIM sem aspectos e PIM com aspectos...45
Quadro 6.1: Propriedades candidatas a aspectos...59

LISTA DE ABREVIATURAS SIGLAS E SÍMBOLOS

ACID	Atomicidade, Consistência, Isolamento e Durabilidade
ADS	Ambiente de Desenvolvimento de Software
AOWS	*Aspect-Oriented Web Service*
BPEL4WS	*Bussiness Process Execution Language For Web Services*
CASS	*Context Aspect Sensitive Service*
CIM	*Computational Independent Model*
CORBA	*Common Object Request Broker Architecture*
COS	Computação Orientada a Serviços
DAML	*DARPA Agent Markup Language*
DiSEN	*Distributed Software Engineering Environment*
GW	Gerenciador de *Workspace*
HTML	*Hyper Text Markup Language,*
HTTP	*HyperText Transport Protocol*
MDA	*Model Driven Architecture*
PIM	*Platform Independently Model*
POA	Programação Orientada a Aspectos
POO	Programação Orientada a Objetos
POP	Programação Pós-Objeto
PSM	*Platform Specific Model*
RDF	*Resource Definition Format*
RMI	*Remote Method Invocation*
RPC	*Remote Procedure Calls*
SGML	*Standard Generalized Markup Language*
SMTP	*Simple Mail Transfer Protocol*
SOA	*Service Oriented Architecture*
SOAP	*Simple Object Access Protocol*
SPC	Sincronização Percepção Comunicação
UDDI	*Universal Description, Discovery and Integration*
UML	*Unified Modelling Language*
XML	*eXtensible Markup Language*
W3C	*World Wide Web Consortium*

WSDL	*Web Service Description Language*
WSLM	*Web Services Management Layer*

SUMÁRIO

1. INTRODUÇÃO 16
1.1 Objetivos 17
1.2 Organização da monografia 17
1.3 Metodologia 18
2. PROGRAMAÇÃO ORIENTADA A ASPECTOS 19
2.1 Introdução 19
2.2 Programação Orientada a Aspectos 20
2.2.1 Componentes 22
2.2.2 Aspectos 22
2.2.3 Pontos de Junção 23
2.2.4 Conjuntos de Pontos de Junção 24
2.2.5 Adendos 24
2.2.6 Ortogonalidade 24
2.2.7 Combinação 25
3. DESENVOLVIMENTO ORIENTADO A SERVIÇOS 26
3.1 Introdução 26
3.2 Definição 26
3.3 Arquitetura de Serviços Web 28
3.4 Tecnologias Associadas 30
3.4.1 XML 30
3.4.2 SOAP 31
3.4.3 WSDL 33
3.4.4 UDDI 34
3.5 Aplicações dos Serviços Web 36
4. INTEGRAÇÃO DE ASPECTOS E SERVIÇOS WEB 37
4.1 Introdução 37
4.2 Integração em Nível de Aplicação 37
4.3 Integração em Nível de Desenvolvimento 38
4.4 Integração em Nível de Composição de Serviços 39
4.5 Integração em Nível de Descrição de Serviços 40
4.6 Integração em Nível de Modelo 40
4.6.1 MDA 41
4.7 Análise das Estratégias de Integração 42
5. AMBIENTE DiSEN 48
5.1 Introdução 48
5.2 Arquitetura do DiSEN 48
5.3 Modelo SPC 52
6. ESTUDO DE CASO 57
6.1 Introdução 57
6.2 Proposta 57
6.3 Modelagem 57
6.4 Considerações Finais 65
7. CONCLUSÃO 66
7.2 Trabalhos Futuros 67
REFERÊNCIAS 68
ANEXO A – PIM do Modelo SPC 71

1. INTRODUÇÃO

Os sistemas distribuídos surgiram com a finalidade de melhor aproveitar a capacidade das máquinas distribuídas pelas redes do mundo. Um sistema distribuído é uma coleção de elementos de processamento ligados através de um subsistema de comunicação e equipados com um software distribuído. Software distribuído é definido como todo o sistema de software que faz com que diferentes elementos de processamento trabalhem como uma única máquina em relação a um conjunto de funcionalidades (COULOURIS *et. al.*, 2001).

Além do melhor aproveitamento dos recursos computacionais, o desenvolvimento de sistemas distribuídos é justificado pelos seguintes fatores: pessoas e informações são distribuídas, desempenho/custo, modularidade, expansibilidade, disponibilidade e confiabilidade. Mesmo com todas estas vantagens, a construção de sistemas distribuídos ainda é uma tarefa complexa. Tal complexidade se deve aos custos que a adoção deste tipo de solução acarreta, tanto em termos de aquisição quanto manutenção.

Os sistemas distribuídos apresentam ainda os seguintes aspectos: compartilhamento de recursos, abertura, paralelismo, independência de escala, tolerância a falhas, transparência e estado compartilhado. Tais características fundamentam as principais razões para a sua grande complexidade.

O paradigma da Computação Orientada a Serviços (COS) contempla as características desejáveis em um sistema distribuído, visto que o mesmo objetiva promover o desenvolvimento de aplicações distribuídas através da composição rápida e de baixo custo de componentes de software autônomos e independentes, os chamados serviços (Newcomer, 2002).

A natureza distribuída e fracamente acoplada das aplicações baseadas em serviços *web* faz com que muitos interesses (coordenação, composição de serviços, segurança e outros), em geral de cunho não-funcional, sejam difíceis de modularizar usando apenas as tecnologias existentes. Por se referir tanto às aplicações provedoras quanto às aplicações consumidoras dos serviços envolvidos, o tratamento desses interesses tende a se espalhar pelo código fonte de diversas aplicações, misturando-se à implementação de seus interesses funcionais. Tal propriedade pode afetar negativamente o desenvolvimento de aplicações orientadas a serviços, dificultando não apenas a sua implementação, mas acometendo também a manutenção e evolução.

A constatação de que existem interesses de um sistema de software que são difíceis de modularizar utilizando modelos tradicionais de desenvolvimento, baseados em mecanismos de decomposição funcional, constitui a principal motivação para o surgimento do paradigma da Programação Orientada a Aspectos (POA) (Kiczales et al., 1997). A POA propõe uma abstração, o aspecto, que permite separar interesses não-funcionais de um sistema que do contrário estariam misturados ao seu código funcional.

Tendo em vista os benefícios oferecidos pela POA, e a necessidade de separar interesses não-funcionais também presentes na COS, motivaram a análise da integração dessas duas tecnologias.

1.1 Objetivos

O objetivo geral dessa monografia é verificar a integração da Programação Orientada a Aspectos com a tecnologia de Serviços Web. A partir do objetivo geral foram definidos os seguintes objetivos específicos:

- Levantamento do estado da arte da POA;
- Levantamento do estado da arte da COS;
- Levantamento bibliográfico das abordagens que propõem a integração de Aspectos e Serviços Web.
- Aplicação de uma abordagem de integração num estudo de caso no ambiente DiSEN.

1.2 Organização da monografia

Além deste capítulo introdutório, esta monografia encontra-se estruturada em sete capítulos, os quais são descritos abaixo, e um anexo.

- Capítulo 2: apresenta uma visão geral do estado da arte da Programação Orientada a Aspecto, descrevendo os seus componentes.
- Capítulo 3: apresenta uma visão geral do estado da arte do Desenvolvimento Orientado a Serviços, abordando a definição, a arquitetura, tecnologias envolvidas e a aplicação dos Serviços Web.
- Capítulo 4: apresenta um levantamento bibliográfico das metodologias que integram a tecnologia de Serviços Web e a Programação Orientada a Aspectos.
- Capítulo 5: descreve o ambiente DiSEN, apresenta a sua arquitetura e descreve o modelo SPC.

- Capítulo 6: detalha a proposta dessa monografia de integração de serviços web e aspectos e apresenta o estudo de caso.
- Capítulo 7: refere-se à conclusão a respeito deste trabalho, além das contribuições e sugestões para trabalhos futuros.

1.3 Metodologia

A metodologia para desenvolvimento desta monografia seguiu os seguintes passos:

1. Fundamentação teórica com estudo de Programação Orientada a Aspectos e Serviços *Web*.
2. Levantamento e análise das metodologias de integração das duas tecnologias;
3. Estudo do ambiente DiSen;
4. Formulação e desenvolvimento do estudo de caso;
5. Considerações finais.

2. PROGRAMAÇÃO ORIENTADA A ASPECTOS

2.1 Introdução

A necessidade de software de qualidade impulsionou o uso da Programação Orientada a Objetos (POO) (BOOCH et al, 2000) em busca de reusabilidade e manutenibilidade, fatores que se relacionam com o custo e a velocidade de desenvolvimento. A reusabilidade implica na diminuição dos custos gerais de desenvolvimento e o fator manutenibilidade por sua vez refere-se à atividade onerosa que consome os maiores esforços durante o processo de desenvolvimento, sendo caracterizada por um alto custo e baixa velocidade de implementação.

A estrutura desacoplada provida pela POO facilita o reuso e a manutenibilidade, aumentando assim a produtividade e o suporte a mudanças de requisitos. No entanto, a POO apresenta algumas limitações (separar preocupações relacionadas às restrições globais do software das propriedades sistêmicas). Elrad et al (2001) discutem tais limitações no que se refere à separação de interesses e algumas técnicas de programação pós-objeto (POP, *Post-Object Programming*) para melhorar a POO. Alguns exemplos de abordagem POP são: Programação Adaptativa (Adaptive Programming), Programação Generativa (G*enerative Programming*), Metaprogramação (M*etaprogramming*), Desenvolvimento Orientado a Característica (*Feature-Oriented Development*), Filtros de Composição (C*omposition Filters*) e Programação Orientada a Aspectos (A*spect Oriented Programming*).

A Programação Orientada a Aspectos (POA) é uma abordagem que tem se sobressaído ultimamente, pois, ela contribui para uma melhor separação de preocupações nos artefatos de software, por conseguinte possibilita que os benefícios da separação de preocupações possam ser melhor incorporados ao processo de desenvolvimento (VIEGAS & VUAS, 2000 apud SOUSA, 2004).

Segundo LOPES (1997), as aplicações, em geral, estão ampliando os limites das técnicas de programação atuais, de modo que determinadas características de um sistema afetam seu comportamento, de forma que as técnicas atuais não conseguem capturar tais propriedades satisfatoriamente.

Desse modo, quando essas propriedades precisam ser implementadas e inseridas no sistema de diversas formas, geralmente arbitrárias, dificultam o reuso, comprometem a legibilidade, aumentando a complexidade no que se refere à implementação dos

componentes funcionais do sistema. Tal fenômeno é denominado de entrelaçamento do código.

Diante de tal cenário surge o conceito de aspectos, uma abstração que objetiva prover suporte a um melhor isolamento de *crosscutting concerns* (Piveta, 2001). Segundo Kiczales et al (1997), um aspecto é uma unidade modular que entrecorta outras unidades modulares. Na seção seguinte aborda-se com mais detalhes a Programação Orientada a Aspectos e os conceitos envolvidos.

2.2 Programação Orientada a Aspectos

A POA, uma extensão da POO, foi introduzida por Gregor Kiczales (1997) com o intuito de fornecer uma forma de desenvolvimento que considera separadamente os requisitos funcionais (funções ou atividades que o sistema faz ou fará) e os não funcionais (condições de comportamento e restrições que devem prevalecer) de um sistema, para sua posterior combinação em um único sistema. Ela objetiva introduzir uma nova dimensão na POO para que existam tanto objetos quanto aspectos, sendo que o primeiro encapsula os requisitos funcionais e o segundo os não funcionais (KICZALES et al, 1997). Constantinides & Hassoun (2002 apud Sousa, 2004) ressaltam que a natureza de aspectos não se restringe a requisitos não-funcionais. Eles podem representar tanto preocupações funcionais quanto não-funcionais.

Na concepção de Elrad et al (2001), a POA consiste na separação dos interesses de um sistema em unidades modulares e posterior composição desses módulos em um sistema completo.

A POA procura solucionar a ineficiência em capturar algumas das importantes decisões de projeto que um sistema deve implementar. Esta dificuldade faz com que a implementação destas decisões de projeto sejam distribuídas pelo código, resultando num entrelaçamento e espalhamento de código com diferentes propósitos, o que torna oneroso o processo de desenvolvimento e manutenção destes sistemas (SOARES & BORBA, 2004).

A adoção da POA pode tornar o código do sistema mais fácil de escrever, compreender, reutilizar e manter, pois as propriedades entrecortantes não aparecem no texto dos componentes, ficando devidamente encapsuladas nos aspectos. Com isso a complexidade dos componentes é reduzida, as alterações são facilitadas e o tamanho do código diminui, proporcionando assim um melhor gerenciamento da complexidade dos componentes e redução dos erros.

As linguagens de programação orientadas a aspectos compõem-se de cinco elementos para permitir a modularização dos interesses ortogonais (ELRAD et al. 2001):

i) modelo de pontos de junção descrevendo os possíveis pontos em que o comportamento do aspecto pode ser adicionado;

ii) meio de identificar os pontos de junção;

iii) meio de especificar o comportamento adicional nos pontos de junção;

iv) unidades que encapsulam a especificação dos pontos de junção e as melhorias comportamentais providas pelo aspecto; e

v) mecanismo para combinar aspectos e componentes.

Diversas linguagens oferecem suporte a implementação do conceito de aspectos. Tais linguagens se encontram estruturadas em dois grupos: as de propósito específico e as de propósito geral (PIVETA, 2001).

As linguagens de propósito específico possuem um escopo restrito, se limitam a resolução de alguns tipos de problemas. Alguns exemplos deste tipo de linguagem são: JST, que especifica a sincronização de objetos; COOL, que aborda mecanismos de exclusão mútua em threads concorrentes, sincronização, notificação e comandos de guarda; RIDL, que explora mecanismos para a manipulação de dados entre diferentes espaços de execução; e D2AL, que centra-se na questão de objetos distribuídos.

As linguagens de propósito geral apresentam um escopo abrangente, são utilizadas para a confeccção de programas genéricos e normalmente são construídas com base em linguagens de programação existentes. São exemplos desta categoria: AspectC++, AspectS, Aspectc# e AspectJ.

Atualmente, a identificação de aspectos está associada à experiência e intuição do desenvolvedor, pois não há um conjunto de passos sistemáticos e nem metodologias para isso. Desse modo, os aspectos ficam quase sempre restritos aos requisitos não-funcionais, pois os requisitos originados de preocupações não-funcionais são naturalmente ortogonais porque impõem restrições em como a funcionalidade do sistema será realizada e por isso, são quase sempre associados a requisitos funcionais (KOTONOYA & SOMMERVILLE, 1998 apud SOUSA, 2004).

As seções seguintes abordam os conceitos de componentes, aspectos, pontos de junção, ortogonalidade e combinação introduzidos por Kiczales et al (1997) que constituem o núcleo da POA (Chavez & Lucena, 2003) e de pontos de corte.

2.2.1 Componentes

Kiczales et al (1997) definem um componente como uma unidade de decomposição funcional de um sistema, uma propriedade ou interesse que pode ser claramente encapsulada em um procedimento generalizado, isto é, um objeto, um método, um procedimento ou, uma API (*Application Programming Interface*). Em Gimenez & Huzita (2005), um componente é definido como uma unidade de software independente, que encapsula, dentro de si, seu projeto e implementação, e oferece serviços, por meio de interfaces bem definidas para o meio externo.

O termo componente é usado para denotar uma entidade que modulariza uma parte funcional do sistema mais ou menos independente das demais partes, e que pode ser naturalmente composta com outros componentes através de mecanismos de composição (CHAVEZ & LUCENA, 2003). Os componentes são provenientes da decomposição funcional de um sistema, independentes de a linguagem ser estruturada ou orientada a objetos, representam os interesses não-ortogonais do sistema.

2.2.2 Aspectos

Aspectos são definidos como propriedades do sistema que entrecortam os componentes e, mais especificamente, como propriedades que afetam o desempenho ou a semântica dos componentes de forma sistemática (Kiczales et al, 1997). Eles envolvem diversos componentes funcionais e não podem ser expressos utilizando as notações e linguagens atuais de uma maneira bem localizada. Algumas propriedades que geralmente são vistas como aspectos em relação à funcionalidade básica de um sistema são:

- Sincronização de objetos concorrentes: Um programa em que dados globais sejam compartilhados necessita sincronizar suas atividades através de mecanismos que possibilitem sua execução (CAHILL & DEMPSEY, 1997); (KICZALES et al, 1997).
- Distribuição: Ao tratar de projeto e implementação de sistemas distribuídos é necessário abordar uma série de problemas que não existem em sistemas não-distribuídos. Na etapa de projeto é necessário modificar a perspectiva de como os componentes devem ser especificados, envolvendo o entendimento de diferentes componentes e o relacionamento entre eles. Quanto à implantação, as linguagens de programação atualmente utilizadas deixam a desejar em prover suporte a essas

diferentes perspectivas e propriedades que são ortogonais umas às outras (LOPES, 1997).

- Tratamento de exceção: As políticas de tratamento de exceção genéricas podem ser vistas e implementadas como aspectos. Como exemplo, pode-se mencionar o uso de pré e pós-condições que devem ser satisfeitas quando uma determinada operação é realizada, tais condições podem ser implementadas como aspectos (OSSHER & TARR, 1998).
- Coordenação de múltiplos objetos: a implementação da coordenação de múltiplos objetos apresenta diversos problemas no que se refere ao reuso e a extensão. A necessidade de sincronizar a integração entre objetos ativos em busca de um objetivo global torna a implementação uma tarefa complexa (HARRISON & OSSHER, 1993).
- Persistência: é um elemento importante em sistemas computacionais que aparece espalhado pelo código do sistema. Tal característica pode ser implementada como um aspecto, independente de como e quais componentes devem ser persistidos. A recuperação de dados é realizada no momento que o objeto é acessado e a atualização da base de dados ocorre no momento da criação ou modificação do estado do objeto (OSSHER & TARR, 1998).

Além destas propriedades, características como serialização, atomicidade, replicação, segurança, visualização, *logging*, *tracing*, tolerância à falhas e obtenção de métricas podem ser vistas como aspectos (OSSHER & TARR, 1998).

2.2.3 Pontos de Junção

Os pontos de junção são elementos semânticos da linguagem de componentes que coordenam os aspectos (KICZALES et al, 1997).

Segundo Chavez & Lucena (2003), o termo pontos de junção denota um elemento relacionado à estrutura ou à execução de um componente do programa que é referenciado, e possivelmente afetado, por um aspecto. Os pontos de junção podem ser estáticos ou dinâmicos. Um ponto de junção estático é posicionado na estrutura do componente, enquanto um ponto de junção dinâmico é posicionado na execução do programa do componente.

Soares & Borba (2004) definem um ponto de junção como um ponto bem definido no fluxo de execução de um programa.

2.2.4 Conjuntos de Pontos de Junção

Os conjuntos de pontos de junção são responsáveis por selecionar pontos de junção, ou seja, eles detectam em que ponto do programa os aspectos deverão interceptar. Em AspectJ um aspecto, geralmente, define conjuntos de junção, que são aninhados por pontos de junção através de operadores lógicos *e*, *ou* e *não* (&&, ||, e !) (LADDAD, 2003).

2.2.5 Adendos

Adendos é o código para ser executado em um ponto de junção que está sendo referenciado pelo conjunto de junção. Existem três maneiras de adendos: antes, durante e depois (*before*, *around* e *after*). Portanto, de acordo com seus nomes, *before* executa antes do ponto de junção, *around* executa antes e depois e *after* executa depois.

O adendo pode modificar a execução do código no ponto de junção, pode substituir ou passar por ele. Usando o adendo pode-se "logar" as mensagens antes de executar o código de determinados pontos de junção que estão espalhados em diferentes módulos. O corpo de um adendo é muito semelhante ao de qualquer método, encapsulando a lógica a ser executada quando um ponto de junção é alcançado (GRADECKI & LESIECLKI, 2003).

2.2.6 Ortogonalidade

O fenômeno da ortogonalidade é diretamente responsável pelo entrelaçamento e espalhamento do código. Em geral, sempre que duas propriedades sendo programadas e compostas diferentemente e ainda assim se coordenarem são ditas ortogonais (KICZALES et al, 1997).

O termo ortogonalidade é usado para denotar o mecanismo de composição utilizado para compor aspectos e componentes. No que se refere às dimensões, a ortogonalidade pode ser homogênea ou heterogênea. A homogênea, também chamada de vertical, pode ser aplicada quando a ortogonalidade fornecer intensa e exatamente o mesmo conjunto de um ou mais componentes. A heterogênea, também conhecida como horizontal, é aplicada

quando a ortogonalidade fornece acentuadamente um subconjunto aplicado a diferentes tipos de componentes (CHAVEZ & LUCENA, 2003).

A ortogonalidade pode ser estática ou dinâmica, sua natureza é determinada pelo tipo (estático ou dinâmico) dos pontos de junção utilizados (CHAVEZ & LUCENA, 2003).

2.2.7 Combinação

A combinação é o processo de compor aspectos e componentes relacionados a uma ortogonalidade num ponto de junção especificado. Um combinador é a ferramenta responsável por realizar a composição entre aspectos e componentes (KICZALES et al, 1997). A figura 2.1 ilustra o processo de combinação de aspectos e componentes.

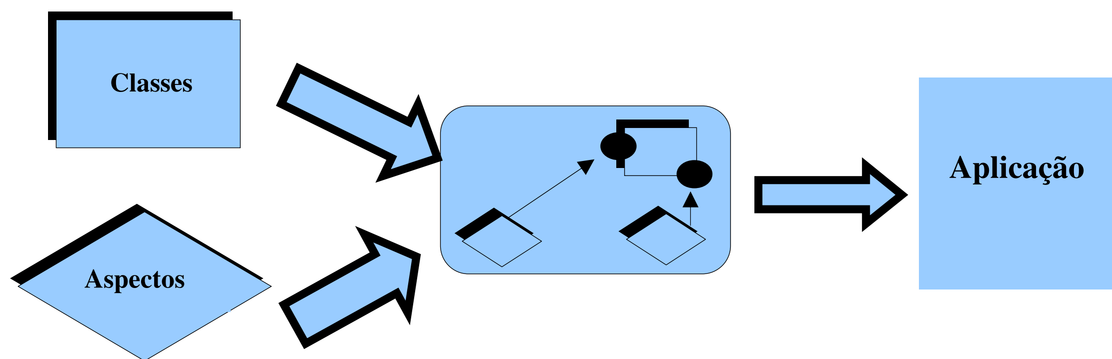

Figura 2.1: Processo de combinação de aspectos e componentes (Adaptado de Souza, 2004).

Um combinador de aspectos pode trabalhar sobre código fonte, *byte code* ou código objeto. Ele pode ser estático ou dinâmico. Combinadores estáticos são aqueles que mesclam componentes e aspectos numa nova versão de código fonte antes ou durante a compilação, enquanto que os dinâmicos realizam a combinação durante a execução (CHAVEZ & LUCENA, 2003).

3. DESENVOLVIMENTO ORIENTADO A SERVIÇOS

3.1 Introdução

Nos últimos anos, com o avanço da *Web* e a redução dos custos de comunicação, as empresas passaram a investir na integração de suas aplicações comerciais via *Web*, ou seja, no *Business-to-Business*. Para tanto, os desenvolvedores dessas aplicações necessitam lidar com requisitos funcionais inovadores e com problemas relacionados à interação e à integração de aplicações tais como escalabilidade, dinamismo, autonomia, heterogeneidade e adaptação de sistemas legados (SNELLl *et al* , 2001).

Diante de tal cenário surge uma nova concepção de software, a qual entende que o mesmo deve entregar-se na forma de serviço. Nessa concepção, o software deixa de ser algo que se instala e passa a ser um serviço obtido através da rede por meio de mecanismos padronizados de troca de mensagens. Esse novo paradigma é chamado de Computação Orientada a Serviço (COS), e representa uma evolução natural do desenvolvimento de software baseado em componentes, oferece uma maneira prática para viabilizar a integração de diferentes aplicações independentemente dos seus ambientes de desenvolvimento e das suas plataformas de execução.

A COS abre novos mercados no mundo de software, tanto para os provedores de serviços específicos, os de pequena escala, como para as grandes organizações que provêem serviços de caráter mais amplo. Cabe ressaltar que o desenvolvimento da COS só foi possível graças ao suporte oferecido pelos padrões e pelas tecnologias de rede.

Nas seções seguintes são apresentadas a definição, arquitetura, tecnologias envolvidas e aplicação dos serviços web.

3.2 Definição

A literatura apresenta diversas definições para os serviços web que os caracteriza ora como uma interface e ora como parte lógica de negócios das aplicações. Algumas definições são:

- “Um serviço *web* é uma interface posicionada entre o código da aplicação e o usuário desse código. Atua como uma camada de abstração, separando os detalhes específicos de plataforma e linguagem de programação de como a aplicação é invocada. Os serviços *web* não são novos, mas representam a evolução de princípios que guiaram a internet por anos” (SNELL *et al* , 2001).

- "Um serviço web é uma interface que descreve uma coleção de operações que são acessíveis pela rede, utilizando mensagens padronizadas. É descrito utilizando uma notação formal em XML, chamada de descrição de serviço" (KREGER, 2001).
- "Uma aplicação identificada por um URI (*Universal Resource Identifier*), cujas interfaces e implementações são capazes de serem definidas, descritas e localizadas, utilizando-se a linguagem XML e suas ferramentas. Um serviço web deve ser capaz de interagir com outras aplicações, através da troca de mensagens baseadas em XML, utilizando os protocolos de comunicação existentes na internet" (AUSTIN *et al.*, 2002).

Em Cerami (2002), um serviço *web* é caracterizado como qualquer serviço que:

- esteja disponível sobre a internet ou uma rede privada (intranet);
- use o padrão XML para a troca de mensagens;
- seja independente de sistema operacional e linguagem de programação;
- seja descrito por uma gramática XML;
- seja possível de descobrir através de mecanismos simplificados de busca.

Segundo Booth *et al.* (2004), um serviço *web* é uma noção abstrata que deve ser implementada por um agente concreto. O agente é o módulo de software ou o hardware que envia e recebe mensagens, enquanto o serviço é o recurso caracterizado pelo conjunto abstrato de funcionalidades que é fornecido. Embora o agente possa mudar, o serviço permanece o mesmo.

O serviço *web* deve apresentar uma descrição e uma semântica. A descrição do serviço contém os detalhes da interface do serviço. Os detalhes de implementação são ocultados pela interface, permitindo que tais aplicações sejam fracamente acopladas, orientadas a componentes e implementadas em diversas tecnologias. A semântica é o comportamento esperado quando se interage com o serviço.

As definições de serviços *web* estão associadas às características das tecnologias que lhes dão suporte, conforme Snell *et al.* (2001) apresenta:

- Baseados em XML - Utilizar a tecnologia XML como uma camada de representação dos dados, facilitando assim a interoperabilidade entre as aplicações;
- Fracamente acoplados – Processo de invocação de aplicações simplificado, sem as restrições impostas pelas outras tecnologias de integração de sistemas, tais como CORBA e RMI.
- Capazes de interagir em modo de execução síncrono ou assíncrono;

- Suporte de chamadas a procedimentos remotos (RPC) – serviços web permitem que as aplicações consumidoras realizem chamadas a métodos remotos;
- Suporte a troca de documento – serviços web permitem a comunicação entre aplicações através da troca de documentos baseados em XML.

3.3 Arquitetura de Serviços Web

A arquitetura de serviços *web* pode ser vista sob duas perspectivas. A primeira analisa individualmente os papéis de cada ator do serviço *web,* a segunda analisa a pilha de protocolos do serviço *web* (CERAMI, 2002).

De acordo com a primeira perspectiva, numa arquitetura de serviços *web* há três papéis principais, os quais são apresentados na Figura 3.1:

- **Provedor de serviço** – é a plataforma que hospeda uma implementação do serviço *web* e a disponibiliza na internet.
- **Consumidor de serviço** – é uma aplicação que utiliza uma operação de descoberta para recuperar a descrição do serviço localmente, ou a partir de módulos de registro, e usa a descrição para invocar ou interagir com a implementação do serviço.
- **Registro de serviço** – é um diretório logicamente centralizado dos serviços. Fornece um local centralizado onde os desenvolvedores podem publicar novos serviços ou encontrar um existente.

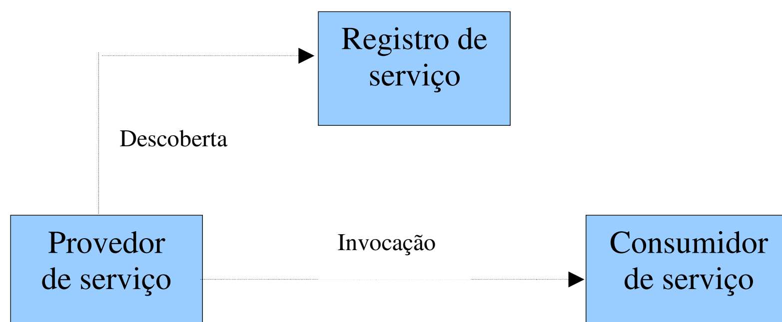

Figura 3.1: Pápeis nos serviços web (Cerami, 2002).

A interação entre os papéis num serviço *web* pode ser realizada através de três operações: publicação, descoberta e ligação (invocação). A seguir, detalha-se cada uma destas operações:

- **Publicação -** o provedor publica metadados sobre um serviço para que as aplicações consumidoras possam encontrá-lo.
- **Descoberta -** é a operação que permite a uma aplicação consumidora recuperar uma descrição do serviço diretamente ou consulta o registro de serviços com o objetivo de encontrar uma descrição do tipo de serviço desejado.
- **Ligação -** esta operação possibilita que a aplicação consumidora invoque ou inicie uma interação com um serviço web a partir das informações contidas na descrição do serviço.

A Figura 3.2 ilustra a arquitetura de interação entre o provedor, o registro e o consumidor, os quais são representados pelo hexágono e são chamados de papéis. Os círculos apresentam os elementos desses papéis.

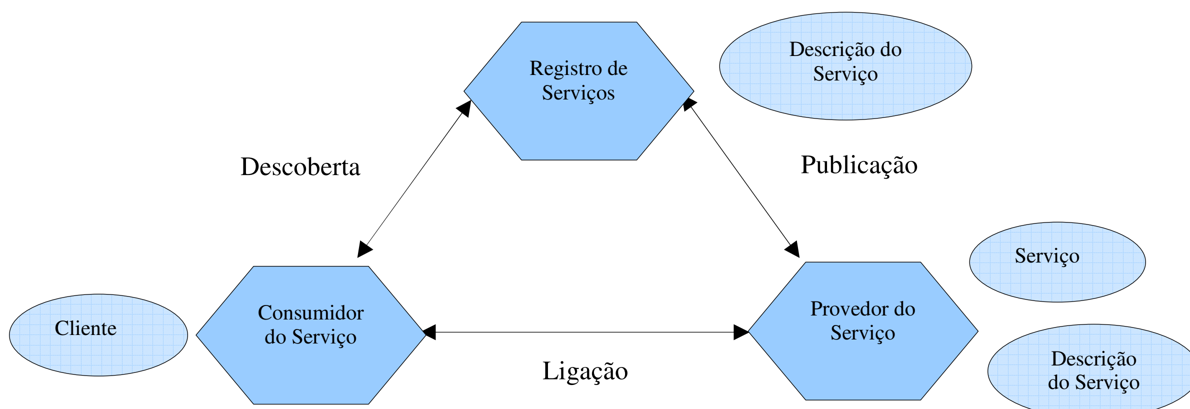

Figura 3.2: Pápeis e operações básicas nos serviços web (Kreger, 2001).

A segunda perspectiva apresenta os serviços *web* estruturados numa pilha, como pode ser visto na Figura 3.3 que segue abaixo. Conforme os serviços *web* evoluem, novas camadas são adicionadas e novas tecnologias são incorporadas a estas camadas.

Descoberta	**UDDI**
Descrição	**WSDL**
Mensagens XML	**XML-RPC, SOAP, XML**
Transporte	**HTTP, SMTP, FTP, BEPP**

Figura 3.3: Pilha de Protocolo dos Serviços Web (Cerami, 2002).

As camadas que compõem a pilha de serviços web são descritas a seguir:

- **Descoberta -** fornece o meio pelo qual as aplicações provedoras podem publicar as descrições de seus serviços e as aplicações consumidoras podem descobrir a descrição dos serviços disponibilizados. A tecnologia UDDI é a mais utilizada nessa camada.
- **Descrição -** esta camada permite a uma aplicação consumidora obter informações necessárias sobre o acesso e a utilização de um serviço web. Representa o meio pelo qual o consumidor obtém informações acerca do serviço. A tecnologia WSDL é o padrão para a descrição de serviço mais usada, no entanto, existem outras menos populares, tais como RDF e DAML.
- **Mensagens XML –** alguns autores chamam esta camada de empacotamento ou serialização. Ela permite a interação entre as aplicações, para tanto as mensagens devem ser empacotadas num formato que ambos os lados possam interpretar. O empacotamento das mensagens trocadas pelos serviços web é predominantemente baseado no protocolo SOAP.
- **Transporte -** é a camada responsável por transportar mensagens entre as aplicações através da internet. O protocolo HTTP é o mais difundido embora não ofereça suporte à comunicação assíncrona.

Em Snell *et. al*(2001) a pilha de serviços web é apresentada com cinco camadas, acrescentando ao modelo acima a camada de rede, a qual é responsável por fornecer a camada de transporte os serviços básicos de comunicação.

3.4 Tecnologias Associadas

Esta seção aborda as principais tecnologias utilizadas na implementação das operações básicas de serviços *web*.

3.4.1 XML

XML (*eXtensible Markup Language*) é uma meta-linguagem de marcação desenvolvida pelo consórcio W3C (*World Wide Web Consortium*) que combina a flexibilidade da SGML (*Standard Generalized Markup Language*) com a simplicidade da HTML (*Hyper Text Markup Language*), sendo considerada um padrão para a publicação e transferência de dados de diferentes domínios de aplicação na *web* (W3C).

O grande diferencial entre XML e HTML refere-se às *tags*, marcações de dados. Em XML as *tags* denotam uma interpretação semântica ao dado delimitado por ela. Já em HTML as *tags* apenas definem uma formatação para a apresentação dos dados.

Segundo W3C, entre os objetivos estabelecidos na especificação da linguagem XML, encontram-se as seguintes características:

- ser diretamente utilizável na Internet;
- ser legível por humanos;
- possibilitar um meio independente para publicação eletrônica
- permitir a definição de protocolos para troca de dados pelas empresas (independentemente da plataforma de *hardware* e *software*);
- facilitar às pessoas o processamento de dados pelo uso de *software* de baixo custo;
- facilitar a utilização de metadados que auxiliam na busca de informações;
- aproximar "produtores" e "consumidores" de informação.

No contexto de serviços *web*, Newcomer(2002) define a XML como o alicerce para a construção dos mesmos, visto que ela fornece um formato para a definição, armazenamento e transmissão dos dados trocados por eles.

3.4.2 SOAP

SOAP (*Simple Object Access Protocol)* é um protocolo de comunicação, criado pelo W3C, que permite a troca de informações entre aplicações. A comunicação é feita através de troca de mensagens em formato XML, incluindo os parâmetros de chamada e os resultados requeridos. Segundo Cerami (2002), o protocolo SOAP (versão 1.1) contém três partes principais:

- especificação do envelope SOAP – a qual define regras específicas para a transferência de dados entre computadores e inclui dados específicos da aplicação, tais como: nome do método a ser invocado, parâmetros do método ou valores de retorno. Também podem ser acrescentadas informações sobre quem pode processar o conteúdo do envelope e como a mensagem pode ser codificada em caso de falhas.
- regras de codificação dos dados – Para compartilhar dados, os computadores devem “concordar” com as regras de codificação e especificação de tipos de dados.

O protocolo SOAP apresenta um conjunto de convenções, baseadas na especificação XML Schema do W3C, para codificação dos tipos de dados.

- convenções de RPC – O protocolo SOAP pode ser usado por diversos sistemas de mensagens, incluindo os unidirecionais e bidirecionais. Isto permite que uma aplicação cliente especifique o nome de um método remoto, incluindo diversos parâmetros, e receba a resposta do servidor.

O protocolo SOAP apresenta as seguintes vantagens: pode atravessar *firewalls* com facilidade; os dados são estruturados em XML; pode ser usado em combinação com vários protocolos de transporte de dados, como HTTP, SMTP e FTP; mapeia satisfatoriamente para o padrão solicitação/resposta HTTP e; pode ser utilizado tanto de forma anônima como autenticada. Entretanto ele apresenta algumas desvantagens: falta de interoperabilidade entre as ferramentas de desenvolvimento do SOA, existem problemas de incompatibilidade entre diferentes implementações SOAP; mecanismos de segurança imaturos; não existe garantia quanto à entrega da mensagem; para enviar uma solicitação a vários servidores um cliente SOAP deve enviar a todos os servidores.

A estrutura da mensagem SOAP, ilustrada na Figura 3.4, é definida em documentos XML através dos seguintes elementos:

- Envelope: identifica o documento XML como uma mensagem SOAP e é responsável por definir o conteúdo da mensagem;
- Header: contém os dados do cabeçalho;
- Body: contém as informações de chamada e resposta ao servidor;
- Fault: contém as informações dos erros ocorridos no envio da mensagem. Esse elemento só aparece nas mensagens de resposta do servidor.

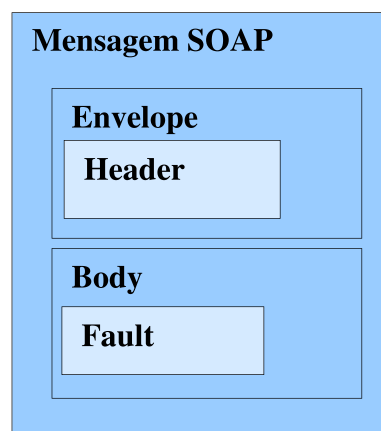

Figura 3.4: Elementos de uma mensagem SOAP (Cerami, 2002).

Nos Serviços Web o protocolo SOAP é usado para invocação dos mesmos e também pode ser utilizado nas operações de publicação e localização de serviços web nos registros UDDI.

3.4.3 WSDL

WSDL (*Web Service Description Language*) é uma linguagem que especifica como descrever serviços web através de XML. Através dela os serviços são descritos como um conjunto de operações de acessibilidade e mensagens.

De uma maneira mais simples podemos definir a WSDL como o contrato entre o consumidor do serviço e o provedor. Ela descreve quatro informações principais:

- informações de interface descrevendo as funcionalidades disponíveis;
- informações de tipos de dados para as mensagens de solicitação e resposta;
- informações obrigatórias sobre o protocolo de transporte usado;
- informações de endereço para localização dos serviços.

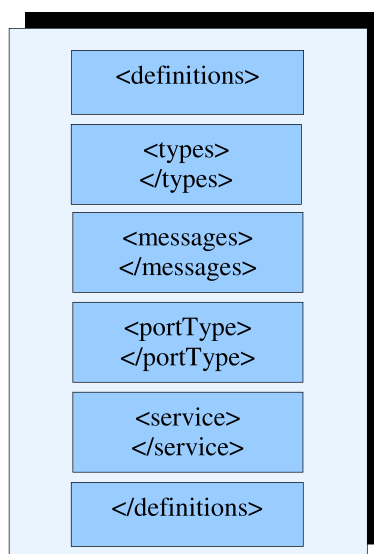

Figura 3.5: Elementos de um documento WSDL (Adaptado de Cerami, 2002).

A Figura 3.5 apresenta que um documento WSDL contém os seguintes elementos:

- Definição (*definitions*): principal elemento de um documento WSDL. Ele define o nome do serviço web e contém a declaração de todos os elementos do serviço.

- Tipo (*types*): esse elemento descreve todos os tipos de dados presentes na mensagem.
- Mensagem (*message*): define os tipos de dados utilizados nas mensagens trocadas.
- Tipo de porta (*portType*): esse elemento descreve o conjunto abstrato das operações disponibilizadas por um serviço web.
- Ligação (*binding*): define o protocolo de comunicação e o formato dos dados esperados para a invocação de cada operação do serviço web.
- Serviço (*service*): nesse elemento se encontra o endereço da implementação do serviço na rede, ou seja, contém a informação de para onde encaminhar solicitação do serviço.

3.4.4 UDDI

UDDI (*Universal Description, Discovery and Integration*) fornece um mecanismo para publicação e descoberta de informações sobre serviços web. Ele atua como um mediador, permitindo que os clientes requisitantes encontrem um fornecedor do serviço apropriado. O padrão UDDI é uma iniciativa da indústria, e tem como objetivo criar uma plataforma independente e aberta para descrever, descobrir e integrar serviços.

Um serviço UDDI registra informações sobre serviços numa estrutura semelhante a uma lista telefônica, tal estrutura divide-se em três partes:

- Páginas Brancas (*white pages*): incluem informações gerais (identificação, descrição, contato, endereço e telefone) sobre uma empresa. Tais informações possibilitam a descoberta de um serviço web pela identificação da empresa.
- Páginas Amarelas (*yellow pages*): descrevem o serviço utilizando diferentes classificações, permitindo assim a descoberta de um serviço pela sua categorização.
- Páginas Verdes (*green pages*): apresentam informações técnicas sobre o serviço web e sua localização.

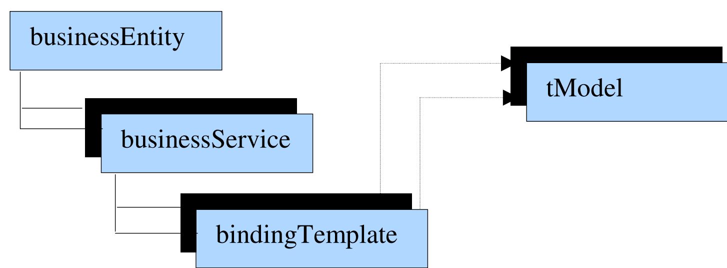

Figura 3.6: Estrutura do UDDI (Cerami, 2002).

O UDDI inclui um Schema XML que descreve quatro especificações principais (Figura 3.6):

- *Business Entities*: esse elemento apresenta informações sobre a empresa que publica o serviço *web*. Esta informação contém o nome da empresa, descrição, endereço e contato para informações.
- *Business Services*: esse elemento inclui informações sobre um único serviço *web* ou um grupo de serviços *web* relacionados, oferecendo informações sobre a descrição de cada serviço em termos de negócios.
- *Binding Template*: esse elemento apresenta informações sobre como obter o acesso e quais os pontos de acesso a um serviço *web*.
- *TModel*: esse elemento contém informações que descrevem uma especificação técnica do serviço.

Com esta estrutura, a UDDI consegue definir a maneira de publicar e descobrir as informações sobre os serviços *web*. A Figura 3.7 ilustra como um repositório UDDI é usado na publicação e na descoberta de um serviço *web*.

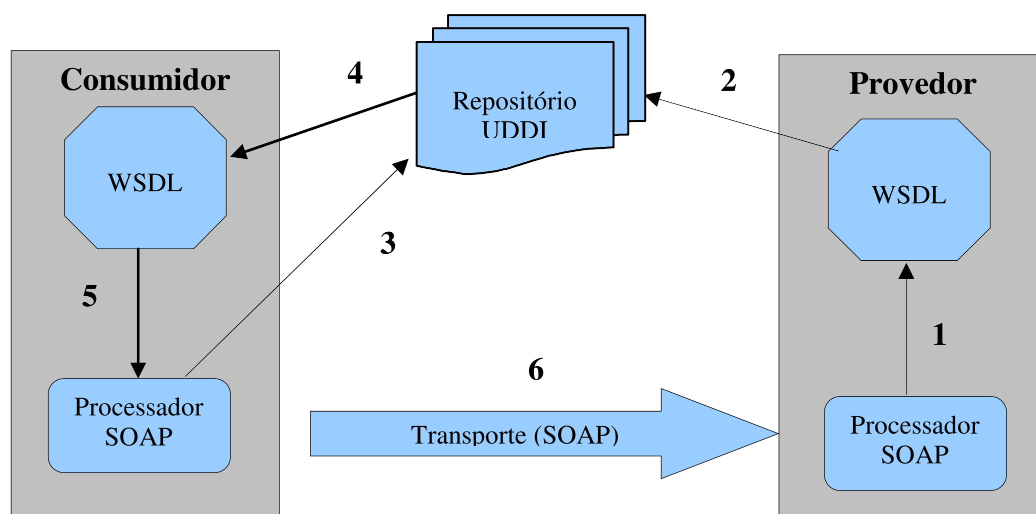

Figura 3.7: UDDI na publicação e descoberta de serviço web (Adaptado de Newcomer, 2002).

3.5 Aplicações dos Serviços Web

Arsanjani (2003) destaca que um dos principais objetivos dos serviços web é o de prover a interoperabilidade entre diferentes sistemas, independente da plataforma e linguagem de programação utilizadas por eles, ou seja, eles devem facilitar a tarefa de interligação das aplicações diminuindo custos e aumentando a produtividade.

O modelo dos serviços web pode ser utilizado para promover o reuso dos componentes existentes nos sistemas integrados, em que cada componente pode representar um serviço distinto, sem sua lógica estar "atrelada" a um único projeto de integração, e seus componentes podem participar de múltiplos sistemas sem serem alterados.

Os serviços web devem ser bem estruturados para que reduzam o risco de exposição indevida de informações dos sistemas e reutilizem o máximo possível do código existente, diminuindo a redundância do código e, por conseguinte, os custos do projeto.

O campo de aplicação dos serviços web é amplo. Essa tecnologia pode ser usada em qualquer tipo de aplicação que necessite de interoperabilidade, compatibilidade com sistemas legados, lidar com domínios administrativos distintos, entre outras coisas.

4. INTEGRAÇÃO DE ASPECTOS E SERVIÇOS WEB

4.1 Introdução

A integração da Programação Orientada a Aspectos (POA) e tecnologia dos Serviços Web é uma tarefa desafiadora, visto que os objetivos dos dois paradigmas são conflitantes. As aplicações orientadas a serviço são de natureza heterogênea e distribuídas enquanto que as tecnologias de suporte a POA se encontram associadas a uma linguagem de programação ou plataforma de execução específica.

Silva (2006) apresenta as estratégias de integração desses dois paradigmas em quatro grupos: integração em nível de aplicação; integração em nível de desenvolvimento; integração em nível de composição de serviços; e integração em nível de descrição de serviços. Além das estratégias descritas no trabalho de Silva (2006), é apresentado o grupo de integração em nível de modelo. Nas seções seguintes aborda-se com mais detalhes tais estratégias de integração.

4.2 Integração em Nível de Aplicação

A integração em nível de aplicação modifica a aplicação propriamente dita e apresenta dependência em relação à linguagem de programação e o ambiente de execução usado pela aplicação. Nessa categoria são apresentados dois trabalhos, um proposto por Henkel et al.(2005) e outro por Verheeck e Cibrán(2003).

Em Henkel et. al(2005), a integração é proposta na implementação de requisitos não funcionais que passam a existir quando serviços *web*, anteriormente disponíveis apenas internamente na organização, precisam ser utilizados por organizações externas. A idéia central é satisfazer tais requisitos sem precisar re-escrever ou modificar grande parte da implementação do serviço. O trabalho centra-se nos requisitos não funcionais de segurança, métricas de qualidade de serviço e monitoramento de desempenho.

A solução de Verheecke e Cibrán (2003) propõe uma camada de gerenciamento de serviços, chamada WSLM (*Web Services Management Layer*) para modularizar aspectos de seleção de serviços, visto que o código para seleção de serviços *web*, em uma aplicação consumidora de serviços, é replicado em cada ponto em que a funcionalidade do serviço é requerida.

A solução apresentada por Henkel et. al (2005) é dependente do uso da linguagem Java para a implementação das aplicações provedoras dos serviços e da linguagem AspectJ para a implementação dos aspectos. A proposta de Verheecke e Cibrán (2003) depende da linguagem JasCo para a implementação dos aspectos e de tecnologias Java para a implementação das aplicações consumidoras.

4.3 Integração em Nível de Desenvolvimento

A integração em nível de desenvolvimento é realizada através do uso ou modificação de recursos providos pela plataforma de desenvolvimento sobre a qual as aplicações baseadas em serviços web são implementadas. Nessa categoria são apresentados três trabalhos, Lafferty e Cahill (2003), Verspecht et al. (2003) e Baligand e Monfort (2004).

Lafferty e Cahill (2003) apresentam um combinador de aspectos, denominado Weave.NET, desenvolvido como um componente da plataforma .NET, que utiliza arquivos de especificações de aspectos em XML. A infra-estrutura de linguagem intermediária comum de .NET é usada para combinar, em tempo de carga, componentes construídos em qualquer linguagem de programação, suportada nesta plataforma a aspectos também construídos em qualquer uma destas linguagens. Uma vantagem desta abordagem é a riqueza de seu modelo de aspectos. No entanto, a solução depende do uso da plataforma .NET, ou seja, os aspectos podem ser combinados de forma independente de linguagem desde que as linguagens escolhidas para implementar os aspectos e os componentes sejam suportadas pela plataforma.

Verspecht et al. (2003) apresentam a linguagem orientada a aspectos JasCo.NET derivada da linguagem JasCo, para fornecer uma extensão orientada a aspectos para a plataforma .NET. Em termos de implementação, JASCO.NET introduz um novo modelo de componentes para serviços web à plataforma .NET. Uma vantagem dessa abordagem é o fato de Jasco.NET oferecer combinação dinâmica de aspectos a serviços web. Da mesma forma que a proposta de Lafferty e Cahill (2003), esta solução possui uma dependência da plataforma .NET.

Em Baligan e Monfort (2004) é proposta uma abordagem orientada a aspectos, que utiliza soluções de código aberto disponíveis para Java, para combinar aspectos de segurança à implementação dos serviços segundo os requisitos das políticas. O arcabouço AXIS é utilizado para encapsular, na forma de aspectos, interesses não-funcionais

inerentes à adaptação de serviços web. A grande vantagem dessa solução centra-se no fato de fornecer uma integração dinâmica e bastante flexível, no entanto, apresenta uma dependência em relação ao uso do arcabouço AXIS e da linguagem Java, limitando assim a implementação dos aspectos e componentes a essas duas tecnologias.

4.4 Integração em Nível de Composição de Serviços

A integração em nível de composição de serviços realiza a integração da noção de aspectos às tecnologias que tratam da composição de serviços. Nesse grupo são mencionados três trabalhos: Courbis e Finkelstein (2005); Charfi e Mezini (2004) e; Cottenier et al. (2004).

Em Courbis e Finkelstein (2005) é apresentada uma solução para a integração dinâmica de aspectos utilizando a linguagem BPEL4WS. É proposto um novo interpretador para BPEL4WS, que usa Xpath como linguagem para especificar os pontos de junção dos aspectos, os quais podem ser associados aos elementos do fluxo de atividade que define os processos BPEL4WS, e Java como linguagem para implementar os adendos. O ponto forte deste trabalho é a independência das tecnologias, visto que a integração é proposta em nível de composição. Como desvantagens, essa solução apresenta um modelo de combinação cujo escopo restringe-se a eventos de interação externos às aplicações e querer que as aplicações alvo dos aspectos sejam implementadas como processos BPEL4WS.

No trabalho de Charfi e Mezini (2005) é apresentada uma extensão orientada a aspectos da linguagem de composição de serviços web BPEL4WS, a AO4BPEL. A AO4BPEL objetiva melhorar a modularidade das especificações de composição de serviços web em BPEL4WS, suportar adaptações dinâmicas dessas composições. Tal solução apresenta a limitação de que só pode ser aplicada no contexto de aplicações baseadas em serviços web implementadas ou executadas como processos BPEL4WS.

Outra solução desse grupo é o trabalho de Cottenier et al (2004) que apresenta a arquitetura CASS (*Context Aspect Sensitive Service),* a qual utiliza tecnologias da POA para integrar aspectos dinâmicos ao desenvolvimento orientado a serviços, e desse modo fornecer um ambiente de composição descentralizado e com sofisticados mecanismos de refinamento de serviços web. Os autores propõem um modelo de aspectos para implementar orquestrações de serviços declarativas, tais como BPEL4WS, instanciando

aspectos de colaboração através de descritores de implantação. Como as demais soluções desse grupo, a de Cottenier et al (2004) apresenta a vantagem de ser independente da linguagem e da plataforma utilizadas na implementação das aplicações alvo do processo de composição. E como desvantagem, a arquitetura requer que as aplicações e os aspectos sejam todos implementados e executados sob o controle de seu ambiente de execução.

4.5 Integração em Nível de Descrição de Serviços

Nesse grupo há apenas um trabalho, o de Singh et al(2005), que estende a linguagem de descrição de serviços web, a WSDL, de modo a enriquecer a descrição dos serviços com informações sobre aspectos a serem integrados às aplicações consumidoras.

Em Singh et al (2005) é proposta a arquitetura AOWS (*Aspect-Oriented Web Service*) que estende os mecanismos WSDL e UDDI com construções para especificar, na forma de aspectos, requisitos funcionais e não-funcionais a serem descobertos e integrados dinamicamente às aplicações consumidoras.

A grande vantagem dessa solução é a identificação e descrição dos aspectos desde a fase de projeto da interface dos serviços, a possibilidade de descobrir e integrar aspectos às aplicações consumidoras dinamicamente e o suporte para a análise de propriedades dos componentes das aplicações através de um mecanismo de especificação formal.

Como desvantagem a solução requer mudanças significativas em componentes básicos da arquitetura orientada a serviços (WSDL e UDDI), o que acaba restringindo, assim, os seus benefícios às aplicações desenvolvidas utilizando a arquitetura AOWS.

4.6 Integração em Nível de Modelo

Do mesmo modo que na integração em nível de descrição de serviços este grupo apresenta apenas um trabalho, o de Ortiz *et. al* (2005), o qual fornece uma abordagem MDA (*Model Driven Architecture*) e aplica técnicas orientadas a aspectos para a modelagem dos Serviços Web.

Em Ortiz *et. al* (2005) propõe-se a modelagem de serviços web, compondo-os de forma bem modularizada, mantendo a decomposição lógica das unidades através do uso de um perfil para modelar aspectos e um outro para modelar serviços web, independentemente da plataforma. Posteriormente o modelo independente é transformado num modelo específico para uma plataforma.

O trabalho apresenta duas alternativas distintas para o PSM (*Platform Specific Model*). A primeira alternativa executa a transformação de modo que não permaneça aspecto algum. A segunda mantém os aspectos deixando-os abstratos ou traduzindo-os em uma linguagem orientada a aspectos.

A grande vantagem do trabalho é a definição do PIM (*Platform Independently Model*) no qual são aplicadas as técnicas orientadas a aspectos e a análise de duas abordagens distintas para o modelo específico dependendo do nível em que a composição será executada, no projeto ou na implementação. As limitações do trabalho referem-se às regras de transformação para traduzir do PSM para o código, as quais devem ser analisadas mais profundamente.

4.6.1 MDA

A MDA é uma arquitetura conceitual para desenvolvimento de software que visa separar decisões orientadas ao negócio de decisões de plataforma permitindo assim maior flexibilidade durante as fases de especificação e de desenvolvimento de sistemas. A idéia central da MDA é a criação de diferentes modelos com diferentes níveis de abstração e a ligação destes modelos através de refinamentos e transformações. Alguns desses modelos existirão independente da plataforma, enquanto outros serão específicos para uma plataforma. Os três modelos a seguir compõem os principais elementos de MDA:

- CIM (*Computational Independent Model*): descreve o domínio da aplicação, sem se ater aos detalhes da estrutura do sistema. É importante para a ajuda no entendimento do problema, como também é uma fonte de vocabulário a ser usado nos demais modelos.

- PIM (*Plataform Independent Model*): descreve o sistema, porém não apresenta os detalhes da tecnologia que será usada na implementação. O PIM oferece especificações formais da estrutura e funcionalidade do sistema, abstraindo-se de detalhes técnicos.

- PSM (*Plataform Specific Model*): combina a especificação do modelo PIM com detalhes específicos de uma determinada plataforma. A transformação de modelos é a chave principal desta abordagem e consiste no processo de converter um modelo em outro modelo do mesmo sistema. A idéia principal é construir modelos no seu mais alto nível de abstração e transformá-los em modelos com baixa abstração, de forma automática ou semi-automática, facilitando e tornando mais rápido o processo de desenvolvimento.

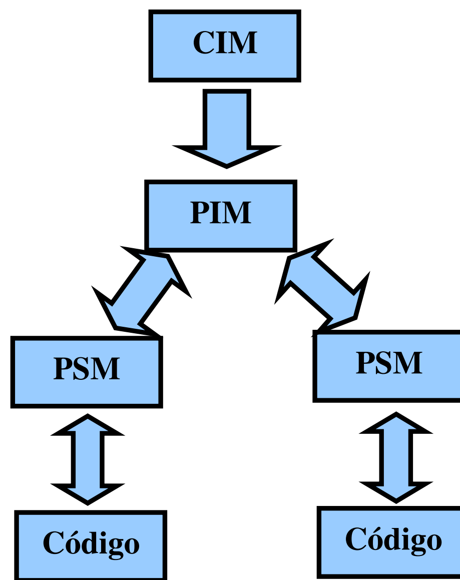

Figura 4.1: Relacionamento entre os modelos da MDA.

Devido as suas características, pode-se dizer que a MDA oferece os seguintes benefícios:

- produtividade: a transformação do PIM para o PSM precisa ser definida uma única vez e pode ser aplicada no desenvolvimento de diversos sistemas. Com isso, tem-se uma redução no tempo de desenvolvimento.

- portabilidade: alcançada através do foco dado no desenvolvimento do PIM, que é independente de plataforma. Um mesmo PIM pode ser automaticamente transformado em vários PSMs de diferentes plataformas.

- interoperabilidade: diferentes PSMs gerados a partir de um mesmo PIM podem conter ligações entre eles, denominadas em MDA de pontes. Quando PSMs são gerados em diferentes plataformas, eles não podem se relacionar entre si. É necessário então transformar os conceitos de uma plataforma para conceitos da outra plataforma. MDA endereça este problema gerando não somente PSMs, como também as pontes necessárias entre eles. Uma ponte pode ser construída através das especificações técnicas das plataformas referentes aos modelos PSMs e de como os elementos existentes no modelo PIM foram transformados nos modelo PSMs.

4.7 Análise das Estratégias de Integração

As estratégias propostas nos trabalho de Henkel et. al (2005) e Verheecke e Cibrán (2003) são dependentes de uma linguagem de programação tanto para a implementação das aplicações provedoras e consumidores quanto para a implementação dos aspectos.

Os trabalhos apresentados por Lafferty e Cahill (2003) e Verspecht et al. (2003) se encontram atrelados a uma plataforma de execução específica, no caso .NET. A proposta

de Baligan e Monfort (2004) limita a implantação de aspectos e componentes a uma linguagem e ao uso de um arcabouço.

As propostas de Courbis e Finkelstein (2005) e de Charfi e Mezini (2005) são restritas no sentido que as aplicações baseadas em serviços web devem ser implementadas ou executadas como processos BPEL4WS. Em Cottenier et al (2004) a limitação refere-se ao fato de que as aplicações e os aspectos devem ser implementados e executados sob o controle do seu ambiente de execução.

O trabalho de Singh et. al (2005) restringe os seus benefícios às aplicações desenvolvidas utilizando a arquitetura AOWS. A proposta de Ortiz *et. al* (2005) apresenta limitações em relação às regras de transformação para traduzir do PSM para o código.

Todos os trabalhos abordados acima, exceto o de Ortiz et. *al* 2005, apresentam dependência em relação a uma determinada linguagem de programação ou plataforma de execução, tanto para implementar as aplicações consumidoras e provedoras de serviços como para implementar os aspectos. Essa dependência é indesejável pelos seguintes motivos:

- restringe o leque de tecnologias de implementação disponível para os programadores;
- vai de encontro à própria natureza heterogênea e fracamente acoplada que caracteriza a COS.

Desse modo, uma solução mais eficaz para realizar a integração dessas duas tecnologias deve ter a característica de independência de tecnologia de implementação como um princípio básico de projeto.

Das estratégias expostas acima a de Ortiz *et. al* (2005) é a que apresenta independência em relação a tecnologia de implementação e/ou plataforma de execução, portanto foi adotada para o desenvolvimento do estudo de caso. Tal proposta apresenta uma abordagem MDA em que são criados diferentes modelos com diferentes níveis de abstração. A aplicação da MDA confere alguns benefícios em relação à produtividade, portabilidade e interoperabilidade.

A grande vantagem da proposta de Ortiz *et. al* (2005) é a definição do PIM (*Platform Independently Model*) no qual são aplicadas as técnicas orientadas a aspectos e a análise de duas abordagens distintas para o modelo específico dependendo do nível em que a composição será executada, no projeto ou na implementação.

A Figura 4.2 apresenta os novos estereótipos definidos para modelar os cinco tipos de elementos orientados a aspectos.

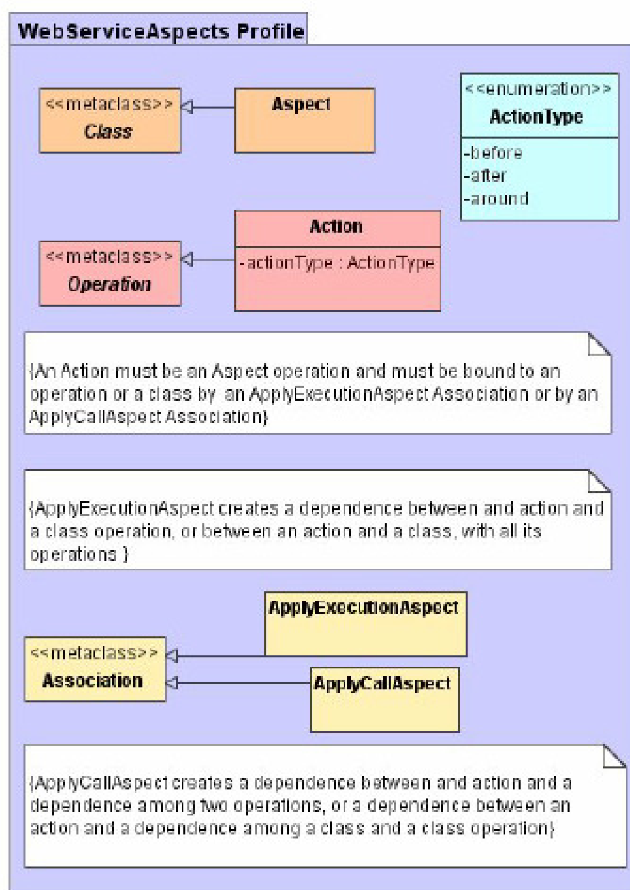

Figura 4.2: Definição da notação Aspect-Oriented UML Profile (Ortiz *et. al* ,2005) .

Devido ao fato que serviços são ‘caixas pretas’, haverá dois possíveis tipos de pontos de interação em aplicações Web Service: primeiro, o ponto no qual uma operação de serviço será executada; segundo, quando um cliente (que também pode ser outro serviço) invoca um operador específico no serviço designado.

Os primeiros serão chamados de ‘aspecto de execução’ pois são ativados quando o método de serviço for executado; por outro lado, os segundos serão chamados ‘aspectos de chamadas’ desde que eles sejam ativados quando uma operação específica for invocada pelo cliente. Então, estes dois tipos de ponto na execução de nossa aplicação formam o modelo de pontos de junção.

Para identificar os de pontos de corte, foram somadas duas associações de estereótipos. Nos ‘aspectos de execução’, a associação *ApplyExecutionAspect* liga a ação que implementa o comportamento novo a ser incluído à operação cuja execução está

sendo interceptada; esta associação representa o ponto de corte neste tipo de aspecto. Para conjuntos de pontos de junção de *callAspects*, soma-se a associação *ApplyCallAspect* que liga *Action* à invocação de um método de serviço de um cliente.

Os adendos são representados pelo estereótipo *Action* que podem ser aplicados às operações e apresentam um Tag para indicar o tipo de ação (*before*, *after* ou *around*), isto é, para indicar em que ponto o comportamento será interceptado.

O estereótipo aspecto contém *Actions*, que são ligadas (quem?? Actions???) a um método do serviço pela associação de *ApplyExecutionAspect* ou a uma dependência entre o cliente e um método do serviço pela associação *ApplyCallAspect*.

No Quadro 4.1 é apresentada uma comparação entre as duas abordagens propostas no trabalho de Ortiz *et. al* (2005).

Quadro 4.1: Comparação entre PSM, PIM sem aspectos e PIM com aspectos.

Elemento	PIM	PSM sem aspectos	PSM com aspectos (usando AspectJ)
Interface do serviço web	Representado pelo estereótipo <<*web service interface* >> .	Representado por <<j*ava web service interface* >> .	Representado por <<j*ava web service interface* >> .
Modelo de Pontos de Junção	Não é representado explicitamente no diagrama.	Não aparece nesse modelo.	Não aparece nesse modelo.
Pontos de corte	Representado pelos estereótipos <<*ApplyExecutionAspect*>> e <<*ApplyCallAspect*>> .	Pode ser representado como o ponto da conexão entre serviços e aspectos.	Representado por <<*ExecutionPointcut*>> e <<*CallPointcut*>>.
Aspectos	Representado pelo estereótipo <<*aspect*>>.	Transformado em parte dos comportamentos da especificação dos serviços e nas relações e lógica de interação entre o serviço nomeado.	Representado pelo estereótipo <<*aspect*>> .
Ligação Aspecto – Serviço	Representado pelos estereótipos <<*ApplyExecutionAspect*>> e <<*ApplyCallAspect*>	Transformado em parte dos comportamentos da especificação dos serviços e nas relações e lógica de	Representado por <<*ExecutionPointcut*>> e <<*CallPointcut*>>.

	> com ligação a ação por <<*Web Service Interface*>>	interação entre o serviço nomeado.	

5. AMBIENTE DiSEN

5.1 Introdução

A necessidade para que se faça realmente uma engenharia do produto de software (planejamento, acompanhamento, execução e controle) é cada vez maior. O uso de ferramentas isoladas oferecem apenas soluções parciais, o que se deseja é usar ferramentas de apoio ao longo de todo o desenvolvimento de software (TRAVASSOS, 1994 apud MIAN et. al, 2001). Diante de tal cenário, a demanda por Ambientes de Desenvolvimento de Software (ADS) tornou-se crescente, visto que os mesmos buscam combinar técnicas, métodos e ferramentas para apoiar na construção de produtos de software, abrangendo todas as atividades inerentes ao processo, tais como: planejamento, gerência, desenvolvimento e controle da qualidade.

Moura (1992, apud Pascutti 2002) define um ADS como sendo um sistema computacional que oferece suporte para o desenvolvimento, reparo e melhorias em software e para o gerenciamento e controle dessas atividades. Para tanto, um ADS é composto por um repositório com todas as informações acerca do desenvolvimento ao longo do seu ciclo de vida e por ferramentas que oferecem apoio às várias atividades, técnicas e gerenciais, no projeto.

O crescente número de empresas que utilizam processos de desenvolvimento de software de maneira distribuída impulsiona o desenvolvimento de projetos de pesquisa que objetivam a criação de ambientes de desenvolvimento de distribuído de software (WIESE *et. al*, 2005).

5.2 Arquitetura do DiSEN

O DiSEN é um ambiente que oferece suporte ao desenvolvimento distribuído de software. Em sua arquitetura podem ser identificadas as características necessárias em ambientes de desenvolvimento de software enumeradas por Carmel (1999, apud Wiese *et. al*, 2005), que são: arquitetura de produtos; suporte a metodologias de desenvolvimento; suporte a gerência de recursos; suporte a persistência de dados; gerência de projetos/processos; infra-estrutura de comunicação e tecnologias de colaboração.

O DiSEN objetiva prover o suporte necessário para o desenvolvimento de software distribuído, podendo a equipe estar localizada em pontos geográficos distintos e trabalhando de forma colaborativa com uma metodologia para o desenvolvimento distribuído de software.

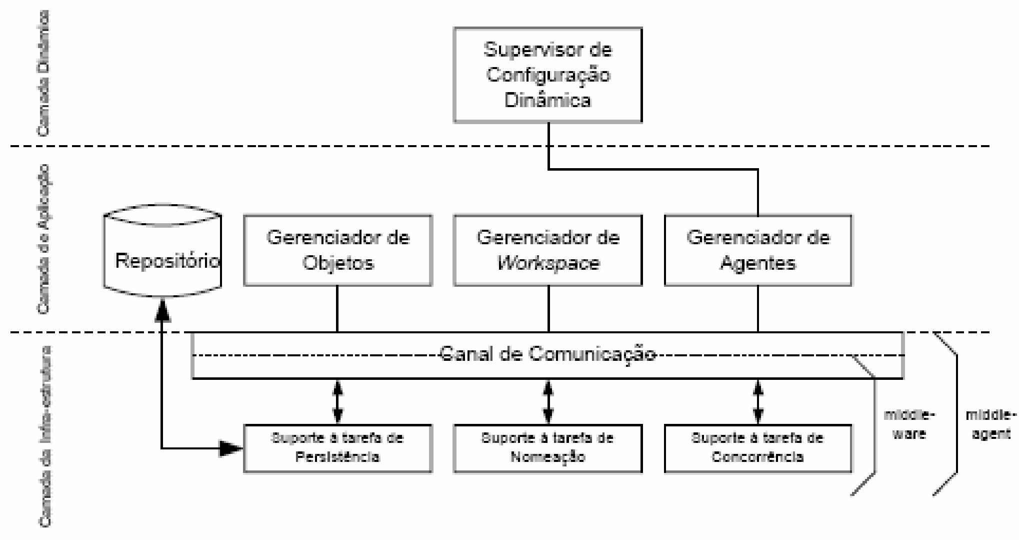

Figura 5.1: Arquitetura do DiSEN (Pascutti , 2002).

A Figura 5.1 ilustra a arquitetura do DiSEN proposta por PASCUTTI (2002), composta por sete elementos os quais se encontram distribuídos em três camadas: Dinâmica, Aplicação e Infra-estrutura.

A camada dinâmica é responsável pela inserção, remoção e configuração dos componentes de software e serviços em tempo de execução. Nessa camada encontra-se o elemento supervisor de configuração dinâmica que pode ser subdividido em: Supervisor de Configuração, que tem como tarefa básica o controle e gerenciamento da configuração do ambiente, e o Supervisor de Serviços, que é o responsável por descobrir, alocar e unir, dinamicamente, serviços distintos.

A camada de aplicação oferece suporte às metodologias de desenvolvimento de software, o repositório para armazenamento dos artefatos e informações necessários ao ambiente e os gerenciadores de objetos, *workspaces* e agentes.

Uma das principais tarefas do ambiente de desenvolvimento de software distribuído, é armazenar, estruturar e controlar, além do próprio software, os artefatos gerados durante o desenvolvimento. Por natureza, tais artefatos são mais complexos que os

itens tratados por sistemas de banco de dados tradicionais, visto que são mais estruturados e requerem operações complexas. O gerenciador de objetos subdivide-se nos seguintes gerenciadores:

- Gerenciador de Acesso: responsável pelo controle e gerenciamento das regras de acesso dos usuários às informações do ambiente.
- Gerenciador de Atividades: responsável pelo gerenciamento das atividades que compõem um processo de software.
- Gerenciador de Recursos: oferece suporte ao gerenciamento de recursos (materiais, ferramentas ou recursos humanos). O gerenciador é responsável pelo gerenciamento de todos os recursos, mantendo a atualização do estado de cada um, bem como a busca por um recurso similar quando uma determinada atividade requisitar um recurso e este estiver em uso ou bloqueado.
- Gerenciador de Artefatos: Um artefato é um produto criado ou modificado durante um processo. Ele é resultado de uma atividade (saída) e posteriormente pode ser usado como matéria-prima (entrada) para uma outra atividade com o objetivo de gerar novos artefatos.
- Gerenciador de Projetos: responsável por gerenciar os projetos criados, bem como as métricas e estimativas associadas a cada projeto.
- Gerenciador de Processos: esse gerenciador preocupa-se em analisar e verificar modelos de processo, obtendo informações durante a execução desses para que possam ser usados posteriormente.
- Gerenciador de Versão e Configuração: responsável pelo gerenciamento das versões dos artefatos.

Como as versões de um projeto são imutáveis, não é permitido que os desenvolvedores trabalhem diretamente no repositório. Eles têm que tirar uma cópia do documento modificá-la e adicionar a cópia modificada no repositório. Desse modo, o gerenciador de *workspace*, representado na Figura 5.2, oferece funcionalidades para criar um ou mais *workspaces* de um repositório. Já o gerenciador de agentes é o responsável pela criação, registro, localização, migração e destruição de agentes.

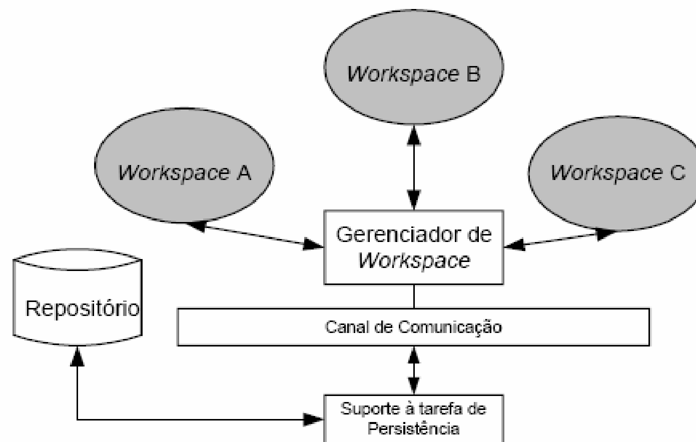

Figura 5.2: Representação do gerenciador de *workspace* (Pascutti , 2002).

A camada de Infra-estrutura é o pilar da arquitetura, provê suporte às tarefas de persistência, nomeação e concorrência, por exemplo. Nela encontra-se o canal de comunicação que é o elemento responsável pela troca de informações entre os elementos do DiSEN e pela comunicação com a camada adjacente.

Segundo Wiese et. al (2005), as características enumeradas por Carmel (1999) são atendidas pelo DiSEN do seguinte modo:

- Arquitetura de Produtos: Componentes/MDA
- Suporte a Metodologias de Desenvolvimento: a camada de aplicação oferece suporte para qualquer metodologia de desenvolvimento.
- Suporte à Gerência de Recursos: Provido pelo Gerenciador de Recursos.
- Gerência de Artefatos: Provido pelo Gerenciador de Artefatos em conjunto com o Módulo de Suporte à Tarefa de Persistência.
- Suporte a Persistência de Dados: Provido pelo Módulo de Suporte à Tarefa de Persistência.
- Gerência de Projetos/Processos: Provido pelos seguintes gerenciadores: Gerenciador de Processos, Gerenciador de Projetos e Gerenciador de Atividades.
- Infraestrutura de Comunicação: Canal de Comunicação.
- Tecnologia de Colaboração: Provido pelo Gerenciador de *Workspaces*.

5.3 Modelo SPC

O modelo SPC (Sincronização, Percepção, Comunicação) propõe um padrão na cooperação entre usuários de workspaces compartilhados para o DiSEN. Segundo Pozza (2005), este modelo poderá ser utilizado por metodologias e ferramentas que façam parte de ambientes de desenvolvimento distribuído de software e que sejam apoiadas por computador. A Figura 5.3 ilustra o modelo SPC e o relacionamento das características de Comunicação, Percepção e Sincronização.

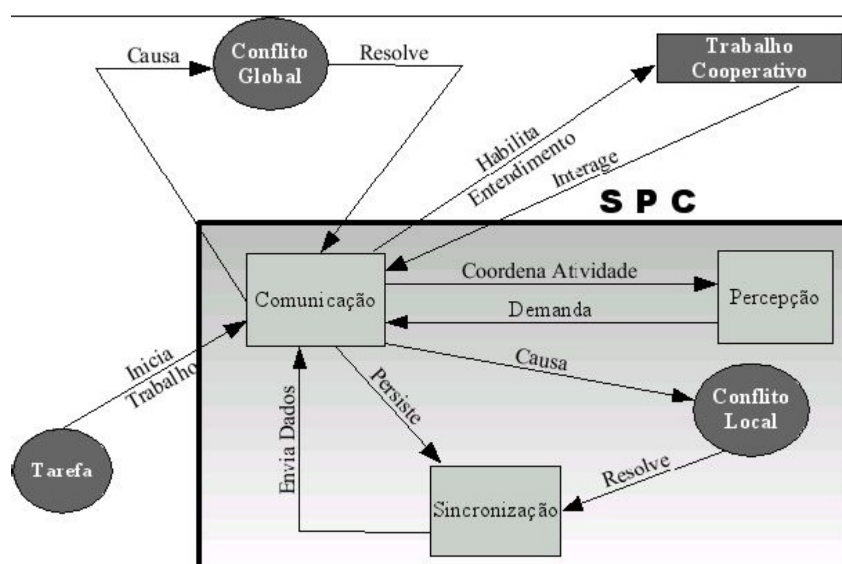

Figura 5.3: Proposta do Modelo SPC apresentada por Pozza (Pozza , 2005).

Figura 5.4: Diagrama de Atividades do SPC (Pozza , 2005).

A Sincronização é uma característica que permite às pessoas trabalharem concorrentemente em um projeto ou atividade de forma coesa. Ela oferece suporte para a persistência dos dados que estão sendo compartilhados no *workspace*, verificando sua consistência. A sincronização segmenta-se nos seguintes prospectos:

- Persistência: apresentada nos trabalhos de Borba (2002 apud Pozza 2005), Mangan (2003 apud Pozza 2005) e Ribeiro (1997 apud Pozza 2005), que associam repositórios com *workspace*, apoiando quais são as necessidades de armazenamento de dados. A fim de evitar que o GW busque constantemente dados no repositório central, o modelo SPC propõe a persistência de informações locais. Na proposta da arquitetura DiSEN, o GW transfere do repositório a configuração que é necessária para o projeto e as

partes que o usuário pode ter acesso, evitando sobrecarregar o canal de comunicação. A cópia local agiliza o processo de cooperação e troca de informação entre *workspaces*.

- Controle de Transação: A Sincronização do modelo SPC deve possuir um controle que garanta as propriedades ACID (atomicidade, consistência, isolamento e durabilidade) quando os dados distribuídos estiverem sendo trabalhados. Em Borba (2002 apud Pozza 2005) é apresentado um controle de transação apoiando esse prospecto que trabalha conjuntamente com uma classe que utiliza o *pattern Facade*. O controle de transação é abordado como um prospecto que implementa três *advices*: o início, *commit* e *rollback* de uma transação.

- Atualização: Notificações são enviadas para os diversos usuários por e-mail, mensagens ou outros meios para avisar sobre alterações ocorridas em um ou mais artefatos. Essas notificações objetivam propiciar aos usuários informações para que os artefatos sejam mantidos consistentes. O controle de concorrência pode utilizar-se de três políticas: otimista, pessimista e fragmentada. O uso dessas políticas dependerá do usuário do *workspace* prioritário, estabelecendo qual a melhor política a ser adotada.

A Percepção, definida como a contextualização das atividades individuais por meio da compreensão das ações realizadas por outras pessoas (Borges 1995, apud Pozza 2005), apresenta duas formas, uma que se refere a noção de presença dos participantes e a outra que se relaciona ao estado (ocupado, ausente, trabalhando, etc.) dos participantes na realização de uma dada atividade ou trabalho. A Percepção contem os seguintes prospectos:

- Métodos: Oferecem suporte para que um usuário possa contextualizar as atividades executadas por membros de equipes com atividades correlatas;

- Categorias (quem, o que, onde, quando, como, quanto): Este prospecto envolve sabero que tem ocorrido, o que está acontecendo e o que poderá vir a acontecer com artefatos compartilhados e no ambiente de trabalho de um grupo. Além disso, pelas informações oriundas da percepção, é possível responder a questões como: Quem realizou uma dada tarefa?, Quando?, Quem está trabalhando agora?, Em que está trabalhando?, Quem é o responsável por uma tarefa? O que ainda falta ser feito?.

Outra característica do modelo SPC é a Comunicação que apóia a troca de dados entre os usuários dos *workspaces*. A comunicação pode ser ponto a ponto, multiponto (*multicast*) e difusiva (*broadcast*). Ela apresenta os seguintes prospectos:

- Flexibilidade na interação: esse prospecto objetiva possibilitar a comunicação entre *workspaces* do modo mais transparente possível. A flexibilidade que será evidenciada

é como se dará a troca de dados entre a Comunicação, a Percepção e a Sincronização e, por conseqüência, entre *workspaces* envolvidos em um trabalho cooperativo.

- Síncrono e assíncrono: Para o trabalho cooperativo, fatores como tempo e espaço, devem ser considerados para o desenvolvimento de um *workspace*. A defasagem das ações de assincronismo e as complexidades das ações síncronas são consideradas no modelo SPC com a Sincronização e a Percepção.

O diagrama de classes para o *framework* de cooperação entre usuários de *workspaces* compartilhados é apresentado na Figura 5.5. Ele mostra o relacionamento entre as operações descritas neste capítulo e objetiva fundamentar a proposta de um *framework* para a cooperação entre usuários de *workspaces* compartilhados.

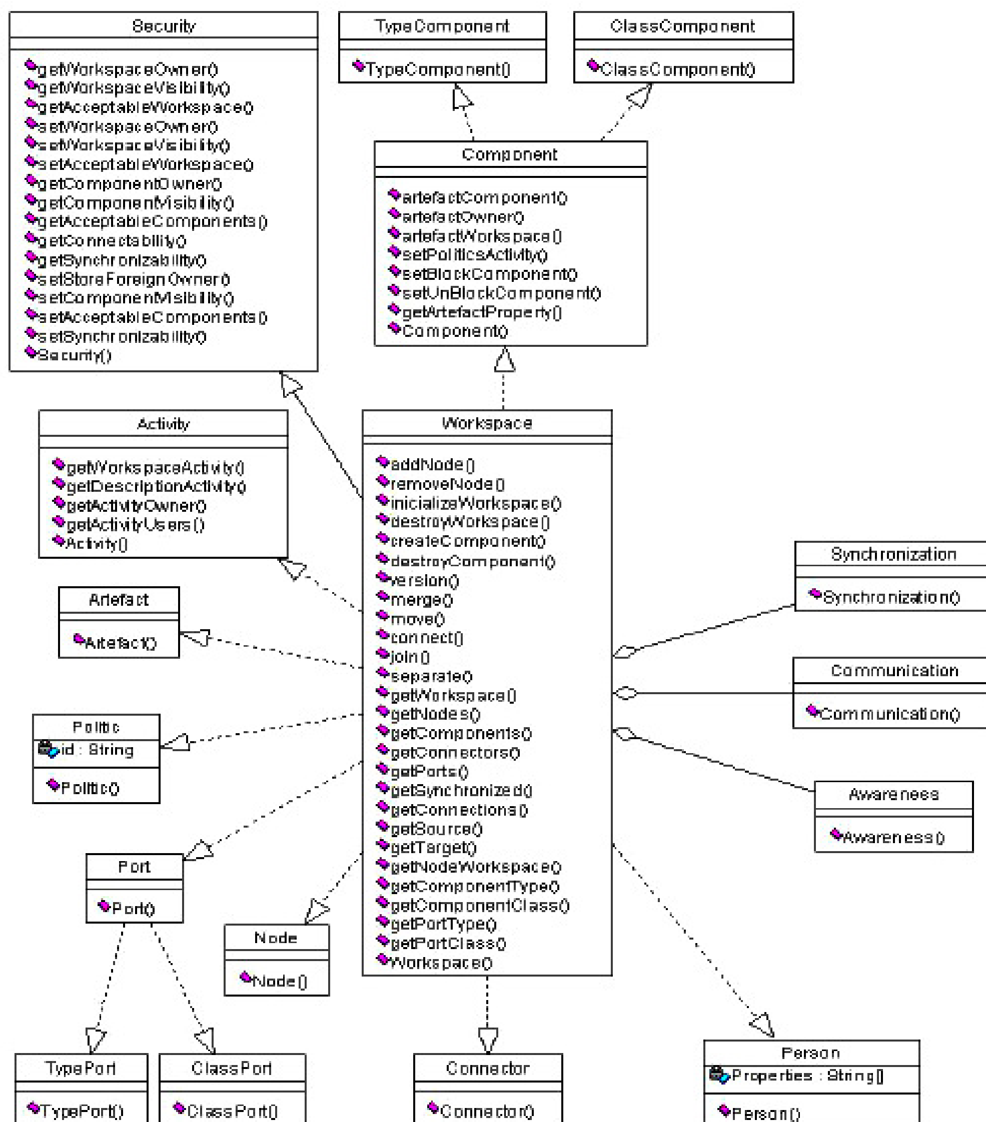

Figura 5.5: Diagrama de Classe do *framework* de cooperação (Pozza , 2005).

Em Pozza (2005) são apresentados alguns diagramas de seqüência que mostram que o Modelo SPC, no que tange à execução das atividades por grupos resultando em cooperação, atende as funcionalidades previstas no ambiente DiSEN.

6. ESTUDO DE CASO

6.1 Introdução

O estudo das estratégias de integração das tecnologias de serviços web e aspectos juntamente com o estudo do ambiente DiSEN corroboraram para o desenvolvimento de uma proposta de integração dessas duas tecnologias no modelo SPC.

6.2 Proposta

Como os objetivos da Programação Orientada a Aspectos (POA) e da tecnologia de Serviços Web são antagônicos a integração dessas duas tecnologias, torna-se uma tarefa bastante complexa.

A proposta dessa monografia centra-se na elaboração de um estudo de caso em que se possa aplicar a integração de aspectos e serviços Web. O cenário de aplicação do estudo de caso é o ambiente DiSEN, um ambiente que oferece suporte ao desenvolvimento distribuído de software, mais especificamente o modelo SPC, que propõe um padrão na cooperação entre usuários de workspaces compartilhados para o DiSEN.

Das metodologias apresentadas e analisadas no Capítulo 4 adotou-se a proposta de Ortiz et. *al* 2005 para aplicação no estudo de caso pois ela independe de tecnologia de implementação e/ou plataforma de execução. Além disso, ela apresenta (ou adota???) uma abordagem MDA, na qual são criados diferentes modelos com diferentes níveis de abstração. O uso da MDA provê alguns benefícios em relação à produtividade, portabilidade e interoperabilidade. O ponto forte dessa proposta relaciona-se com a definição do PIM (*Platform Independently Model*) onde são aplicadas as técnicas orientadas a aspectos e a análise de duas abordagens distintas para o modelo específico dependendo do nível em que a composição será executada, no projeto ou na implementação.

6.3 Modelagem

Um dos principais objetivos dos serviços *web* é o de prover a interoperabilidade entre diferentes sistemas, independente da plataforma e da linguagem de programação utilizadas por eles. O campo de aplicação dessa tecnologia é bem amplo. Ela pode ser

usada em qualquer aplicação que requeira interoperabilidade, compatibilidade com sistemas legados, lidar com domínios distintos e entre outros. Além disso, o modelo dos serviços web pode ser utilizado para promover o reuso dos componentes existentes nos sistemas integrados.

No contexto do modelo SPC, as três características (Sincronização, Percepção e Comunicação) e a *Workspace* foram definidas como serviços *web.* Das características enunciadas por Snell *et al*.(2001) e Cerami (2002) definiu-se que as características a serem implementadas como serviços web devem:

- estar disponíveis sobre a internet ou uma rede privada (intranet);
- utilizar o padrão XML para representação e troca de dados;
- ser independentes de sistema operacional e linguagem de programação;
- ser possível de descobrir através de mecanismos simplificados de busca.

A Figura 6.1 ilustra o modelo SPC na arquitetura dos serviços web.

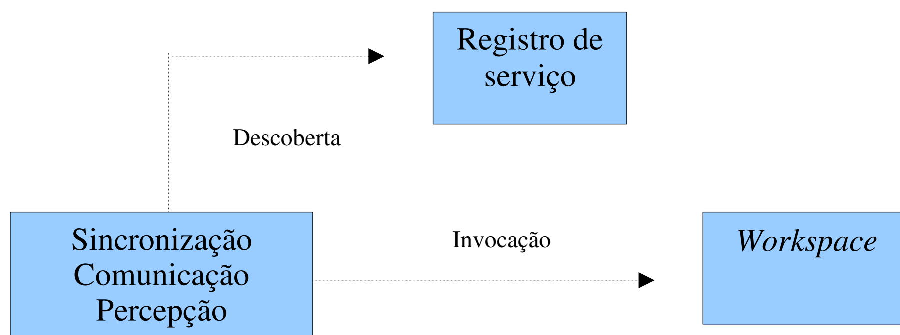

Figura 6.1: Representação do modelo SPC na arquitetura dos serviços web.

Com base na Figura 5.5 e nas operações, definidas em Pozza (2005), que oferecem suporte a Sincronização, Percepção e Comunicação elaborou-se o diagrama a seguir. As classes *Port, Artefact, Person, Connector* e *Politic* não foram representadas pois não são objetos de estudo deste trabalho.

A Figura 6.2 apresenta as classes propostas por Pozza(2005) para o modelo SPC com a distribuição dos métodos. A Figura 6.3 ilustra as classes com os possíveis aspectos identificados.

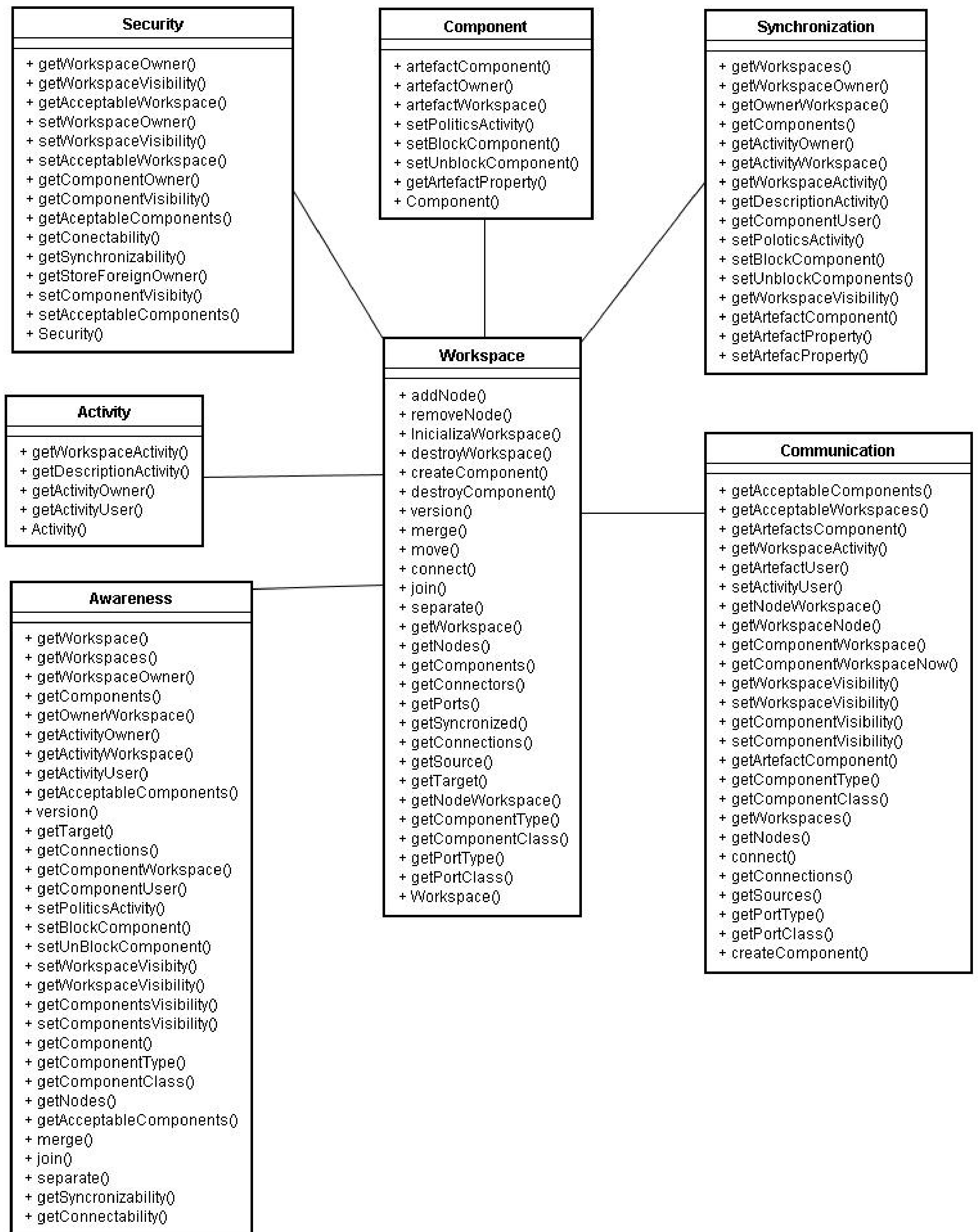

Figura 6.2: Diagrama com a distribuição dos métodos entre as classes. (Adaptado de Pozza, 2005).

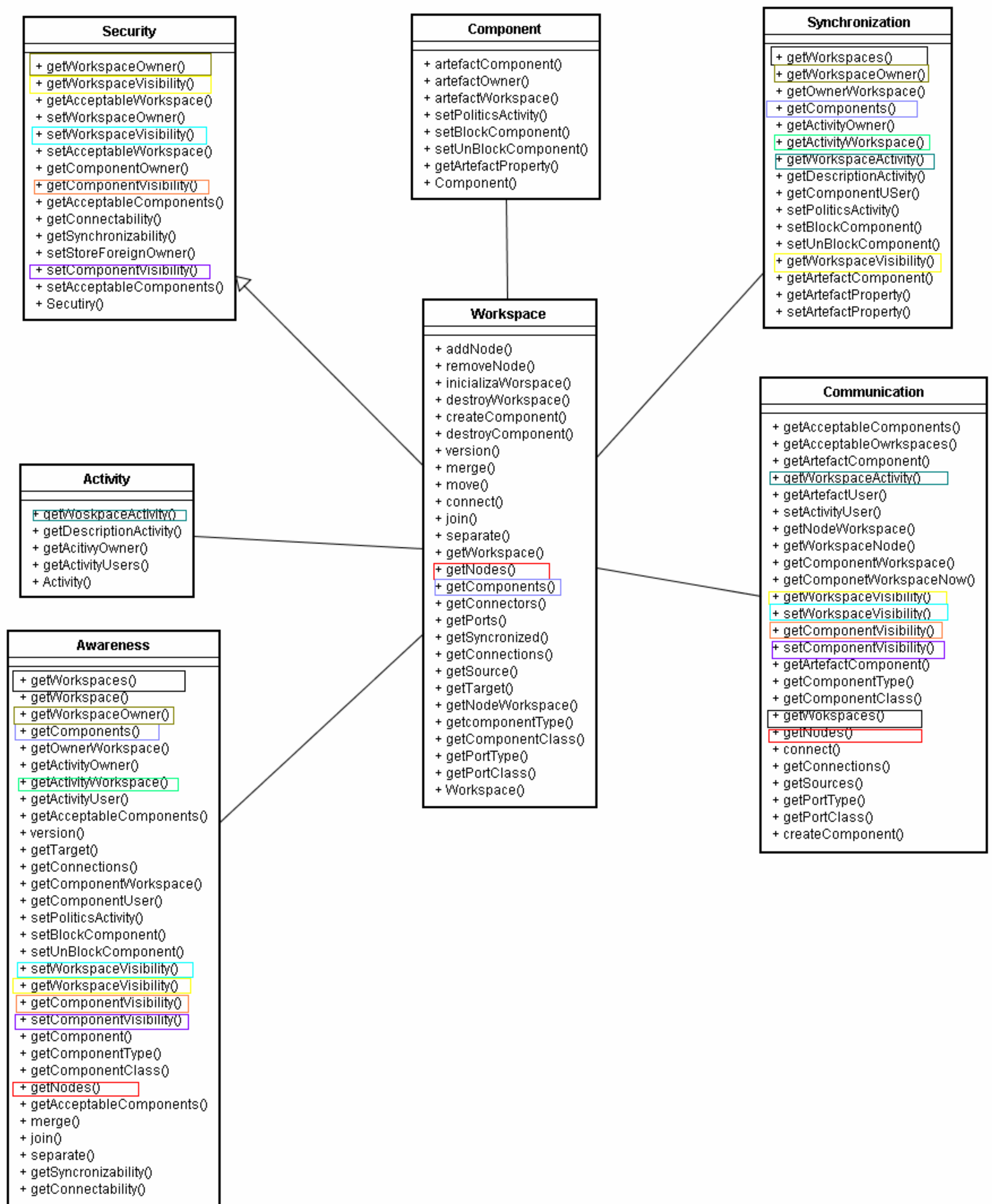

Figura 6.3: Diagrama com a identificação dos possíveis aspectos. (Adaptado de Pozza , 2005).

O Quadro 6.1 apresenta as propriedades que se encontram espalhadas no modelo SPC e em quais classes elas se encontram.

Quadro 6.1: Propriedades candidatas a aspectos.

Propriedades/ Classes	**Workspace**	**Security**	**Awareness**	**Synchronization**	**Activity**	**Communication**
getWorkspaces			**X**	**X**		
getWorkspaceVisibility		**X**	**X**	**X**		
setWorkspaceVisibility			**X**	**X**		
getComponentVisibility		**X**	**X**	**X**		
setComponentVisibility		**X**	**X**	**X**		
getComponents	**X**		**X**	**X**		
getWorkspaceOwner		**X**	**X**	**X**		
getActivityWorkspace			**X**	**X**		
getNodes	**X**		**X**			**X**
getWorkspaceActivity				**X**	**X**	

Um componente é definido como qualquer aplicação que consuma ou forneça um serviço *web*, ou seja, os serviços *web* são representados como componentes. Alguns detalhes foram omitidos a fim de simplificar a apresentação do modelo e os serviços web são representados como “caixas pretas”. Tendo como base as propriedades definidas no Quadro 6.1 e os prospectos do modelo SPC foram definidos os aspectos.

As propriedades, requisitos funcionais, foram agrupadas como aspectos, de acordo com suas funcionalidades, por se encontrarem espalhadas e entrecortando a aplicação em vários pontos. O prospecto de persistência e atualização, interesses não-funcionais, são exemplos clássicos de aspecto. A persistência independentemente da classe ou dos objetos a serem persistidos aparece espalhada pelo código. Da mesma forma que a persistência, a atualização aparece dispersa ao longo do código.

As Figuras 6.4, 6.5, 6.6 e 6.7 ilustram as classes do modelo SPC separadamente para melhor visualização da modelagem. O modelo completo do estudo de caso é apresentado no anexo A.

A Figura 6.4 apresenta o PIM utilizando aspectos e serviços web para a Workspace. A classe *workspace* é interceptada por dois aspectos, *NodesGroup_Aspect* e *ComponentGroup_Aspect* que contém *getNodes* e *getComponents* como *action.* A ligação com estereótipo *ApplyExecutionAspect* limita a *action*, especificando a qual método ela será aplicada. As notas associadas à *action* especificam em que ponto o método será interceptado, ou seja, o ponto em que o aspecto será chamado. A *action getNodes* será invocada durante a execução do método *getNodes()* e a *getComponents* durante a execução do método *getComponents()*.

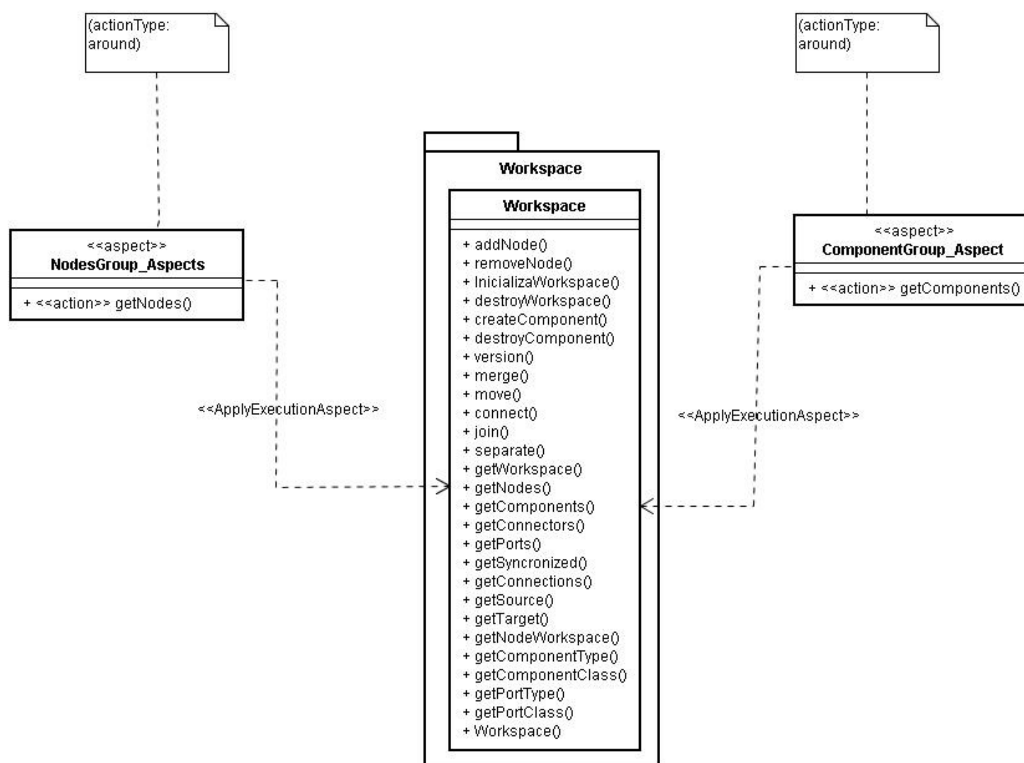

Figura 6.4: Workspace e seus aspectos.

Na Figura 6.5 é ilustrado o PIM para a característica Percepção que é representada pela classe *Awareness* e é entrecortada por cinco aspectos: *WorkspaceGroup_Aspect*, *Workspace_Aspect*, *NodesGroup_Aspect*, *Activity_Aspect* e *ComponentGroup_Aspect*.

Workspace_Aspect contém três *action*: *getWorkspaceVisibility, setWorkspaceVisibility* e *getWorkspaceOwner* os quais são relacionados com ligações de estereótipo *ApplyExecutionAspect*, que especifica a qual método a *action* será aplicada. As notas associadas a cada *action* indicam onde o método será interceptado. As notas

relacionadas indicam que a *action* será invocada durante a execução dos métodos relacionados.

A *WorkspacesGroup_Aspect*, estereotipada como *aspect* apresenta uma *action* que intercepta o método *getWorkspaces()* e é chamada durante a execução do método. A *ComponentGroup_Aspect* contém uma *action getComponents* que se relaciona com o método *getComponents*() por uma relação com estereótipo *ApplyExecutionAspect.*

Activity_Aspect possui duas *actions*: *getWorkspaceActivity* e *getActivityWorkspace,* as quais se relacionam com os métodos da classe por uma ligação com estereótipo de *ApplyExecutionAspect* que a relaciona com um método da classe.

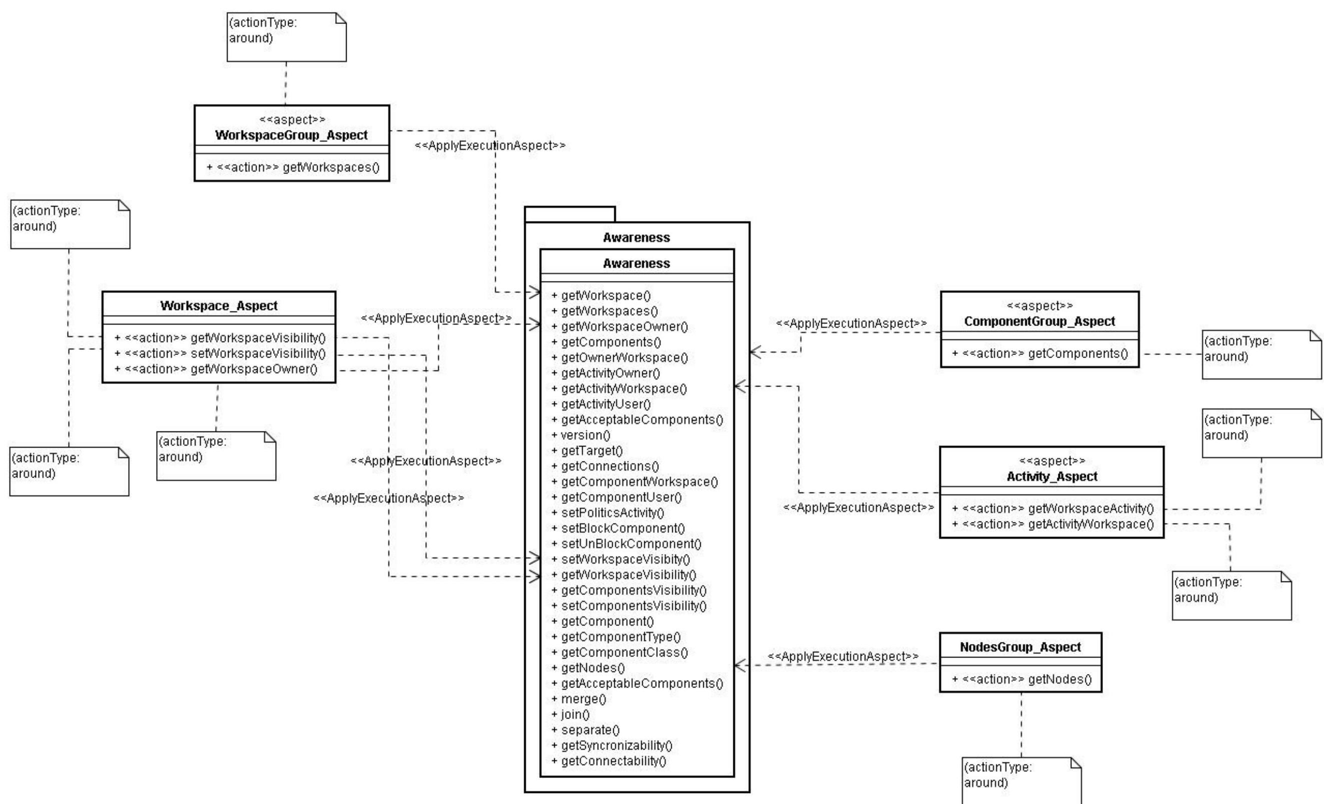

Figura 6.5: Percepção e seus aspectos.

O modelo PIM para a Sincronização é mostrado na Figura 6.6. A característica é representada pela classe *Synchronization,* a qual é interceptada por cinco aspectos: *ComponentGroup_Aspect*, *Persistence*, *Update*, *Workspace_Aspect* e *WorkspaceGroup_Aspect* . Os aspectos possuem *action* que são relacionadas aos métodos

das classes por meio de uma ligação *ApplyExecutionAspect* e apresentam notas que indicam em que momento eles serão invocados pelos métodos.

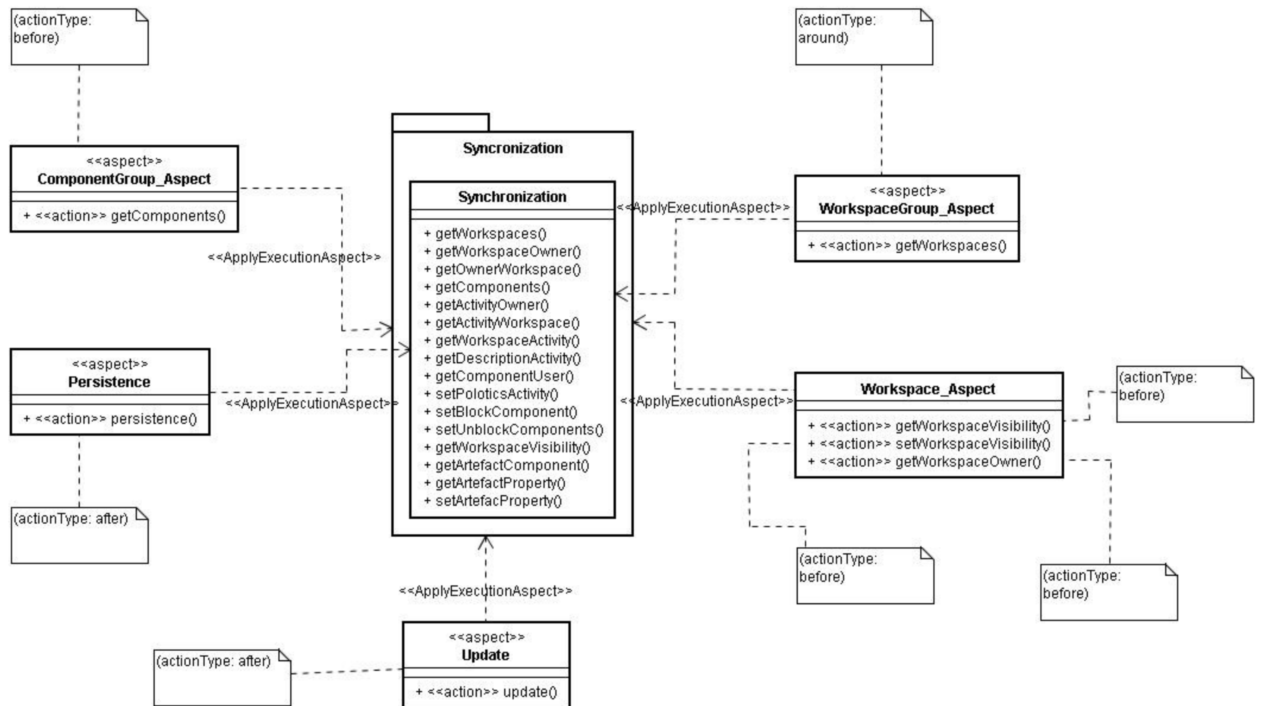

Figura 6.6: Sincronização e seus aspectos.

O modelo PIM para a Comunicação é ilustrado na Figura 6.7. A característica é representada pela classe *Communication,* a qual é interceptada por quatro aspectos: *Component_Aspect*, *Activity_Aspect*, *Workspace_Aspect* e Nodes*Group_Aspect* . Os aspectos possuem *action* que são relacionadas aos métodos das classes por meio de uma ligação *ApplyExecutionAspect* e apresentam notas que indicam em que momento eles serão invocados pelos métodos.

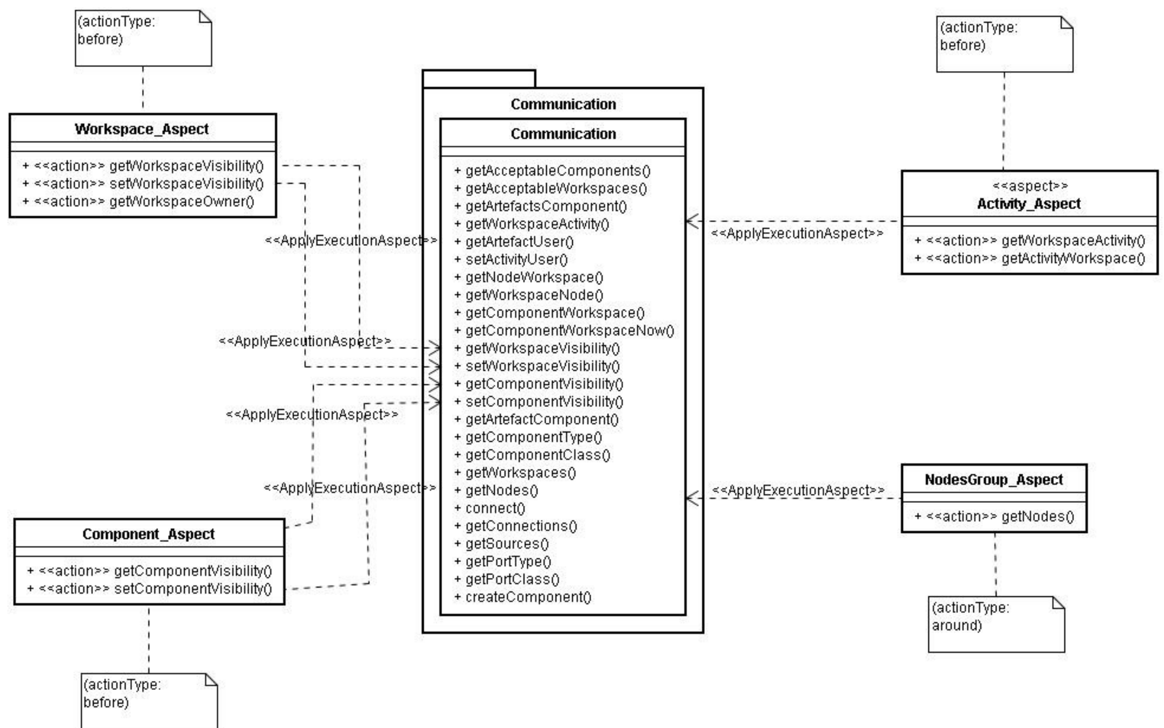

Figura 6.7: Comunicação e seus aspectos.

6.4 Considerações Finais

Na seção anterior foi definido o PIM (*Platform Independently Model*) para as características (Sincronização, Comunicação e Percepção) e a *Workspace*. A definição do PIM confere maior modularidade, reusabilidade, manutenibilidade e independência de plataforma e/ou tecnologia o que corrobora para redução de erros e melhor gerenciamento da complexidade dos componentes.

A metodologia adotada (Ortiz et. Al, 2005), confere alguns benefícios em relação à produtividade, portabilidade e interoperabilidade. Além disso, há uma facilidade para se utilizar esta abordagem, pois a mesma usa a notação UML, apenas acrescentando alguns estereótipos.

7. CONCLUSÃO

O cenário de desenvolvimento de software tem evoluído nos últimos anos em relação a novas metodologias, processos e tecnologias, o que propicia um ambiente para o surgimento de novos paradigmas de desenvolvimento. Dentre esses paradigmas que tem surgido, dois, em particular, têm sido cada vez mais adotados pelos desenvolvedores de software: o paradigma da P*rogramação Orientada a Aspectos* (POA) (KICZALES *et al.*, 1997), e o paradigma da C*omputação Orientada a Serviços* (COS) (PAPAZOGLOU e GEORGAKOPOULOS, 2003).

A POA propõe novos mecanismos e abstrações para facilitar a separação de interesses de software que seriam muito difíceis de modularizar utilizando apenas os recursos dos paradigmas tradicionais. A COS propõe um novo modelo de desenvolvimento, onde componentes de software são implementados e integrados de uma forma fracamente acoplada à linguagem de programação, plataforma de execução ou protocolo de transporte, utilizados por estes componentes.

O principal objetivo deste trabalho foi o de buscar metodologias que integram as duas tecnologias, analisá-las, destacando suas vantagens e desvantagens e, posteriormente, adotar a mais adequada para a elaboração do estudo de caso no ambiente Disen.

A proposta de Ortiz et. al(2005), adotada para a integração das duas tecnologias, apresenta independência em relação a tecnologia de implementação e/ou plataforma de execução e apresenta uma abordagem MDA em que são criados diferentes modelos com diferentes níveis de abstração. A aplicação da MDA confere alguns benefícios em relação à produtividade, portabilidade e interoperabilidade.

Seguindo essa abordagem definiu-se o modelo independente de plataforma (PIM) para o modelo SPC, conferindo assim maior modularidade, reusabilidade e manutenibilidade o que por sua vez proporciona um melhor gerenciamento da complexidade dos componentes e redução dos erros.

O trabalho contribui para uma melhoria do modelo SPC em relação à modularidade, reusabilidade e manutenibilidade. A composição das características do modelo como serviço web viabiliza a integração com outras aplicações independentemente dos ambientes de desenvolvimento e plataforma de execução. Já o uso de aspectos permite que o código seja mais fácil de escrever, compreender, reutilizar e manter uma vez que as propriedades entrecortantes ficarão encapsuladas, proporcionando uma redução de complexidade dos serviços web e facilitando as alterações.

7.2 Trabalhos Futuros

No decorrer do desenvolvimento desta monografia pode-se vislumbrar algumas possibilidades de trabalhos futuros, tais como:

- Dar continuidade ao trabalho especificando o modelo específico para uma plataforma (PSM) através das duas abordagens apresentadas por Ortiz *et. al* (2005);
- Implementação do estudo de caso utilizando a linguagem Java e AspectJ para o modelo de aspectos;
- Realizar mais estudos de casos para investigar sistematicamente as vantagens e limitações da abordagem utilizada;
- Avaliar outras abordagens para integração das duas tecnologias;
- Analisar a arquitetura do ambiente Disen como um todo e realizar a identificação de aspectos e serviços web, bem como sua integração;
- Investigar métricas que permitam avaliar quantitativamente os ganhos de modularização obtidos.

REFERÊNCIAS

ARSANJANI, A.; HAILPERN, B.; MATIRN, J.; TARR, P. *Web Services: Promises and Compromises*, *ACM Queue*, New York, NY, USA, v.1, n.1, p. 48-58, Mar. 2003.

BOOCH, G.; JACOBSON, I.; RUMABUGH, J.*UML: Guia do Usuário*. Campus, 2000.

CAHILL, V.; DEMPSEY, J. Aspects of System Support for Distributed Computing. In Aspect-Oriented Programming Workshop at ECOOP'97, Finlândia: Springer-Verlag LNCS, 1997. *Proceedings* ECOOP.

CERAMI, E. Web Services Essentials. O'Reilly, 2002.

CHAVEZ, C. V. F. G.; LUCENA, C. J. P. *A Theory of Aspects for Aspect-Oriented Software Development*. Em: Simpósio Brasileiro de Engenharia de Software (SBES), 2003, Manaus. 17º SBES.

COULOURIS, G.; DOLLIMORE, J.; KINDBERG, T. *Distributed System: Concepts and Design*. Pearson Education Endinburgh Gate Ltd, 2001.

ELRAD, T. AKSIT, M.; KICKZALES, G.; LIEBERHERR, K.; OSSHER, H. *Discussing Aspects of AOP*. Em: Anais do ACM, 2001.

GIMENES, I. M. S. ; HUZITA, E. H. M. *Desenvolvimento Baseado em Componentes – Conceitos e Técnicas*. Ciência Moderna, 2005.

GRADECK, J.; LESIECKI, N. *Mastering AspectJ: Aspect-Oriented Programming in Java*. Wiley, 2003.

HARRISON, W.; OSSHER, H. *Subject-Oriented Programming (a critique of pure objects).* Proceedings of the Conference on Object-Oriented Programming: Systems, Languages and Applications. 411-428, Washington, ACM, 1993.

KICZALES, G.; LAMPING, J.; MENDHEKAR, A.; MAEDA, C.; LOPES, C. V.; LOINGTIER, J.; IRWIN, J. *Aspect-Oriented Programming*. Em: European Conference on Object-Oriented Programming (ECOOP), Finland, 1997.

KREGER, D. C.; CAHILL, V.; *Web Services Conceptual Architectura (WSCA 1.0)* Disponível por WWW em : http://www-3.ibm.com/software/solutions/webservices/documentation.html. Acessado em: 10/2006.

LOPES, C. I. V. *D: a Language Framework For Distributed Programming*. Ph. D. Thesis, Northeast University, 1997.

NEWCOMER, E. *Understanding Web Services- XML, WSDL SOAP and UDDI*. Pearson Education, 2002.

ORTIZ, G.; CLEMENTE, P. J.; HERNANDEZ, J.; AMAYA, P. A.; *How to Model Aspect-Oriented Web Services*. In the Workshop on Model-drive Web Engineering (MDWE 2005), University of Technology, Sidnei, Australia. Disponível http://www.lcc.uma.es/~av/mdwe2005/agenda.html. Acesso em outubro/2006.

OSSHER, H.; TARR, P. Operation-Level Composition: A Case in (Join) Point. In Aspect-Oriented Programming Workshop at ICSE'98. Kyoto (Japão), 1998. *Proceedings…*ICSE.

LADDAD, R. *AspectJ in Action – Practial Aspect-Oriented Programming*. Manning, 2003.

MIAN, P.G. ; NATALI, A . C. C; FALBO, R. A. *Ambientes de Desenvolvimento de Software e o Projeto ADS*. Revista Engenharia, Ciência e Tecnologia, Volume 04, Número 04, ISSN 1414-8692, pp 3 - 10, Vitória, Espírito Santo, Brasil, Julho/Agosto 2001.

PASCUTTI, M. C. D. *Uma Proposta de Arquitetura de um Ambiente de Desenvolvimento de Software Distribuído Baseada em Agentes*. Dissertação (Mestrado) – Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação em Computação, Porto Alegre, p.29-52, 2002.

PIVETA, E. K. *Um Modelo de Suporte a Programação Orientada a Aspectos*. Dissertação (Mestrado em Ciência da Computação). Centro Tecnológico. Florianópolis-SC. Universidade Federal de Santa Catarina, p.25-44, 2001.

POZZA, R. S. *Proposta de um Modelo para Cooperação baseado no Gerenciador de Workspace no Ambiente DiSEN.* Dissertação (Mestrado em Ciência da Computação). Departamento de Informática. Maringá-PR. Universidade Estadual de Maringá, p. 50-83, 2005.

SOARES, S.; BORBA, P. *Desenvolvimento de Software Orientado a Aspectos Utilizando RUP e AspectJ*. Tutorial In: Simpósio Brasileiro de Engenharia de Software (SBES), 2004, Brasília.

SILVA, C. F.; *Uma Linguagem de Especificação de Aspectos para o Desenvolvimento Orientado a Serviços.* Dissertação (Mestrado em Informática). Universidade de Fortaleza, p.40-53, 2006.

SOUZA, G. M. C. *Uma Abordagem Direcionada a Casos de Uso para o Desenvolvimento de Software Orientado a Aspectos.* Dissertação (Mestrado em Ciência da Computação). Centro de Informática. Recife-PE. Universidade Federal de Pernambuco, p. 30-43, 2004.

SNELL, J. ; TIDWELL, D.; KULCHENCKO, P. *Programming Web Services with SOAP*. O'Really, p. 21-69, 2001.

WIESE, I. S.; STEINMACHER, I.; POZZA, R.; HUZITA, E. H. M.; AMORIN, E.F.; PASCUTTI, M. C. D. Uma Proposta de Arquitetura para Ambientes de Desenvolvimento Distribuído de Software. Workshop de Ingenieria de Software y Bases de Dados. In CACIC 2005. *Proceedings*...(em cd). 2005

W3C. *eXtensible Markup Language*. Disponível por WWW em: http://www.w3c.org/xml. Acessado em: 07/2006.

ANEXO A – PIM do Modelo SPC

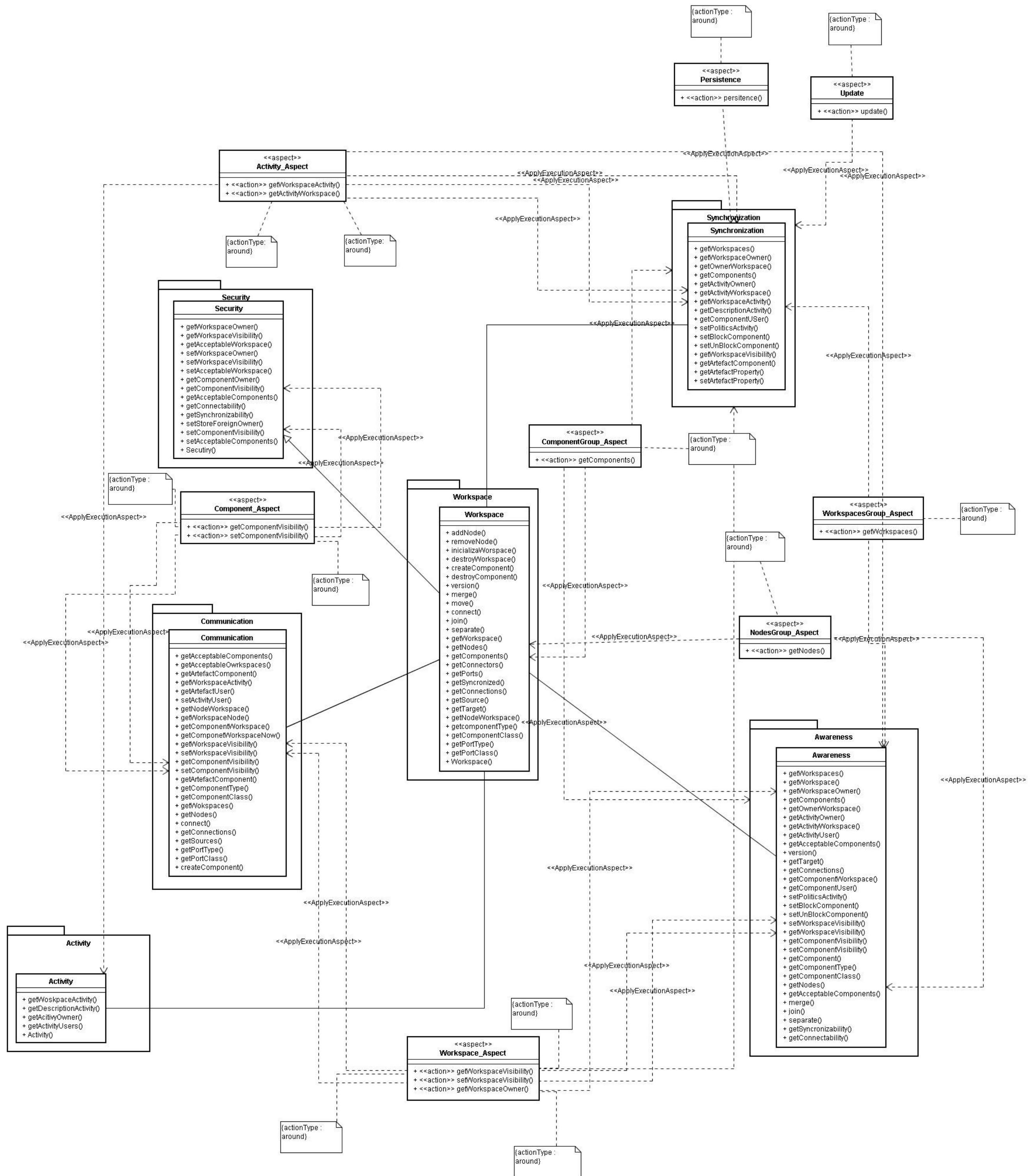

Figura A1: PIM do Modelo SPC utilizando serviços web e aspectos.