UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
CURSO DE CIÊNCIA DA COMPUTAÇÃO

EDUARDO CASOTTI POSTAL

Módulo de Monitoramento de Serviços Web em CWSMarts Utilizando Aspectos

Trabalho de Graduação.

Prof. Dr. Leandro Krug Wives
Orientador

Porto Alegre, Julho de 2011

SUMÁRIO

LISTA DE ABREVIATURAS E SIGLAS

BPEL	Business Process Execution Language
CWS	Composite Web Service
CWSMart	Composite Web Services Mart
HTTP	Hypertext Transfer Protocol
OASIS	Organization for the Advancement of Structured Information Standards
QoS	Quality of Service
SLA	Service Level Agreement
SOA	Service Oriented Architecture
SOAP	Simple Object Access Protocol
UDDI	Universal Description Discovery and Integration
URI	Uniform Resource Identifier
SW	Serviço Web
WSCDL	Web Service Choreography Description Language
WSDL	Web Service Description Language
W3C	World Wide Web Consortium
XLANG	Extensible Language
XML	Extensible Markup Language
POA	Programação Orientada a Aspectos

LISTA DE FIGURAS

LISTA DE TABELAS

RESUMO

Serviços Web compostos são obtidos pela combinação de vários serviços Web que objetivam a realização de uma tarefa comum. Eles são projetados para atender às requisições de usuários a partir do momento em que nenhum serviço Web é capaz de fazê-lo individualmente.

Um *CWSMart* é uma estrutura desenvolvida para hospedar serviços Web compostos de acordo com a semelhança de suas funcionalidades. Essa estrutura é dividida em módulos, sendo que cada qual desempenha uma tarefa particular. Este trabalho aborda o desenvolvimento do módulo de monitoramento presente em um *CWSMart* objetivando a coleta de informações relativas a execução dos serviços Web hospedados em sua estrutura.

Uma das questões fundamentais consiste em monitorar os serviços de maneira transparente, sem que o programador tenha que chamar o monitor explicitamente. Para tanto, este trabalho utiliza Aspectos como forma de definir mecanismos que injetam automaticamente chamadas ao serviço monitor, sem que o programador explicitamente as realize. Assim, o código de monitoramento fica separado, isolado do código do serviço em si.

Palavras-Chave: serviços Web, serviços Web compostos, *CWSMarts*, monitores.

Module to monitor Web services stored on CWSMarts using aspects

ABSTRACT

Composite Web services are developed through the combination of many Web services performing different functions aiming a final common task. They are designed to attend user requests that no single Web service is capable to do.

A CWSMart is a structure created to host composite Web services based on the similarity of its functions. This structure is composed by different modules and each of them plays a particular task. This work discusses the development of a CWSMart monitoring module and the entire process of collect execution-related information from Web services hosted in the CWSMart structure.

One of the main issues consists on monitoring services in a transparent way, i.e., the programmer should not explicitly call the monitor to log services activities. Thus, this work proposes the use of Aspects to define mechanisms that are able to automatically inject calls to the monitor. This way, the monitoring code is kept isolated from the service code.

Keywords: Web services, composite Web services, CWSMarts, monitors.

1 INTRODUÇÃO

Um serviço Web é caracterizado por ser uma aplicação cujo objetivo é implementar uma determinada funcionalidade capaz de ser invocada por usuários e por outras aplicações através da submissão de mensagens apropriadas. Esses serviços merecem notoriedade por serem capazes de prover um modelo simples de programação e implantação de aplicações através da web.

Desse modo, ao explorarmos a área da computação reservada aos serviços Web, é comum encontrarmos composições desses componentes destinadas a cumprir objetivos maiores e mais amplos. Porém, compor serviços Web para que trabalhem em conjunto na realização de uma tarefa maior é algo complexo.

Essas práticas de composição apresentam uma grande quantidade de desafios. Começando pela escolha dos serviços Web que irão fazer parte dessa composição até a garantia de que sua execução será confiável e segura, entre muitos outros. O que acaba tornando a formação de serviços Web Compostos um assunto merecedor de um estudo profundo.

A partir dessas premissas, Maamar et al. (2009) sugerem a proposta de desenvolver centros especializados onde os serviços Web já se encontrariam combinados e agrupados formando serviços Web Compostos prontos para serem utilizados. Esses centros são chamados de *Composite Web Services Marts* (*CWSMarts*).

A função dos *CWSMarts* reside em hospedar serviços Web compostos que possuam as mesmas funcionalidades, independendo de como são especificados, da forma como operam, etc. Assim, a característica básica dos *CWSMarts* é formar grupos de serviços Web baseados, primordialmente, em suas funções, desconsiderando características de menor importância.

O módulo monitor é uma dentre as várias partes que agregam a estrutura citada e é responsável por monitorar serviços Web compostos durante o seu funcionamento. Portanto ele será o responsável por capturar as informações em tempo de execução dos sistemas monitorados. Essa ação tem como finalidade obter diversos dados relacionados à execução dos serviços Web e então, a partir desses parâmetros, atualizar o repositório de descrições, contido no *CWSMart*.

Com essas informações adicionais no repositório, é possível inferir novos dados a respeito da execução dos serviços Web, já que detalhes a respeito de seu desempenho só são conhecidos após a análise de sua execução.

Inicialmente, a proposta central deste trabalho estava focada somente no projeto e desenvolvimento do monitor citado. Definindo informações monitoradas relevantes e registrando eventos pertinentes para a monitoração. Porém, conforme o trabalho foi avançando, dificuldades foram encontradas para fazer com que os serviços informassem seu status ao monitor sem que seu código fosse explicitamente modificado (i.e., desejava-se um mecanismo capaz de monitorar serviços sem que seu código fosse alterado ou sem que o programador tivesse que fazer tal registro explicitamente). Com

isso, o trabalho sofreu algumas mudanças de planejamento. As barreiras surgiram principalmente no momento da implementação desse monitor e no momento de coletar tais informações automaticamente. Devido a esses fatores, essa etapa acabou consumindo um tempo maior que o esperado, mudando os rumos do trabalho. A partir desse ponto, o foco dado ao monitor, com relação ao conteúdo monitorado, acabou se tornando secundário e se passou a procurar uma forma que minimizasse os efeitos da monitoração nos sistemas monitorados. E a pergunta inicial “o que monitorar?” acabou se tornando “como monitorar?”.

Com esse novo foco o trabalho busca um meio de fazer com que a monitoração seja realizada de modo transparente aos serviços monitorados. Isso é alcançado através do uso de aspectos. Porém esse método introduz algumas limitações que serão abordadas nas seções referentes aos detalhes da implementação.

Quanto à estrutura do texto, ela será organizada da seguinte forma: a introdução objetiva situar o leitor no contexto geral em que está inserido o trabalho, dando uma ideia sobre os assuntos que serão tratados. O capítulo 2 apresenta os conceitos relacionados ao desenvolvimento do projeto, dando uma descrição sucinta e alguns exemplos a respeito de cada termo utilizado para o desenvolvimento do trabalho. O capítulo 3 apresenta trabalhos relacionados à construção de sistemas de monitoramento, serviços Web e serviços Web compostos. O capítulo 4 constitui o centro do trabalho. Ele tem o objetivo de apresentar o projeto realizado e dar os detalhes referentes à implementação do monitor. Para finalizar, o capítulo 5 abrange as conclusões obtidas com este trabalho, abordando os resultados alcançados, as limitações do projeto e as perspectivas para trabalhos futuros.

2 CONCEITOS RELACIONADOS

Os conceitos apresentados a seguir constituem o escopo necessário para o entendimento do presente trabalho. São abordados detalhes referentes à Arquitetura Orientada a Serviços e a serviços Web. Ambos os assuntos primordiais para o entendimento e implantação da estrutura proposta.

A estrutura de um serviço Web é a solução mais aceita capaz de atender aos requisitos da Arquitetura Orientada a Serviços. Sua composição geralmente ocorre utilizando a linguagem BPEL. Nas seguintes seções os assuntos mencionados são apresentados e detalhados de acordo com sua importância ao longo do desenvolvimento do trabalho.

2.1 Arquitetura Orientada a Serviços

A Arquitetura Orientada a Serviços, do inglês *Service Oriented Architecture* (SOA), é definida como um estilo arquitetural capaz de suportar a orientação a serviços (SOA THE OPEN GROUP, 2005). Preconizando assim que as funcionalidades realizadas pelas aplicações devam ser disponibilizadas sob a forma de serviços (LUBLINSKY, 2007). Dessa maneira, a SOA coloca a prestação de serviços como o seu eixo central, dando destaque à sua gestão e ao cliente. Logo, nada mais é do que um modo de pensar em termos de desenvolvimento baseado em serviços e em seus resultados.

A arquitetura SOA é baseada nos princípios da computação distribuída e utiliza o paradigma *request/reply* para estabelecer a comunicação entre os sistemas clientes e os sistemas responsáveis pela implementação das funcionalidades (RAGHU, 2005). A palavra chave para essa arquitetura é flexibilidade, já que através da sua natureza fracamente acoplável, possibilita que os serviços tenham a sua interface completamente independente de sua implementação. Como consequência, desenvolvedores e integradores de sistemas são capazes de construir aplicações por composição de serviços sem saber o modo como eles foram implementados.

Serviços desenvolvidos sob a ótica da SOA possuem certas características fundamentais para o seu funcionamento. Entre elas podemos citar o fato de possuírem interfaces auto descritivas e independentes de plataforma. Eles se comunicam através de mensagens formalmente definidas, tendo em vista que a comunicação entre consumidores e provedores se dá em ambientes heterogêneos com pouquíssima informação a respeito dos provedores. Em sua publicação, os serviços são mantidos por registros que atuam com base na listagem de diretórios, além de cada serviço SOA possuir uma qualidade de serviço associada a sua funcionalidade.

2.1.1 Serviço

O Serviço é a abstração básica introduzida pela Arquitetura Orientada a Serviços. É uma função independente capaz de aceitar requisições e retornar resultados através de uma interface padronizada e bem definida. Como um de seus princípios não deve depender do estado de outras funções ou processos, exceto nos casos de serviços compostos. Além disso, a tecnologia utilizada na sua implantação não deve fazer parte da definição. Conforme definido por SOA The Open Group (2005), um serviço pode ser caracterizado e compreendido como:

- Uma representação lógica de uma atividade de negócio repetitiva;
- Uma entidade autocontida;
- Uma composição de um ou vários serviços;
- Uma "caixa preta" para os seus consumidores.

2.1.2 Qualidade de Serviço

Através da existência de sistemas responsáveis pela realização de missões críticas, torna-se necessário o preenchimento de alguns pré-requisitos por parte da Arquitetura Orientada a Serviços. Esses pré-requisitos estão relacionados com a forma com que o serviço será prestado e podem ser compreendidos como segurança, confiabilidade, entre outros. São denominados Qualidade de Serviço, do inglês *Quality of Service* (QoS).

Várias especificações relacionadas à Qualidade de Serviços estão se transformando em padrões desenvolvidos pelo *World Wide Web Consortium* (W3C) e pela *Organization for the Advancement of Structured Information Standards* (OASIS). A seguir, alguns desses padrões mencionados são apresentados.

2.1.2.1 Confiabilidade

A partir do momento em que diversas mensagens e documentos são trocados pelos consumidores e provedores de serviços, um modelo que garanta a confiabilidade da entrega dessas mensagens faz-se necessário. Esse ambiente relatado, com várias trocas de mensagens, é característico da arquitetura SOA. Desse modo, segundo Raghu (2005) a confiabilidade na entrega de mensagens é cumprida ao preencher certas características, tais como a eliminação de mensagens duplicadas, a garantia de que a mensagem será entregue ao destinatário, entrega para um ou vários usuários, entre outras.

2.1.2.2 Gerenciamento

Pode ser entendido como uma infraestrutura que possibilita aos administradores gerenciar os serviços que estão executando em ambientes heterogêneos. No momento em que o número de serviços expostos aumenta cada vez mais, essa estrutura adquire grande importância. (RAGHU, 2005)

2.1.2.3 Orquestração

Como já mencionado, serviços podem ser usados para integrar dados, aplicações e componentes. Consequentemente, conforme Erl (2008), essa integração requer que sejam padronizados fatores como comunicação assíncrona, processamento paralelo e transformação de dados. Obtendo, desse modo, a orquestração como um *QoS*.

2.1.2.4 Segurança

De acordo com Raghu (2005), a especificação de segurança está focada na troca de credenciais, integridade das mensagens e confidencialidade das mensagens trocadas. É de fácil entendimento a necessidade desses mecanismos para a utilização de certos serviços, já que muitos necessitam lidar com cuidado as informações transmitidas. Podendo ser exemplificadas desde a troca de senhas a informações sigilosas que não devem ser lidas por ninguém a não ser o receptor.

2.2 Serviço Web

Um serviço Web é caracterizado por ser uma aplicação cujo objetivo é implementar uma determinada funcionalidade capaz de ser invocada por usuários e por outras aplicações através da submissão de mensagens apropriadas. Esses serviços merecem notoriedade por serem capazes de prover um modelo simples de programação e implantação de aplicações através da Web.

Segundo a definição do W3C (http://www.w3.org/TR/ws-arch), um serviço Web é um software identificado por uma URI, cujas *interfaces e bindings* são capazes de ser definidas, descritas e descobertas por artefatos XML, além de suportar interações diretas com outros softwares que fazem uso de mensagens XML trocadas via aplicações Web.

Levando em consideração a definição adquirida através da *Webopedia* (http://www.webopedia.com/TERM/W/Web_Services.html), um serviço Web é definido como uma maneira padronizada de integrar aplicações Web utilizando os padrões abertos XML, SOAP, WSDL e UDDI sobre um protocolo de Internet. O XML é usado para adicionar *tags* aos dados, a tecnologia SOAP tem a função de transferir dados, WSDL é utilizado para descrever os serviços disponíveis e UDDI é usado para listar quais serviços estão disponíveis.

Serviços Web são componentes modulares, fracamente acoplados e auto descritivos capazes de prover modelos simples de programação e implantação através da Web. Além disso, serviços Web estão associados com uma funcionalidade que representa o tipo de serviço oferecido. Dentre as funcionalidades existentes é possível citar: previsão do tempo, conversão de moeda, aluguel de carro, aluguel de hotéis, entre outros.

De forma mais abrangente, os serviços Web são vistos como um modelo de implementação de Arquitetura Orientada a Serviços (SOA). Essa arquitetura define uma série de padrões e políticas cujos princípios estão embasados na disponibilidade de funcionalidades sob a forma de serviços. Os serviços Web são um exemplo de tecnologia que está contido na Arquitetura Orientada a Serviços, e podem ser considerados como o método mais utilizado atualmente para a implantação da SOA.

2.2.1 Operações Fundamentais

Com a finalidade de ilustrar o modo como opera um WS, são detalhadas cinco operações fundamentais para o seu funcionamento: descrição, publicação, descoberta, invocação e composição. Tais operações são definidas sob o paradigma da Arquitetura Orientada a Serviços e estão relacionadas diretamente com a forma sob a qual o serviço Web irá operar e interagir com os usuários e consumidores.

2.2.1.1 Descrição

Segundo Maamar et al. (2009), a descrição de serviços Web especifica tanto propriedades funcionais quanto não-funcionais desses sistemas. Um de seus objetivos é fornecer acesso ao WS, a outros serviços, de forma automática. Para a elaboração de uma descrição, o *World Wide Web Consortium* (W3C) propõe o uso da linguagem *Web Service Description Language* (WSDL), baseada em XML e capaz de preencher os requisitos apresentados cima.

2.2.1.2 Publicação

A publicação, de acordo com Erl (2005), se refere à operação em que os provedores de serviços Web anunciam seus serviços à comunidade. Essa publicação é realizada através da utilização de diretórios e registros existentes na Web. Dentre os protocolos mais utilizados podemos citar o *Universal Description Discovery and Integration* (UDDI).

2.2.1.3 Descoberta

Tem como objetivo central encontrar, através de requisições provindas dos consumidores, o serviço Web que melhor atenda às necessidades requeridas por esses usuários, de acordo com Maamar et al. (2009). Essa descoberta só é possível devido às descrições publicadas pelos provedores de serviços em registros tais como o UDDI.

2.2.1.4 Invocação

Ocorre no momento em que outros serviços chamam uma função referente a um serviço Web (MAAMAR et al., 2009). Essa operação utiliza o protocolo *Simple Object Access Protocol* (SOAP) visto que cada serviço Web pode operar utilizando uma linguagem diferente. Sendo assim, o SOAP funciona como um contrato que objetiva padronizar a comunicação do serviço Web com os outros sistemas presentes.

2.2.1.5 Composição

Forma encontrada para agrupar serviços Web objetivando a realização de uma tarefa maior e mais complexa. Ela é utilizada quando apenas um único serviço Web não consegue realizar uma função proposta pelo usuário, necessitando assim a colaboração de outros serviços Web para ajudar na sua realização. Desse modo, a composição é obtida através da combinação de serviços Web disponíveis que se enquadrem nas funções desejadas. Entre os métodos utilizados para a composição de serviços Web é importante citar as linguagens *Business Process Execution Language* (BPEL) e *Web Services Choreography Description Language* (WSCDL).

2.2.2 Tecnologia

O objetivo da tecnologia de serviço Web é possibilitar que aplicações trabalhem em conjunto sobre protocolos padrões de Internet sem a intervenção humana. A seguir estão listadas as principais tecnologias utilizadas para a implantação e utilização de um serviço Web.

2.2.2.1 Extensible Markup Language

A linguagem XML é considerada a base do funcionamento de um serviço Web. Sua sintaxe especifica como os dados são representados e com que qualidade de serviço esses dados são transmitidos, além de detalhar a publicação e descoberta de serviços. O

XML, de acordo com W3C (http://www.w3.org/XML), fornece a descrição, o armazenamento e o formato da transmissão para a troca de dados.

2.2.2.2 Simple Object Access Protocol

A tecnologia SOAP é desenvolvida com base em XML e HTTP e utilizada na comunicação realizada entre o consumidor e o provedor de serviços. Como não impõe qualquer semântica sobre o modelo de programação ou sobre uma implementação específica, permite que o serviço e o cliente sejam desenvolvidos utilizando linguagens distintas.

2.2.2.3 Web Services Description Language

O WSDL é uma linguagem cuja especificação foi desenvolvida pelo W3C (http://www.w3.org/TR/wsdl) e tem como função principal descrever serviços Web sob o formato da linguagem XML. Especifica tanto propriedades funcionais quanto não funcionais desses sistemas

2.2.2.4 Universal Description Discovery and Integration

Desenvolvido para registro e organização de serviços Web, o UDDI é um padrão aprovado pela OASIS (http://uddi.org/pubs/uddi_v3.htm) e define um método para publicar e descobrir diretórios de serviços em uma Arquitetura Orientada a Serviços. Um sistema UDDI é um serviço Web com funções próprias, tais como a gerência de informações sobre provedores, implementações e metadados de serviços. Os usuários fazem uso dessa ferramenta para descobrirem os serviços que lhes interessam e obter, a partir daí, informações necessárias para a comunicação com o serviço escolhido.

2.3 Serviço Web Composto

A composição de serviços Web é uma técnica utilizada no momento em que as necessidades dos usuários não podem ser atendidas utilizando um único serviço Web. Dessa maneira, um serviço Web composto é obtido pela combinação de vários serviços Web disponíveis objetivando a realização de uma tarefa comum. Muitas linguagens usadas na composição de serviços Web são encontradas na literatura, assim como WS-BPEL (http://www.ibm.com/developerworks/library/specification/ws-bpel), WSCDL (http://www.ebpml.org/ws_-_cdl.htm) e XLANG (www.ebpml.org/xlang.htm).

Ao contrário da composição tradicional de elementos, a composição de serviços Web não implica a integração física ou administrativa dos componentes. A fronteira de um serviço Web composto é apenas lógica, tendo em vista que os seus componentes podem ter localizações geográficas distintas e estar sob diferentes domínios administrativos.

A infraestrutura de composição tem a função de um *middleware*, oferecendo um *framework* que facilite a definição, criação e execução de serviços Web complexos, focando assim a sua preocupação na lógica de negócio, e não nos detalhes da implementação.

A composição de serviços Web mantém interna a implementação das operações dos serviços. A especificação de um serviço composto é propriedade de uma empresa e é mantido privado, apesar de poder usar outros serviços Web externos como seus componentes. A composição torna-se, dessa maneira, transparente ao usuário.

Pode-se dizer então que os serviços Web são a plataforma mais adequada à composição devido às suas interfaces bem definidas e ao uso de padrões usados para descrever e especificar as suas interações.

2.4 Orquestração e Coreografia

Orquestração e coreografia são dois processos distintos de composição de serviços Web. Nesta seção são detalhadas e expostas as diferenças entre as duas abordagens.

A palavra chave que define o processo de orquestração é centralização. Na orquestração é definido um processo central, que também pode ser um serviço web, que possui o controle sobre os demais serviços e coordena a execução das diferentes funções envolvidas no processo. Nessa operação, os serviços web que compõem a estrutura não sabem que estão sendo coordenados e que participam de um processo que visa um objetivo maior. Apenas o coordenador da orquestração tem uma visão geral do que está acontecendo e pode tomar atitudes baseadas nesse conhecimento. Assim, a orquestração tem suas operações definidas explicitamente além da ordem de execução dos serviços web orquestrados.

Já o processo de coreografia utiliza outra metodologia para fazer a composição de serviços Web. Nesse processo não existe o papel do coordenador que centraliza as ações. Sendo assim, todos os serviços web participantes da coreografia estão cientes que fazem parte de uma estrutura que visa objetivos maiores. Portanto, para alcançar esses objetivos, os serviços web realizam o trabalho em conjunto baseado na troca de mensagens.

2.5 Business Process Execution Language

Linguagem conhecida sob o acrônimo BPEL pode ser traduzida para o português como Linguagem de Execução de Processos de Negócio. É a linguagem mais utilizada na composição de serviços Web. Padronizada pela OASIS (www.oasis-open.org) é capaz de especificar as interações entres os WS. Os processos existentes na linguagem BPEL exportam e importam informações fazendo uso exclusivo de *interfaces* serviço Web.

Um processo BPEL tem o poder de especificar a ordem exata em que os serviços Web serão executados. Assim, é possível expressar desvios condicionais, loops, declarar variáveis, definir o tratamento de exceções, entre outras operações que somadas podem compor complexos processos de negócios.

Conforme Juric (2009) é possível dizer que um processo BPEL consiste de vários passos, denominados ações, ou em inglês, *activities*. Ações primitivas representam construções básicas e realizam tarefas simples. Abaixo são listadas algumas dessas ações presentes na linguagem:

- invoke: invoca outro serviço Web;
- receive: espera uma invocação do cliente;
- assign: utilizado para manipular variáveis;
- throw: utilizado no tratamento de exceções;
- wait: espera por algum tempo determinado;
- terminate: finaliza o processo.

BPEL dá a liberdade de combinar várias ações primitivas como meio de definir algoritmos complexos. Para combinar ações primitivas, são necessárias ações estruturais, dentre as quais podem ser citadas como as mais importantes:

- sequence: permite a execução de operações de maneira ordenada;

- flow: define um conjunto de ações que serão executadas em paralelo;

- switch: realiza desvios condicionais;

- while: realiza ciclos.

Nas seguintes subseções, é enfatizado o funcionamento de alguns dos elementos citados. Desse modo, é explicado o seu modo de funcionamento e exemplificado o seu uso em um processo BPEL.

2.5.1 Elemento invoke

Conforme Erl (2005), este elemento identifica a operação que será invocada pelo processo durante sua execução. Esta operação é implementada por terceiros e aguarda uma requisição. O elemento *invoke* possui cinco atributos que especificam os detalhes da invocação.

```
<invoke name=”ValidateWeeklyHours”
      partnerLink=”Employee”
      portType=”emp:EmployeeInterface”
      operation=”GetWeeklyHoursLimit”
      inputVariable=”EmployeeHoursRequest”
      outputVariable=”EmployeeHoursResponse” />
```

2.5.2 Elemento receive

O elemento *receive* aguarda a requisição de um cliente externo para executar as suas funções (ERL, 2005). Neste caso, o serviço pode ser visto como um provedor de serviços esperando ser invocado.

```
<receive name=”receiveInput”
      partnerLink=”Client”
      portType=”tns:TimesheetSubmissionInterface”
      operation=”Submit”
      variable=”ClientSubmission”
      createinstance=”EmployeeHoursResponse” />
```

2.5.3 Elemento sequence

O elemento *sequence*, de acordo com Erl (2005) permite organizar uma série de operações e atividades a serem executadas em uma ordem seqüencial pré-determinada. A figura 2.3 seguinte apresenta um esqueleto da estrutura *sequence* contendo algumas ações primitivas presentes na linguagem que serão executados em seqüência.

```
<sequence>
      <receive>
      ...
      </receive>
      <assign>
      ...
      </assign>
      <invoke>
      …
      </invoke>
      <reply>
      …
      </reply>
</sequence>
```

2.5.4 Elemento switch

Este elemento possibilita a utilização da lógica condicional similar às construções utilizadas nas linguagens de programação tradicionais (ERL, 2005). O elemento *switch* é responsável por limitar o escopo da lógica condicional, onde várias estruturas *case* podem ser aninhadas testando diferentes condições através da utilização do atributo *condition*. A partir do momento em que esse atributo adquire o valor "verdadeiro", as operações definidas dentro da estrutura *case* são executadas. Por sua vez, o elemento *otherwise* só é executado a partir do momento em que nenhuma condição presente nas estruturas *case* são satisfeitas.

```
<switch>
      <case condition=
            ”getVariableData”(‘EmployeeResponseMessage’),
                              (‘ResponseParameter’ = 0”>
      ...
      </case>
      <otherwise>
      ...
      </otherwise>
</switch>
```

2.6 Comunidade de Serviços Web

Benatallah et al. (2003) definem o conceito de comunidade de serviços Web como sendo uma coleção de serviços Web com uma funcionalidade comum, embora estes serviços web possuam propriedades não funcionais distintas.

Já sob o ponto de vista de Medjahed e Bouguettaya (2005), uma comunidade de serviços Web é um meio de organizar os serviços que compartilham um mesmo domínio de interesse com respeito a uma ontologia.

E segundo Maamar et al. (2007), uma comunidade é definida através de um serviço Web abstrato representativo que conduz à comunidade, sem fazer referência explícita aos serviços Web concretos que estão presentes nessa comunidade e implementam essa funcionalidade em tempo de execução.

2.7 Programação Orientada a Aspectos

Existem muitos problemas de programação em que técnicas de programação procedural ou de orientação a objetos não são suficientes para capturar com clareza algumas das importantes decisões de projeto que o programa deve implementar. Isso faz com que a implementação de tais decisões de projeto acabe ficando espalhada ao longo do código, resultando em um código confuso, extremamente difícil de desenvolver e manter (KICZALES et al., 1997).

É nesse cenário que está inserida a programação orientada a aspectos (POA). A POA é um método de desenvolvimento de software no qual o código é separado e organizado de acordo com a sua importância para a aplicação. O paradigma se propõe a aumentar a modularidade do software incluindo métodos e ferramentas que possibilitam a modularização em nível de código fonte.

Existem certos problemas relacionados à implementação de algumas funcionalidades que linguagens de programação, tais como a programação orientada a objetos, são incapazes de resolver. Esses problemas são chamados de *cross-cutting concerns.* Eles são definidos como um conjunto de funcionalidades partilhadas por diversos métodos do sistema. As características dessas funcionalidades impedem que elas sejam agrupadas em um determinado módulo, fazendo com que acabem aparecendo em diversas partes do código. Uma das principais causas que impossibilita o agrupamento de tais funcionalidades é a maneira singular como elas são utilizadas em cada um dos métodos.

Um exemplo de *cross-cutting concern* é a funcionalidade de um log de dados. Sendo que, na programação orientada a objetos, um log de dados é implementado em uma única classe, ele será referenciado em todas as partes do código que necessitarão fazer o log de seus dados. Partindo do princípio que em uma aplicação quase todos os métodos necessitam fazer o log de seus dados, as chamadas a essa classe ficarão espalhados ao longo de todo o código do sistema. Isso faz com que o programa se torne confuso e de difícil manutenção.

Conclui-se que é difícil implementar tais funcionalidades pois elas fazem parte dos requisitos básicos do projeto, pertencendo não a um módulo, mas sim a todo o sistema. É dentro desse escopo que são utilizadas as técnicas de Programação Orientada a Aspectos, garantindo de maneira eficiente a resolução de tais problemas.

2.7.1 AspectJ

AspectJ é uma extensão da linguagem Java que implementa o paradigma de Programação Orientada a Aspectos. Tornou-se um padrão de POA amplamente utilizado por garantir simplicidade e usabilidade aos usuários-finais. Abaixo são apresentados alguns elementos presentes na linguagem:

- ***Join Point*:** é um ponto existente no fluxo de execução de um programa e especifica a parte do código em que um determinado aspecto deve ser executado. São os pontos do código que satisfazem os *pointcuts*. Em outras palavras, é a parte do código onde ocorrem os *cross-cutting concerns.*
- ***Pointcut*:** é formado por um conjunto de diversos *join points.* Sempre que a execução do programa atinge um dos *join points* descritos, o código associado a esse *pointcut* é executado.

- ***Advice:*** é a parte de código executada no momento em que as condições de um *pointcut* são satisfeitas.

A figura 2.1, seguinte, exemplifica os termos apresentados mostrando a sua utilização em uma aplicação.

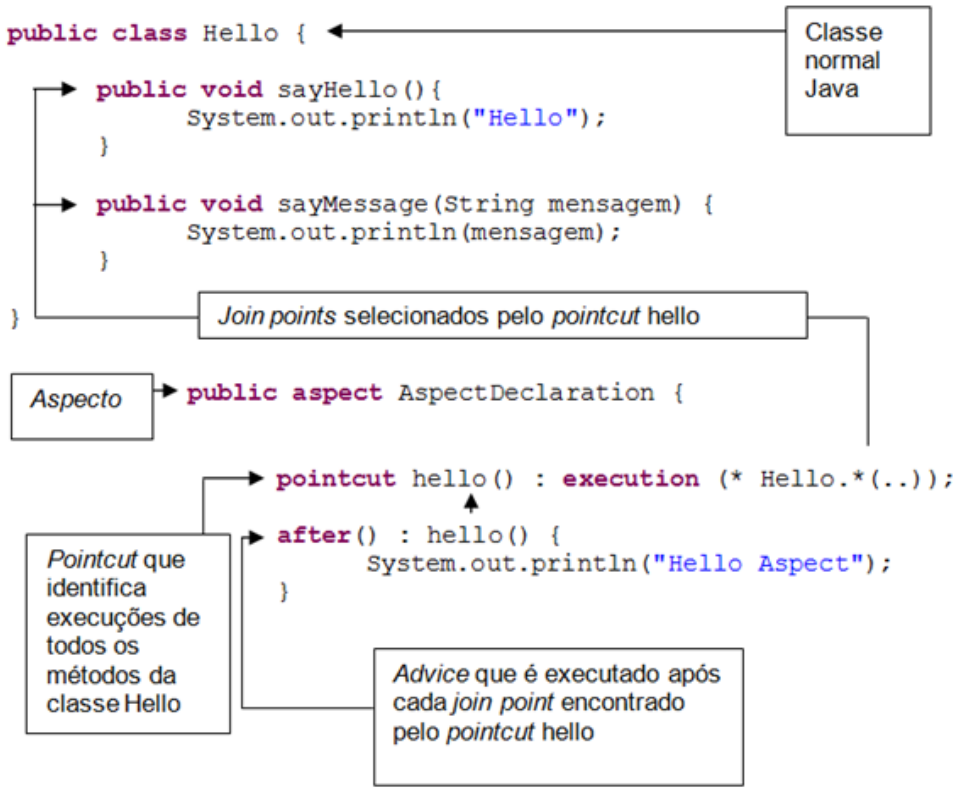

Figura 2.1: Aplicação utilizando POA através de AspectJ (http://javaframework.org)

2.8 Framework

Framework é uma estrutura de suporte definida para auxiliar no desenvolvimento e organização de projetos de software. Ele provê uma funcionalidade genérica através da união de códigos comuns a várias aplicações. Sua utilização durante o desenvolvimento de projetos de software aumenta a qualidade do software produzido além de gerar um ganho de produtividade nesta etapa.

Existem várias definições de framework disponíveis na literatura. Com base no levantamento feito por Lumertz (2010) abaixo são apresentadas algumas dessas definições que merecem destaque:

Framework é uma coleção de classes abstratas e concretas que representam um subsistema, estas classes (abstratas e concretas) podem ser estendidas ou adaptadas para construir um novo subsistema (PREE, 1995).

E para Silva (2000), a abordagem de frameworks orientados a objetos utiliza o paradigma de orientação a objetos para produzir uma descrição de um domínio para ser reutilizada. Sendo assim um *framework* é uma estrutura de classes inter-relacionadas, que correspondem a uma implementação incompleta para um conjunto de aplicações de um domínio, sendo que esta estrutura de classes deve ser adaptada para a geração de aplicações específicas.

De acordo com Gamma et al. (2000), um *framework* predefine os parâmetros de projeto, de maneira que o projetista/implementador da aplicação possa se concentrar nos aspectos específicos de sua aplicação.

3 TRABALHOS RELACIONADOS

Nesta seção são apresentados casos existentes na literatura que desenvolvem ferramentas de monitoramento. A seguir será detalhada a sua forma de implantação e execução, partindo do princípio que o estudo de experiências bem sucedidas irá auxiliar no projeto de um novo monitor de serviços Web compostos.

Zurowska e Deters (2009) apresentam o desenvolvimento de um método de monitoramento baseado em estados. Segundo eles, as causas mais comuns de perda de desempenho de um serviço Web são o aumento do número de requisições ou o decréscimo das capacidades do servidor que hospeda esse SW devido a outras tarefas que não os seus processos. Em ambos os casos citados, o que determina a sobrecarga é o rápido aumento nos tempos de resposta do serviço Web às chamadas realizadas por terceiros. Para serem distinguidos esses dois tipos de sobrecarga e, subsequentemente, tomadas as medidas apropriadas, devem ser analisados os tempos de resposta do SW e os tempos entre as chegadas das requisições.

Este método de análise e monitoramento de serviços Web é concebido através de estados que possuem os seguintes atributos:

- tempo de resposta normal: é o tempo de resposta do servidor para os casos em que a sobrecarga não é detectada. É obtido através da média obtida nas últimas requisições.
- tempo atual entre as chegadas das requisições: tempo entre as últimas requisições enviadas ao servidor.
- status: com base nos atributos acima, o status determina se o servidor está realmente sobrecarregado.

O modelo proposto começa no estado *starting* e passa na sequência para *normal*. Durante a permanência nesse estado, se o tempo de resposta passa a aumentar rapidamente, o estado é alterado para *observing*. Essa etapa, por sua vez, é responsável por verificar se o aumento no tempo de resposta é permanente. Se for constatado que o aumento não é permanente volta-se para *normal*, caso contrário uma das seguintes opções de status devem ser selecionadas: o*verloadingClient* ou *overloadingServer*. Essa escolha é feita com base nos tempos entre as chegadas das requisições. Se esses tempos estão diminuindo podemos supor que o cliente está sobrecarregado e assim o estado a ser atribuído é *overloadingClient*. Porém, se esses tempos permanecem estáveis a opção escolhida é *overloadingServer*. Além dos caminhos já citados, os estados de sobrecarga podem ser atingidos se não ocorrer nenhum evento em relação à requisição feita durante um tempo de resposta muito maior que o normal.

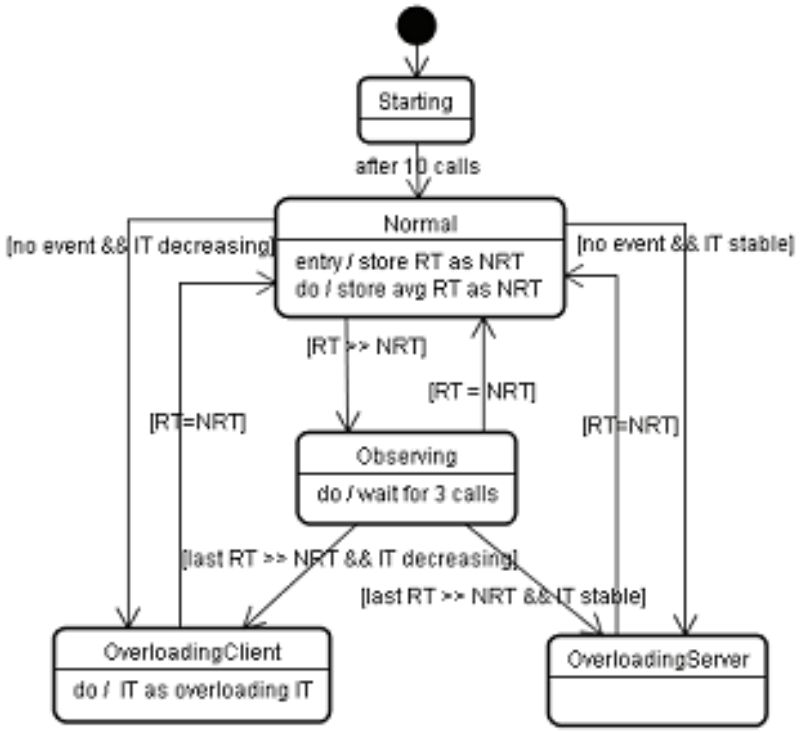

Figura 3.1: Tipos de status de SW (ZUROWSKA e DETERS, 2009)

Este modelo de estados é uma proposta um pouco mais elaborada do que o modelo de monitor proposto. Além de coletar os dados com base nos tempos de operação do servidor, o monitor, através do algoritmo apresentado, consegue estabelecer se o sistema monitorado está sofrendo algum tipo de sobrecarga ou não. Este método consegue tirar conclusões a respeito de seus dados, de forma diferente do monitor abordado neste trabalho, que apenas coleta os dados e os armazena em uma base de dados tal qual um log de operações. Através de uma coleta de tempos mais simplista, registrando os tempos de início e fim dos métodos dos sistemas monitorados e sem a coleta de respostas como no trabalho de Zurowska e Deters (2009), o presente trabalho tenta descobrir se os serviços estão disponíveis e executando de forma aceitável.

Já a proposta de Mao e Le (2009) consiste em realizar a composição de serviços Web baseada na qualidade de seus serviços (QoS). Para tanto, é necessário a elaboração de uma estrutura eficiente que permita alcançar o objetivo traçado. Dentre os componentes dessa estrutura mencionada existe um que realiza a tarefa de monitoramento.

A partir do momento em que um usuário requisita um serviço, o servidor de composição irá verificar as necessidades do usuário através de um agente de composições (*composition broker*), analisar o QoS de vários serviços candidatos e retornar ao usuário o serviço de composição otimizado com base em um algoritmo determinado. Quando o usuário invocar o serviço indicado o agente de composição irá monitorar a sua execução. Através desse monitor é possível determinar a qualidade da execução. Caso essa qualidade dos serviços seja inferior que a esperada, será iniciado um programa que gere novos acoplamentos entre os serviços.

A estrutura proposta também possui um repositório SLA (*Service Level Agreement*). O SLA é um acordo formal entre o provedor do serviço e o seu usuário capaz de garantir um bom desempenho na prestação do serviço. Antes da publicação do SLA no

repositório existente, também se faz necessário a utilização de um sistema de monitoramento dos serviços Web envolvidos, com a finalidade de testar seu desempenho e eficiência.

O foco principal do trabalho de Mao e Le (2009) está na composição de serviços Web através de uma abordagem voltada para a análise do QoS. A forma como o monitor é implementado não ganha muito destaque no trabalho e se torna um pouco obscuro o modo como o sistema aborda a questão do monitoramento. O foco recai sobre a forma com que os serviços serão compostos com base nos QoS apresentados. Esse trabalho acrescenta, então, informações sobre a qualidade dos serviços prestados e não informa com detalhes como ocorre a coleta de dados. Porém, conforme citado, a medida em que este trabalho teve uma mudança no seu enfoque, as informações que deveriam ser monitoradas acabaram cedendo lugar para o estudo da forma com que o monitor coletaria tais informações, se distinguindo do trabalho de Mao e Le (2009).

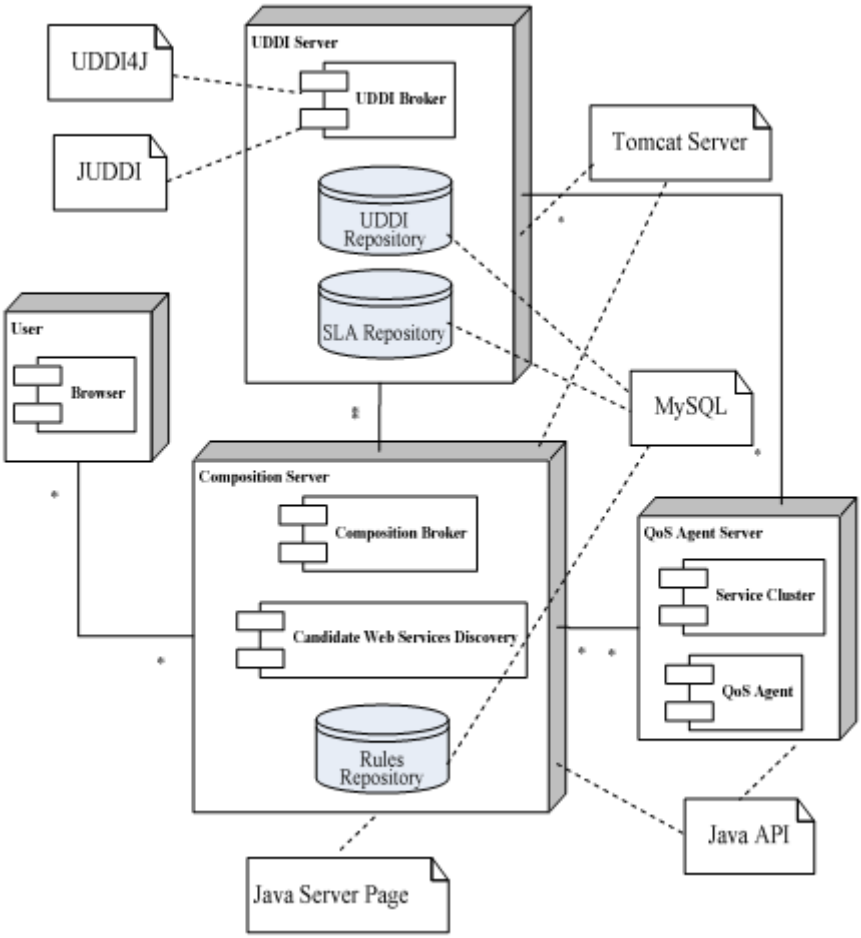

Figura 3.2: Projeto do QoS-Based Web Service Composition (MAO e LE, 2009)

4 MONITOR DE INTERAÇÕES DE SERVIÇOS WEB COMPOSTOS

Este capítulo descreve os detalhes de projeto e implantação de um monitor de interações de serviços Web. O monitor constitui um módulo da estrutura do *CWSMart* e tem como responsabilidade monitorar os serviços Web presentes no repositório. A seguir é descrita a arquitetura geral de um *CWSMart*, conforme definido por Maamar et al. (2009), assim como a arquitetura do módulo monitor pertencente a essa estrutura. Também são abordados os detalhes do *framework*, estendido para auxiliar na monitoração, e são descritos os passos traçados para efetivamente monitorar os serviços Web presentes em um *CWSMart*.

Nessa seção é válido ressaltar que o objetivo do trabalho, que inicialmente se propunha a estudar o que seria importante monitorar, se desviou do tema proposto para tratar o problema de como seria realizada essa monitoração de modo transparente e sem causar prejuízos aos sistemas monitorados. Assim, essa seção começa apresentando o monitor e o escopo onde ele se insere. Apresenta as informações monitoradas, que são relativamente simples e não condensam um grande nível de informação. E por fim entra na parte que realmente se tornou o objetivo do trabalho: definir uma forma de monitoração que fosse transparente aos sistemas monitorados.

4.1 Arquitetura Genérica do CWSMart

Uma arquitetura genérica de quatro camadas para o desenvolvimento e gerenciamento dos *CWSMarts* é proposta por Maamar et al. (2009). Essa arquitetura representa o modo como os serviços Web são agrupados até formarem os chamados *marts* de serviços Web compostos. Ela não especifica os detalhes de um *CWSMart*, mas sim o escopo geral em que ele está inserido. As camadas em questão, propostas pela arquitetura, são: *component*, *community*, *composite* e *mart*. As camadas *component* e *composite* são as bases sobre as quais as camadas *community* e *mart* são respectivamente desenvolvidas. A seguir são detalhadas as funções de cada camada e as relações estabelecidas entre elas.

A primeira camada, denominada *component*, é composta por serviços Web componentes, desenvolvidos e descritos por provedores que os publicam em registros dedicados, como o UDDI, aguardando requisições de clientes. Esses serviços Web implementam funcionalidades capazes de satisfazer certa necessidade particular, como, por exemplo, a reserva em um quarto de hotel.

A segunda camada, nomeada *community*, hospeda serviços Web componentes e os divide em grupos que possuem as mesmas funcionalidades. O conteúdo do nível *component*, independente da forma como é descrito, atualizado ou publicado, é

responsável por alimentar o nível *community*. O único quesito relevante a essa associação são as funções desempenhadas pelos serviços Web. Dessa forma, os serviços Web são agrupados em *communities*, de acordo com a sua similaridade funcional. Segundo Bui e Gacher (2005), apesar da heterogeneidade de serviços Web, suas funcionalidades são suficientemente bem definidas e homogêneas para permitir a concorrência de mercado.

Subindo para a terceira camada, denominada *composite*, é realizada a integração de serviços Web componentes, pertencentes a diferentes *communities*, em serviços Web compostos. Essa camada surge das necessidades complexas dos usuários nos dias atuais e, assim como as demais camadas, obtém o seu conteúdo a partir do nível inferior. Segundo Maamar et al. (2009), esse processo é realizado em duas etapas. A primeira identifica as *communities* que se encaixam com as funcionalidades necessárias ao usuário. E a segunda etapa seleciona, a partir das *communities* encontradas na primeira etapa, os serviços Web que implementarão tais funcionalidades. Cada *community* selecionada contribui com exatamente um serviço Web que fará parte do serviço Web composto resultante.

Finalmente, a quarta camada é definida como *mart.* Ela é a contraparte da camada *community*, porém hospeda serviços Web compostos ao invés de serviços Web componentes. Dessa maneira são formados *marts* de serviços Web compostos (de modo análogo às *comunities* de serviços Web componentes). Essas composições são divididas em grupos com as mesmas funcionalidades, formando os *Composite Web Services Marts* (*CWSMarts*).

Abaixo, na Figura 2.3, é apresentada a arquitetura de 4 camadas descrita anteriormente, que rege a construção de um *CWSMart*.

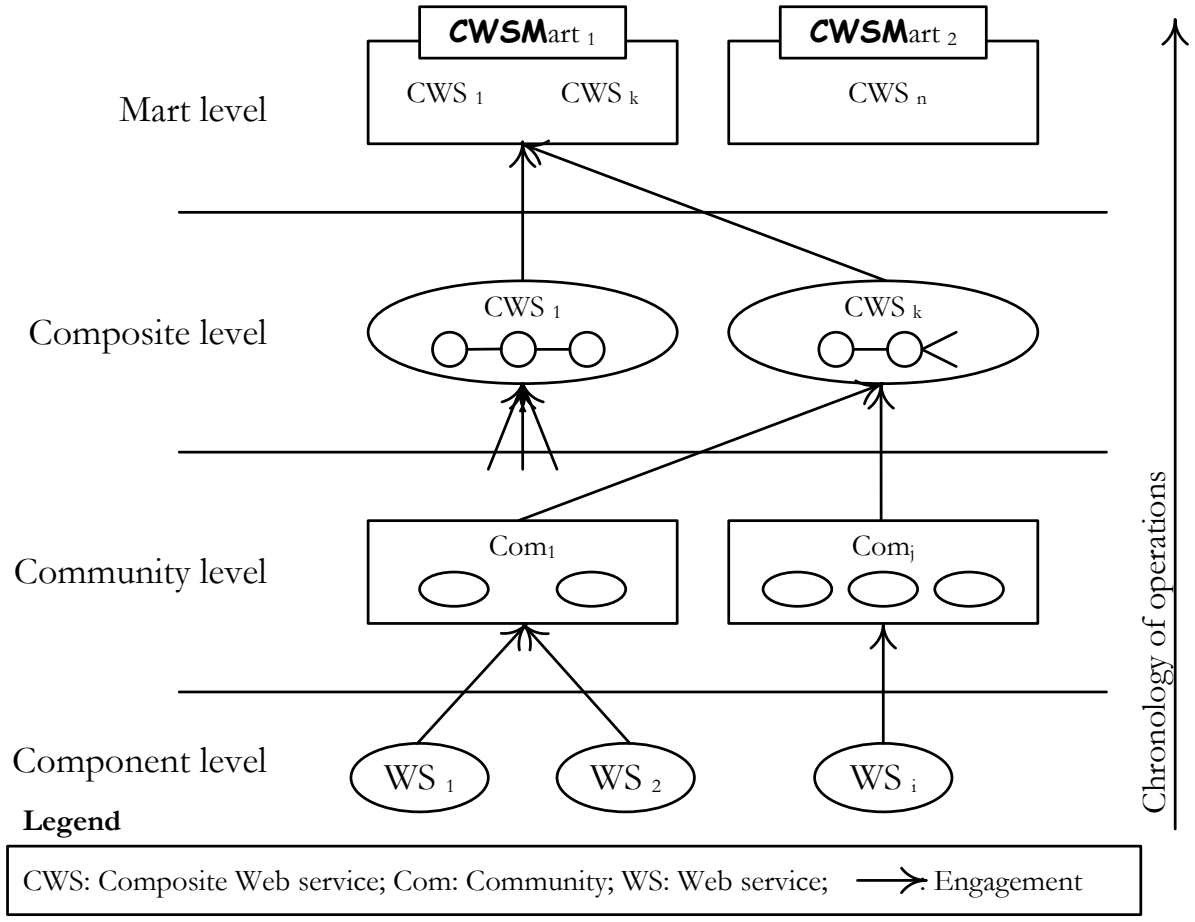

Figura 4.1: Arquitetura para suportar os *CWSMarts* (MAAMAR et al., 2009)

4.2 Estrutura e Especificação do CWSMart

Um *CWSMart*, acrônimo para *Composite Web Service Mart*, é um repositório de especificações de serviços Web compostos capaz de hospedar componentes de acordo com a semelhança de suas funcionalidades. A estrutura de um *CWSMart* consiste em um repositório de descrições e um conjunto de módulos divididos de acordo com a tarefa a ser desempenhada.

A função dos *Composite Web Services Marts* é hospedar serviços Web compostos que possuam as mesmas funcionalidades, independente de como são especificados, da forma como operam, etc. (MAAMAR et al., 2009). Portanto, a característica básica dos *CWSMarts* é formar grupos de serviços Web baseados, primordialmente, em suas funções, desconsiderando assim as outras características envolvidas. O objetivo e as funcionalidades dos módulos são detalhados a seguir:

- **Módulo *management*:** responsável por atender às requisições feitas pelos usuários e posteriormente executar os *CWS*. É através deste módulo que o repositório de descrições do *mart* é consultado em busca de serviços Web que satisfaçam a requisição realizada. No momento em que o *CWS* apropriado é encontrado, o módulo *management* se encarrega de implantar este serviço nas plataformas computacionais presentes.
- **Módulo *matcher*:** compara semanticamente as funcionalidades de um dado serviço Web composto com as funcionalidades do *mart* candidato a hospedá-

lo (MAAMAR et al., 2007). Com base nos resultados obtidos, o módulo decide se o *CWS* candidato será efetivamente integrado ao *mart*. Em caso de aprovação, o *CWS* é encaminhado para o módulo *description*, responsável por inserir o serviço web composto no *mart*. Para os casos em que não há aprovação por parte do módulo *matcher*, o *CWS* é encaminhado ao módulo *recommendation*, onde serão recomendados possíveis *marts* que se enquadrem às suas funcionalidades.

- **Módulo *recommendation*:** conforme explicado anteriormente, este módulo é acionado no momento em que o módulo *matcher* rejeita a inserção de um *CWS* em um *mart*. Sendo assim, este módulo tem a finalidade de recomendar outros *marts* que possam se encaixar com as funcionalidades do *CWS* e consequentemente aceita-lo em seus repositórios.
- **Módulo *description*:** ao contrário do módulo *recommendation*, o módulo *descriprion* entra em execução quando o módulo *matcher* aprova a inserção do *CWS* candidato no *mart* em questão. Sua principal responsabilidade é analisar a especificação do *CWS* e gerar, a partir dessa, outra especificação em termos de atributos. Finalmente, então, esta especificação é armazenada no repositório.
- **Módulo *mining*:** responsável por coletar as informações fornecidas pelo módulo *monitoring* e, então, alimentar o *data mart*. Também disponibiliza meios para a análise dos dados do *data mart* e do repositório de especificações do *CWS*, possibilitando assim, a busca por padrões recorrentes no *mart* ou entre os *CWS* residentes no *mart*.
- **Módulo *monitoring*:** depois de feita a inserção do serviço Web composto no *mart* o *CWS* é posto então em execução. Nessa etapa é que entra em ação o modulo *monitoring*. Ele monitora os serviços Web compostos em tempo de execução e realiza a coleta de informações que serão repassadas para o módulo *mining*. Entre essas informações se destacam o tempo necessário para a execução de um serviço, o sucesso da operação, entre outros.
- **Módulo *agent:*** a partir de consultas realizadas ao módulo *mining*, o módulo *agent* faz uma análise a respeito do uso do *mart*. A partir desses dados, obtém os serviços Web compostos e seus respectivos serviços Web componentes que não estão sendo utilizados e que, consequentemente, devem ser removidos do repositório. A tarefa de remoção, por sua vez, é realizada pelo módulo *garbage collection*.
- **Módulo *garbage collection*:** gerencia o processo de remoção de *CWS* não usados do repositório. Função vital capaz de garantir o desempenho e a qualidade (*QoS*) do *CWSMart*.

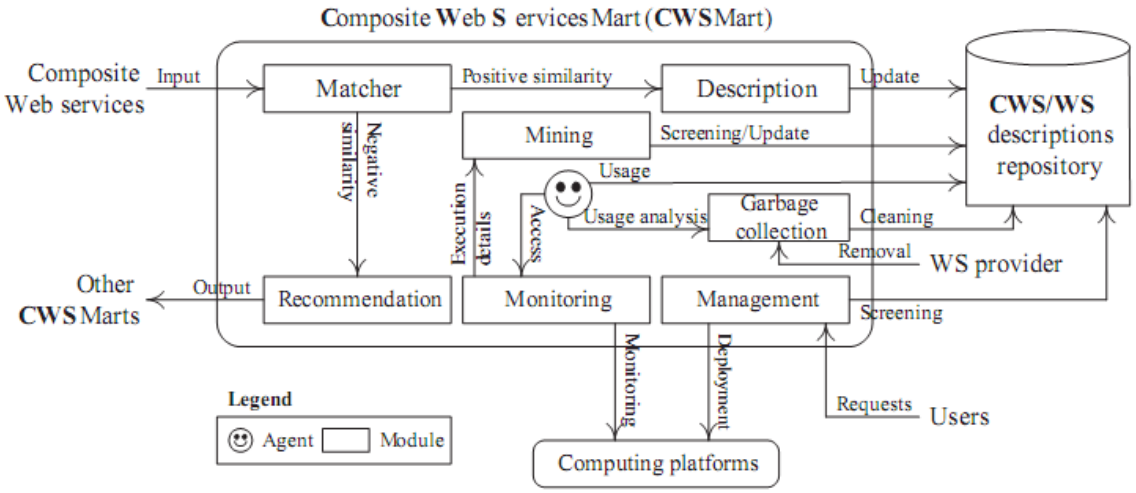

Figura 4.2: Estrutura interna de um *CWSMart* (MAAMAR et al., 2009)

Levando em conta tal arquitetura e especificação, Lumertz (2010) desenvolveu um framework para o desenvolvimento de *CWSMarts*. O *framework* contém as classes e operações de nível mais abstrato para o desenvolvimento de *marts* de serviços Web compostos, fornecendo algoritmos muito básicos e simples para a maioria dos componentes (sendo que alguns deles não foram especificados, tais como *garbage collector*, *monitoring*, *mining* e *management*). Maiores detalhes sobre o framework são descritos na seção 4.4. O trabalho atual estende o *framework* proposto por Lumertz, desenvolvendo um módulo de monitoramento mais elaborado e completo. A arquitetura de tal módulo é descrita a seguir.

4.3 Arquitetura do Monitor

O monitor é um módulo pertencente à estrutura do *CWSMart*. Os alvos de suas funções de monitoramento são os serviços Web compostos presentes nesse *mart*. Seu papel é coletar informações referentes aos serviços Web em tempo de execução e repassá-las para outros módulos capazes de interpretá-las. Como serviços Web compostos são formados pela agregação de serviços Web componentes, o monitor atua diretamente sobre esses SW componentes, partindo do princípio de que ao monitorar todas as partes envolvidas no processo se monitora o comportamento do processo como um todo.

Existem pelo menos duas abordagens referentes ao modo como o monitor pode ser desenvolvido. A primeira está embasada na premissa de que os sistemas monitorados não sabem que estão sendo monitorados (e tal monitoração é realizada de maneira transparente). Já a segunda considera que os sistemas monitorados estão cientes a respeito desse fato e explicitamente chamam o monitor para registrar suas atividades. Na primeira abordagem o monitor se torna o responsável integral pela aquisição dos dados necessários para o monitoramento. Sem qualquer ajuda adicional dos sistemas monitorados para auxiliar na obtenção da informação. Um exemplo de busca de dados realizada por um monitor que segue tais princípios é a análise de *logs* de execução de serviços Web. Em contrapartida, na segunda abordagem de desenvolvimento, os sistemas monitorados estão cientes sobre esse fato, e auxiliam o monitor provendo informações necessárias durante o processo de monitoramento.

Dentre as abordagens apresentadas, o monitor desenvolvido opera de acordo com a segunda. Essa abordagem foi adotada por oferecer maior simplicidade na implantação e por evitar a sobrecarga de atividades do monitor. Existe a desvantagem de que os sistemas monitorados deverão implementar funções extras, não relacionadas às suas atividades, porém, desse modo, é possível distribuir as responsabilidades de monitoramento. Logo, estando cientes de que estão sendo monitorados, esses sistemas colaboram enviando informações referentes à sua execução ao monitor.

Em termos de implantação, o monitor, assim como os sistemas monitorados, também é um serviço Web que opera dentro do *CWSMart*. Ele está sempre ativo a espera de mensagens enviadas pelos *CWS*, representando um ponto central de coleta de informações. Em relação ao armazenamento das informações coletadas, o monitor possui uma base de dados relacional, operando de forma similar a um *log* de dados. Ela é constantemente alimentada e registra todas as informações referentes às várias execuções dos serviços Web. Essa base de dados também é acessada por outros módulos do *CWSMart*, que farão uso de tais informações de acordo com as funções a que se propõem.

Para que o monitor seja acessado pelos demais serviços Web, ele provê um método público encarregado de atender às recorrentes chamadas às suas funções. Esse método é o responsável por receber as requisições com as informações sobre a execução dos *CWS*. Ele recebe cinco parâmetros distintos necessários para registrar a chamada. Dentre eles estão: o IP do serviço Web responsável por prover o serviço, o IP do cliente que está requisitando tal serviço, a data e a hora em que se deu a execução, o tipo de operação realizada e finalmente o status atribuído a essa operação.

Quanto aos parâmetros que devem ser enviados ao monitor, os dois IPs são registrados com a finalidade de estabelecer a rota traçada desde a saída da requisição do serviço até a chegada ao seu provedor de destino. Além disso, com os parâmetros de status de cada operação, é possível registrar o tipo de tarefa executada pelo serviço Web e o status referente a essa tarefa, como, por exemplo, se ela foi iniciada, finalizada, etc.

4.4 Informações Monitoradas

Um ponto importante a ser destacado no desenvolvimento de um monitor é a escolha dos atributos a serem monitorados. O levantamento sobre os dados que merecem ser capturados deixará a base de dados com informações mais ricas e agregará um maior conhecimento acerca da execução dos serviços Web compostos.

Dentre as informações coletadas pelo monitor estão: o identificador (Ex.: IP) do serviço Web responsável por prover o serviço, o identificador (Ex.: IP) do cliente que está requisitando o serviço, o *timestamp* do momento em que ocorreu o registro da operação, o nome da operação realizada e o status atribuído a essa operação. Esses dados são obtidos sempre no início da execução da operação e no seu final.

Com base nessas informações coletadas é possível inferir algumas conclusões ao analisar a base de dados. Ao obter o serviço Web que está provendo o serviço, o cliente que o está requisitando e os *timestamps* relativos às operações de cada um, se tem uma ideia do tempo necessário para a requisição ser atendida. Também se torna possível avaliar o tempo que o serviço Web leva para prestar o seu serviço quando recebe a requisição. A partir de tais dados se tem uma noção da disponibilidade dos serviços, além de saber se tal serviço consegue executar a tarefa que lhe é proposta ou não.

A partir dessas informações monitoradas se consegue traçar a forma como um serviço Web composto está executando. Provendo tais dados, os módulos responsáveis pela sua análise possuem uma boa fonte de conhecimento para estudo. Assim, podem agir para descobrir possíveis serviços Web que não estão correspondendo às expectativas e prover formas de melhorar o serviço desempenhado pelos serviços Web compostos no *CWSMart*.

Em um momento inicial, entre os atributos monitorados estavam a latência, o throughput e a disponibilidade dos serviços. Porém conforme o projeto foi avançando, outras questões surgiram para ganhar destaque. A principal delas foi a forma como esses dados seriam coletados. Dificuldades foram encontradas para estabelecer um método de monitoração transparente aos sistemas monitorados. Consequentemente, os atributos a serem monitorados se tornaram mais simples, registrando as operações e o status referente a elas, conforme apresentado na figura 4.3.

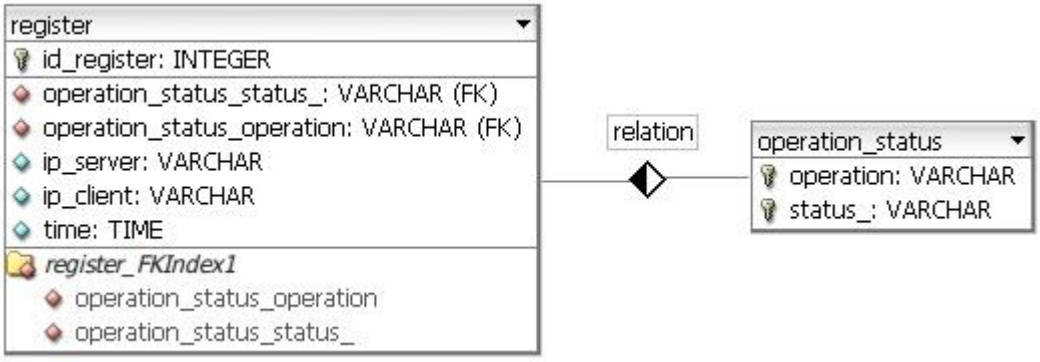

Figura 4.3: Diagrama ER da base de dados do monitor.

4.5 Framework para Marts de Serviços Web Compostos

O *Framework* para *Marts* de Serviços Web Compostos permite a definição de *CWSMarts* específicos. Ele define e especifica módulos e funcionalidades básicas de um *CWSMart* capazes de ser estendidas. Esse *framework* é descrito em Lumertz (2010). A seguir, é apresentada a arquitetura do *framework* de forma abrangente e recomenda-se, para um entendimento pleno dos detalhes do projeto, a leitura de seu trabalho.

Um *framework* é um conjunto de classes cooperativas que implementam os mecanismos que são essenciais para um domínio de problemas específicos. Um programador pode criar uma funcionalidade nova no domínio do problema estendendo as classes do *framework*. Ao contrário de um padrão de projeto, um *framework* não é uma regra geral de projeto. Normalmente, um *framework* usa múltiplos padrões (HORSTMANN, 2007).

Ainda, segundo Horstmann (2007), e conforme citado por Lumertz (2010), um "*framework* de aplicação" consiste em um conjunto de classes que implementa serviços comuns a certo tipo de aplicações. Para criar aplicações reais, o programador cria subclasses a partir de algumas classes do *framework* e implementa funcionalidades adicionais que são específicas da aplicação que ele está construindo. Em um "*framework* de aplicação", as classes do *framework*, e não as classes específicas da aplicação, controlam o fluxo de execução. Esse fenômeno normalmente é chamado de "inversão de controle".

O *framework* para *Marts* de Serviços Web Compostos se encaixa na categoria de "*framework* de aplicação" e implementa os módulos especificados na seção 4.2 de modo que eles possam ser estendidos. A classe principal de cada módulo presente no *framework* é abstrata, fazendo com que a aplicação que o esteja utilizando precise estender essas classes, implementado algumas das funcionalidades e algoritmos.

Portanto, para utilizar os métodos disponibilizados é necessário, ao instanciar o *framework*, passar como parâmetro o nome da classe que implementa a "fábrica" concreta de módulos. Assim, sempre que é executado um método público do *framework* pela primeira vez, a "fábrica" entra em ação instanciando os módulos envolvidos na operação. Desse modo, o *framework* controla o fluxo de execução, aplicando o conceito de "inversão de controle" e utilizando, quando necessário, os métodos implementados pela aplicação.

O *framework* contém as classes e operações de nível mais abstrato para o desenvolvimento de *marts* de serviços Web compostos, fornecendo algoritmos muito básicos e simples para a maioria dos componentes. Alguns componentes, inclusive, não foram especificados, tais como *garbage collector*, *monitoring*, *mining e management.*

4.6 Extensão do Framework

Conforme relatado anteriormente, o monitor foi desenvolvido seguindo o princípio de que os sistemas monitorados estão cientes sobre esse fato e auxiliam consequentemente na coleta de informações. Porém, é imprescindível que esse método de monitoração não cause prejuízos aos sistemas monitorados, obrigando que eles implementem uma série de novas funções que não correspondem ao real escopo de suas atividades. Com base nesse pressuposto, se torna necessário a implantação de uma ferramenta que auxilie os sistemas monitorados no desenvolvimento e execução das tarefas relacionadas ao módulo de monitoramento.

Sendo assim, para suprir tais necessidades o *framework* apresentado na seção 4.4 foi estendido para prover funções básicas relacionadas à forma com que o monitor será utilizado. Até então o *framework* não implementava nenhuma funcionalidade relacionada ao módulo de monitoramento, deixando a implementação de seus métodos a critério das aplicações que o utilizavam.

Dentre as funções adicionadas com a extensão do *framework* consta todo o processo relacionado à conexão e interação com o serviço web monitor. As classes do *framework* proveem métodos que chamam as operações implementadas pelo monitor, evitando que os clientes desperdicem esforços desenvolvendo tais métodos de comunicação com o serviço Web. Retira-se, dessa forma, a responsabilidade, por parte dos clientes, de desenvolver métodos responsáveis pelo estabelecimento da conexão e comunicação com o sistema de monitoramento.

4.7 Aplicação de Aspectos aos Sistemas Monitorados

A partir do momento em que foi definida a abordagem utilizada pelo monitor, também foi definido que, mesmo os sistemas monitorados estando cientes desse fato e colaborando com o envio de informações, esse processo deve ser feito de forma transparente aos mesmos. Evitando que cada serviço Web monitorado necessite implementar uma série de métodos intrusivos ao escopo de suas funcionalidades.

Para atingir tais requisitos, com o intuito de garantir a modularização e a organização do código, atuou-se em duas frentes distintas. A primeira delas foi a extensão do *framework* tratado na seção 4.5. Com essa prática o cliente se exime do trabalho de implementar os métodos para conexão e comunicação com o monitor. Porém, o uso exclusivo dessa prática não garante a total transparência das atividades de monitoração, já que os sistemas monitorados ainda serão os responsáveis por chamar tais métodos sempre que uma atividade deva ser registrada. A segunda frente age com o intuito de sanar essa limitação e está relacionada ao uso do paradigma de Programação Orientada a Aspectos.

O objetivo de usar tal paradigma de programação é evitar que o código dos sistemas monitorados se torne confuso e de difícil manutenção pelo fato de serem adicionadas chamadas referentes ao monitor no escopo de suas funcionalidades. A chamada dos métodos do monitor terá a função similar à de um log de dados, sendo que praticamente toda a operação executada pelos serviços web deverá ser registrada pelo módulo de monitoramento. Portanto, essas chamadas são espalhadas ao longo de grande parte do código dos sistemas monitorados, representando um problema de *cross-cutting concern* a essas aplicações.

De acordo com o que foi apresentado, a Programação Orientada a Aspectos é um paradigma que garante uma solução para os problemas de *cross-cutting concern.* Portanto, para sanar esse problema foi introduzida, nos serviços web monitorados, uma classe que implementa um aspecto responsável por agrupar as chamadas aos métodos do monitor.

É definido então que para cada método presente nos sistemas monitorados serão definidos dois *join points*. *Join points* são as partes do código em que o fluxo de execução é desviado para o aspecto. O primeiro deles será definido a partir do momento em que qualquer método presente no sistema monitorado começa a sua execução e o segundo no momento em que esse método termina de executar. Portanto, sempre que um método for executado, esses *join points* irão disparar um *advice* presente na classe que implementa o aspecto. Esse *advice* por sua vez irá executar os métodos referentes à coleta de informação por parte do monitor. Uma vez que os *join points* foram definidos da mesma forma para todos os sistemas monitorados, a classe responsável por implementar o aspecto será a mesma em todos os clientes.

Dessa maneira a execução dos métodos de monitoramento se torna transparente aos sistemas monitorados e não interfere no código da aplicação. Deixando-o claro e sem comprometer a manutenção do código. A seguir o código que implementa o aspecto é apresentado e o significado de suas funções é detalhado.

```
public aspect Interceptor {

    before(MonitoredWebService mws): target(mws) && execution(* *(..))
    {
      String methodName =
      thisJoinPointStaticPart.getSignature().getName();
      mws.insertRegister(mws.WScontext, methodName, "begin");
    }

    after(MonitoredWebService mws): target(mws) && execution(* *(..))
    {
      String methodName =
      thisJoinPointStaticPart.getSignature().getName();
      mws.insertRegister(mws.WScontext, methodName, "end");
    }

}
```

O aspecto de nome *Interceptor* presente nos serviços Web monitorados define dois *advices* em seu corpo. Esses *advices* são executados sempre que o sistema monitorado executar algum de seus métodos. Será analisado apenas o *advice* executado a partir da cláusula *before*, considerando que o *advice* sob a cláusula *after*, tem praticamente o mesmo funcionamento, se diferenciando apenas pelo fato de ser chamado após a execução dos métodos.

Na primeira linha de código é definido o *pointcut* que deverá ser satisfeito para que ocorra a execução do respectivo *advice*. Sintaticamente, o *pointcut* é descrito à direita do símbolo “:”. Neste caso, o *pointcut* busca qualquer método que seja disparado pelo objeto de tipo MonitoredWebService. A palavra-chave *target* é quem faz a exigência de que o método seja disparado por um objeto específico e a palavra-chave *execution* é quem faz as exigências em relação ao nome dos métodos. Para finalizar, a palavra-chave *before* informa que o respectivo *advice* será executado antes do método que satisfez as condições do *pointcut*.

```
before(MonitoredWebService mws):target(mws)&& execution(* *(..))
```

Conforme definido anteriormente, um *advice* representa a parte de código executada no momento em que as condições de um *pointcut* são satisfeitas. Sendo assim, o *advice* é representado pela seguinte parte de código:

```
String methodName =
thisJoinPointStaticPart.getSignature().getName();
mws.insertRegister(mws.WScontext, methodName, "begin");
```

A primeira linha de código captura o nome do método que satisfez as condições do *pointcut*. Já a segunda linha de código executa o método presente no *framework* responsável por enviar as informações ao monitor.

Conforme explicado, a parte do código executada a partir da cláusula *after* possui a mesma funcionalidade, porém é disparada sempre após a execução dos métodos que satisfazem o *pointcut*.

Para finalizar, é importante destacar que a utilização do *framework* implementado em Java e o uso de AspectJ acaba se tornando um limitante ao monitor e aos sistemas monitorados. Isso se deve ao fato de que a implementação dos serviços Web monitorados seja feita utilizando exclusivamente a linguagem Java, tornando o serviço Web dependente de uma tecnologia específica. Assim, o monitor será capaz de coletar dados advindos apenas de serviços Web que se encaixam nessas condições, limitando

suas funções de monitoramento. Dessa forma, esse fator limitante vai contra os princípios da SOA de que os serviços devam ser independentes de plataforma. Porém, mesmo limitando os serviços Web que estão no conjunto de componentes monitorados, essa abordagem é um caminho viável sendo que esse trabalho objetiva dar um primeiro passo e fazer uma proposta inicial de monitoramento que pode ser trabalhada e evoluída no futuro.

5 CONCLUSÕES

O trabalho realizado objetivou a construção de um módulo monitor de serviços Web para a estrutura de um *Composite Web Services Mart* e, principalmente, a diminuição do impacto da monitoração nos sistemas monitorados. Para isso envolveu primeiramente o estudo do estado da arte dos *CWSMarts*, englobando, dessa forma, os conceitos de serviços Web e serviços Web compostos. De acordo com a evolução do trabalho foi definido o modo de operação do monitor e então acrescentados novos elementos ao escopo do projeto, como a utilização do paradigma de programação orientada a aspectos e a extensão do *framework* para *marts* de serviços Web compostos. É importante destacar que a implantação do monitor em si, perdeu espaço, ao longo do desenvolvimento do trabalho, para a resolução do problema referente ao impacto desse monitor nos sistemas monitorados.

Esse trabalho é um primeiro passo na monitoração de sistemas e tem sua atuação limitada por diversos fatores que devem ser refinados no futuro. Ele contribui a partir do momento em que consegue, a partir da abordagem em que os sistemas monitorados auxiliam na monitoração, deixar as funções e chamadas relativas ao monitor transparentes aos componentes monitorados. Esse objetivo foi atingido através do uso de aspectos, e apesar de não ser a solução ideal para o problema, mostra que é algo viável como um meio inicial e que pode ser incrementado e melhor trabalhado no futuro.

A seguir é feita uma análise a respeito do presente trabalho, apresentando os resultados obtidos, as limitações do projeto e perspectivas sobre essa área de estudo.

5.1 Resumo de Resultados

O comportamento do monitor foi estudado através de sua aplicação prática na monitoração de serviços Web. Os serviços Web monitorados executam operações simples com a única função de servir de exemplo para disparar o processo de monitoração. Através do ambiente estabelecido, o monitor conseguiu realizar a coleta de dados sempre que lhe foi requisitado. Sempre que os sistemas monitorados executavam algum método, essa operação era persistida pelo monitor.

Analisando a parte correspondente ao envio das informações no lado do cliente, sempre que lhe foi requisitado, ele executou o seu respectivo método e o aspecto implementado foi capaz de interceptar a chamada com sucesso, realizando o envio de informações ao monitor.

Mas além de realizar a coleta de dados citada, o trabalho conseguiu atingir seu objetivo final de fazer com que as chamadas ao monitor fossem transparentes aos

sistemas monitorados. Apesar de usar AspectJ, e limitar os serviços Web ao uso de Java, como uma primeira abordagem, a execução dos métodos foi automática e sem a necessidade de introduzir código adicional às classes dos sistemas monitorados.

Portanto, dentro de um escopo limitado, o trabalho cumpre com a proposta de tentar minimizar os efeitos sofridos pelos sistemas monitorados durante a monitoração. Também realiza de uma forma simples a coleta dos dados que, em um primeiro passo, não são os ideais a serem monitorados, mas servem como uma primeira abordagem aberta a refinamentos.

5.2 Limitações do Trabalho

Através da elaboração deste trabalho, foram encontrados muitos obstáculos no seu desenvolvimento. Desde a escolha das informações que devem ser monitoradas até o modo como o monitor deve ser implementado. Sendo este trabalho uma proposta inicial de monitoramento de serviços, ele possui uma série de limitantes que permitem o seu funcionamento apenas em um determinado escopo de operação.

É possível considerar um limitante para o trabalho a falta de alguns atributos relacionados ao desempenho da rede na coleta de dados realizada pelo monitor. Os atributos coletados são basicamente referentes aos tempos de início e fim das operações, além do nome da operação realizada e de seu status. O monitor não captura dados referentes à rede em que os serviços Web estão operando. Isso acaba sendo um limitante, pois não é possível fazer um levantamento a respeito do fluxo de dados da rede que está sendo utilizada. Assim não é possível saber ao certo a capacidade de tráfego da rede e se as requisições estão sendo encaminhadas aos serviços Web em um tempo razoável. Porém, através da análise dos dados coletados é possível estabelecer conclusões a respeito do fluxo de dados, contornando um pouco essa deficiência.

Outro fator a se destacar diz respeito às informações monitoradas. A partir do momento em que o foco do trabalho foi alterado, a escolha das informações que deveriam ser monitoradas acabou sendo deixado em segundo plano. Por essa razão, as informações monitoradas são um pouco simplistas e não expressam os dados ideais para que se possa avaliar, com clareza, a execução de um serviço Web.

O uso de AspectJ também é considerado um limitante ao trabalho. Apesar de a linguagem suprir as necessidades, através do uso de aspectos, de tornar o monitoramento transparente aos sistemas monitorados, ela acaba impondo certas restrições. Com o seu uso, os sistemas monitorados ficam limitados ao uso da linguagem Java, não podendo ser implementados utilizando outra linguagem de programação. Conforme citado anteriormente, isso acaba indo contra os princípios da SOA, de que cada serviço Web pode ser implementado independente de uma tecnologia, e então implica em uma restrição que limita o grupo de serviços Web que poderão ser monitorados.

Também é válido acrescentar, que ao tentar abraçar questões de natureza distinta, o trabalho acabou se tornando um tanto quanto abrangente. Assim, ao tentar contornar vários obstáculos, o projeto apresentou várias limitações e acabou contribuindo de uma maneira diferente à que estava proposta incialmente.

5.3 Perspectivas

Como perspectivas futuras, é possível levantar o estudo de métodos que façam com que os serviços Web não dependam da tecnologia para serem monitorados. Utilizando alguma outra forma que permita a total transparência das funções relacionadas ao monitoramento.

Também é necessário um estudo mais aprofundado com relação às informações que devem ser monitoradas, já que o presente trabalho desempenha essa tarefa de um modo um tanto quanto superficial. Dessa forma seria interessante um estudo que abordasse com detalhes o que seria realmente importante para o módulo *mining.*

Além disso, não existe apenas uma maneira de realizar a monitoração de serviços Web. Esse é um módulo que pode ser proposto de formas diferentes para depois ser realizada uma comparação entre as demais abordagens. Conforme apresentado, a abordagem desse projeto é embasada no fato de que os serviços Web sabem que estão sendo monitorados. A outra abordagem ainda está aberta a estudos e pode representar um trabalho futuro. Um *CWSMart* pode implementar, inclusive, mais do que um monitor, ficando a critério do usuário a escolha de qual atende melhor suas necessidades.

Com o desenvolvimento do módulo monitor se abre o caminho para o estudo do módulo *mining.* Esse módulo será o responsável por analisar as informações coletadas pelo monitor e traçar conclusões com base nesses dados. Será possível, a partir da implementação do módulo *mining*, encontrar padrões recorrentes no *mart.*

REFERÊNCIAS

ERL, T. **SOA Principles of Service Design**. Prentice Hall, 2008.

PAPAZOGLOU, M. **Web Services: Principles and Technology**. 1st Edition. Prentice Hall, 2007.

MAAMAR, Z.; WIVES L. K.; AL-KHATIB G.; SHENG Q. Z.; VITT A. R. D.; BENSLIMANE D**. From Communities of Web Services to Marts of Composite Web Services**, 2008.

MAAMAR, Z.; WIVES, L. K.; TATA, S.; SELLAMI, M. **Towards a Marriage between Web Services and Recommender Systems**.

ERL, T. **Service-Oriented Architecture: Concepts, Technology, and Design.** Prentice Hall, 2005.

MAO, Y.; LE, J. **Research on QoS-Based Web Service Composition**, 2009 International Conference on Advanced Computer Control, 2009

ZUROWSKA, K.; DETERS, R. **Load Management in Model-Aware Execution of Composite Web Services**, Proceedings of the 2009 ACM symposium on Applied Computing, 2009.

JURIC, M.B. **A Hands-on Introduction to BPEL**. Disponível em <http://www.oracle.com/technology/pub/articles/matjaz_bpel1.html> Acesso em: Jun 2011.

HORSTMANN, C. **Padrões e Projeto Orientados a Objetos**. 2nd ed. Porto Alegre: Bookman, 2007.

BUI, T.; GACHER, A. **Web Services for Negotiation and Bargaining in Electronic Markets: Design Requirements and Implementation Framework.** *In Proceedings of the 38th Hawaii International Conference on System Sciences (HICSS'2005)*, Big Island, Hawaii, USA, 2005.

BENATALLAH, B.; SHENG, Q. Z.; DUMAS, M. **The Self-Serv Environment for Web Sevices Composition.** IEEE Internet Computing, 7(1), Janeiro/Fevereiro 2003.

MEDJAHED, B.; BOUGUETTAYA, A. **A Dynamic Foundational Architecture for Semantic Web Services**. Distributed and Parallel Databases, Kluwer Academic Publishers, 17(2), Março 2005.

GAMMA, E. et al. **Padrões de Projeto: Soluções Reutilizáveis de Software Orientado a Objetos.** Porto Alegre: Bookman, 2000. 364p.

PREE, W. **Design Patterns for Object-Oriented Software Development.** New York-USA: Addison-Wesley Publishing Company 1995. 268p.

SILVA, R. P. **Suporte ao Desenvolvimento e Uso de Frameworks e Componentes.** Porto Alegre: 2000. 262p. Tese (Doutorado em Ciência da Computação) – Programa de Pós-Graduação em Computação, UFRGS, 2000. Disponível em: <http://www.inf.ufsc.br/~ricardo/download/tese.pdf>. Acesso em: Jun. 2011.

LUMERTZ, R. S. **Um Framework para Marts de Serviços Web Compostos**. 2010. 60 f. Projeto de Diplomação (Bacharelado em Ciência da Computação) – Instituto de Informática, UFRGS, Porto Alegre.

KICKZALES, G. et al. **Aspect-Oriented Programming.** *Published in proceedings of the European Conference on Object-Oriented Programming (ECOOP)*, Finland. June 1997.

SOARES, S.; BORBA, P. **AspectJ – Programação Orientada a Aspectos em Java.** Disponível em: <http://www.cin.ufpe.br/~scbs/artigos/AspectJ_SBLP2002.pdf>. Acesso em: Jun. 2011.

OASIS. UDDI Spec TC. Disponível em: <http://uddi.org/pubs/uddi_v3.htm> Acesso em: Mai. 2011.

SOA The Open Group. *Definition of SOA*. Disponível em <http://opengroup.org/projects/soa/doc.tpl?gdid=10632> Acesso em: Mai. 2011.

LUBLINSKY, B. *Defining SOA as an architectural style*. Disponível em: <http://www.ibm.com/developerworks/architecture/library/ar-soastyle/> Acesso em: Mai. 2011.

RAGHU, R. K. What is service-oriented architecture?. Disponível em <http://www.javaworld.com/javaworld/jw-06-2005/jw-0613-soa.html> Acesso em: Mai. 2011

WEBOPEDIA. Web Services. Disponível em <http://www.webopedia.com/TERM/W/Web_Services.html> Acesso em: Mai. 2011.

WORLD WIDE WEB CONSORTIUM – W3C. Web Services Architecture. Disponível em <http://www.w3.org/TR/ws-arch> Acesso em: Mai. 2011.

WORLD WIDE WEB CONSORTIUM – W3C. Extensible Markup Language. Disponível em <http://www.w3.org/XML> Acesso em: Mai. 2011.

WORLD WIDE WEB CONSORTIUM – W3C. Web Services Description Language. Disponível em <http://www.w3.org/TR/wsdl> Acesso em: Mai. 2011.

ANEXO A DICIONÁRIO DE DADOS DA BASE DE DADOS DO MONITOR

Tabela A.1: Dicionário de dados da tabela register.

Nome da coluna	*Tipo de Dado*	*Comentário*
id_register	serial	Chave primária
ip_server	varchar(100)	Identificador do serviço
ip_client	varchar(100)	Identificador do cliente
time	timestamp	Hora da operação
operation	varchar(100)	Operação realizada
status	varchar(100)	Status da operação