Pós-Graduação em Ciência da Computação

"COMPOSIÇÃO ADAPTATIVA DE *WEB SERVICES*"

Por

FERNANDO ANTONIO AIRES LINS

Dissertação de Mestrado

Universidade Federal de Pernambuco
posgraduacao@cin.ufpe.br
www.cin.ufpe.br/~posgraduacao

RECIFE, FEVEREIRO/2007

UNIVERSIDADE FEDERAL DE PERNAMBUCO
CENTRO DE INFORMÁTICA
PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

FERNANDO ANTONIO AIRES LINS

"COMPOSIÇÃO ADAPTATIVA DE *WEB SERVICES*"

ESTE TRABALHO FOI APRESENTADO À PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO DO CENTRO DE INFORMÁTICA DA UNIVERSIDADE FEDERAL DE PERNAMBUCO COMO REQUISITO PARCIAL PARA OBTENÇÃO DO GRAU DE MESTRE EM CIÊNCIA DA COMPUTAÇÃO.

ORIENTADOR: NELSON SOUTO ROSA

RECIFE, FEVEREIRO/2007

Lins, Fernando Antonio Aires
Composição adaptativa de Web services /Fernando Antonio Aires Lins. – Recife: O autor, 2007.
xiii, 106 p.: il., fig., tab.

Dissertação (mestrado) – Universidade Federal de Pernambuco. CIN. Ciência da Computação, 2007.

Inclui bibliografia, glossário e apêndices.

1.Sistemas Distribuídos. 2. Web Services. 3. Adaptabilidade. 4. Composição de serviços. I. Título.

004.36 CDD (22.ed.) MEI2007-005

Agradecimentos

- ✓ A Deus;

- ✓ A minha família, em especial a minha mãe Maria José, meu pai Nelson (*in memorium*), minha avó Aida, minha irmã Patrícia, meus tios Graça e Kleber, meus padrinhos Isac e Lia e aos meus primos Lucas e Paloma, por terem apoiado todas as minhas iniciativas profissionais e por estarem sempre presentes em minha vida, seja em horas boas ou tristes, me dando alegria e amor;

- ✓ A minha parceira e jornalista Camila Nascimento, pelos momentos de alegria, amor, descontração, pela ajuda na revisão ortográfica deste trabalho e principalmente pelo apoio incondicional nos momentos mais difíceis da realização do mesmo;

- ✓ Ao professor e amigo Nelson Rosa, pela sua orientação sempre rica de boas idéias e conselhos pertinentes;

- ✓ Ao também professor Francisco Heron, pela sua amizade e ensinamentos desde a época de graduação;

- ✓ A UFRPE, por ter apoiado esta minha iniciativa de capacitação, em especial o professor José Rodrigues Lemos e aos colegas de trabalho Fred e Ulisses;

- ✓ Aos amigos do mestrado, em especial Anderson Moreira, Helô Petry e José Carlos, pelos conhecimentos compartilhados e pelas várias farras que juntos fizemos;

- ✓ Aos amigos Carlos Vitor, Juliana Lima, Mariana Figueiredo, Daienne Amaral, Paulo Henrique Farias, Henrique Gama, Nívia Quental e Rodrigo Gomes, pelos vários momentos de descontração e por terem sempre me apoiado nas horas em que mais precisei.

Resumo

Web services têm desempenhado um importante papel no desenvolvimento de aplicações distribuídas. Em particular, a possibilidade de composição de serviços já implementados com o intuito de prover uma nova funcionalidade se constitui uma abordagem interessante para a construção de sistemas distribuídos e de processos de negócios (*business processes*), na medida em que diminui o tempo total de desenvolvimento e promove a reusabilidade de serviços já testados e utilizados.

A possibilidade de realizar a composição de *web services* de forma dinâmica, em tempo de execução, aparece nesse contexto com uma elevada importância. A composição dinâmica permite que mudanças sejam percebidas pelo sistema durante a sua execução, sem a necessidade de reinicialização de sistemas, o que poderia levar a indisponibilidade do serviço e a possível perda de clientes.

Algumas propostas têm surgido para composição dinâmica de *web services*, mas elas não adotam especificações já estáveis e notadamente difundidas, como WS-BPEL. Ao mesmo tempo, estas propostas usualmente implicam em modificações nas regras da especificação dos processos de negócios, dificultando ainda mais a implementação do sistema.

Este trabalho propõe uma nova abordagem para possibilitar a adaptabilidade na composição de *web services* através de modificações na semântica da primitiva de invocação de serviços, mantendo a sintaxe do principal padrão existente para este fim (WS-BPEL) inalterada. A partir das modificações propostas, programadores podem definir composições dinâmicas de *web services* sem alterar o código fonte da aplicação. Como implementar a adaptação passa a ser tarefa do ambiente e não do programador, o que torna esta tarefa mais transparente para o mesmo. Por fim, a adaptabilidade surgiria como mais uma possibilidade no projeto, mantendo todas as outras características da composição inalteradas.

Palavras-chave: *Web service*, adaptabilidade, composição de serviços, processos de negócios.

Abstract

Web services have played an important role in the development of distributed applications. In particular, the possibility of composing already implemented web services in order to provide a new functionality is an interesting approach for building distributed systems and business processes, in a way that it diminishes the total time of development and promotes the reusability of services already tested and used.

The possibility of dynamically composing web services, at runtime, is also apparent. Dynamic composition allows that changes can be perceived by the system during its execution, without requiring any reset of the systems, what belongs to harm the availability of the system and the possible loss of clients.

Current approaches address this point, but they do not adopt established specifications, like WS-BPEL, as they usually propose modifications in the rules of business process specifications by imposing more difficulties to implement the system.

This work proposes a new approach for improving web service adaptability through a semantics modification in the invocation primitive, whilst it keeps the syntax of WS-BPEL unaffected. Hence, programmers may define dynamic web service compositions without changing the source code. The task of how to implement the adaptation belongs now to the environment, exempting the programmer of this task, becoming the task most easy for the programmer. Finally, the adaptability would appear as a new possibility in the project, keeping all the others characteristics of the composition unchanged.

Keywords: Web service, adaptability, service composition, business process.

Índice

Lista de Figuras

Lista de Tabelas

Glossário

SOA	*Service-Oriented Architecture*
SOAP	*Simple Object Access Protocol*
WSDL	*Web Service Description Language*
SMTP	*Simple Mail Transfer Protocol*
UDDI	*Universal Discovery, Description and Integration*
WSRF	*Web Services Resource Framework*
QoS	*Quality of Service*
POA	*Programação Orientada a Aspectos*

CAPÍTULO

1

Introdução

"Não há vento favorável para quem não sabe aonde vai".

Guillaume D'Orange.

Este capítulo apresenta uma introdução do trabalho no contexto geral da área que ele se aplica, em especial na composição de *web services* e adaptabilidade. Inicialmente, os antecedentes deste trabalho são apresentados e é definida a área de conhecimento na qual ele está inserido. Em seguida, as motivações iniciais são apresentadas, destacando o contexto nos quais as mesmas se encaixam. Os principais aspectos envolvidos na composição adaptativa de *web services* são então introduzidos, relacionando-os brevemente com trabalhos já existentes. Posteriormente, os principais problemas em aberto nesta área são evidenciados. Finalmente, são apresentadas a proposta do trabalho e a estruturação geral deste documento.

1.1 Área do Conhecimento

Este trabalho tem seu foco centrado em SOA (*service-oriented architecture* – arquiteturas orientadas a serviço) [Tsai 2006]. Em verdade, o conceito das arquiteturas orientadas a serviço já existe há muito tempo [van der Aalst 2003], mas agora ele reapareceu com maior força devido à necessidade de integração das mais diversas tecnologias que temos atualmente. O SOA pode ser considerada uma metodologia [Plummer 2004] que visa maximizar a reutilização de serviços pré-existentes e a integração com novos serviços, com o intuito de aumentar a eficiência e a produtividade das empresas. A proliferação da utilização dos *web services* é uma prova da larga aceitação e do futuro promissor desta tecnologia.

1.2 Composição Adaptativa de *Web Services*: Caracterização do Problema

A composição de *web services* tem surgido como uma importante estratégia para possibilitar a colaboração entre organizações [Benatallah 2002]. Mais especificamente, a composição permite que aplicações mais complexas sejam construídas através da combinação de serviços mais básicos. Esta estratégia acelera o desenvolvimento de sistemas distribuídos, favorece a reusabilidade de serviços e facilita o desenvolvimento de sistemas mais complexos. Em uma composição, é especificada a ordem de execução das operações a partir de uma coleção de *web services*, os dados compartilhados entre estes *web services*, que parceiros estão envolvidos e como eles estão envolvidos no processo de negócio [OASIS 2007].

Neste contexto, novos *web services* estão em constante desenvolvimento. Em um processo de negócio, pode ser necessária a atualização de um *web service* ou até mesmo a sua troca por um outro com a mesma interface, porém com diferente implementação. Por exemplo, uma implementação mais segura pode ser preferida em detrimento de uma detentora de menor tempo de execução. Então, é usualmente necessário identificar atualizações/novidades em *web services* e implementar as mudanças assim que possível, sem a necessidade de parada de todo o sistema. Em particular, a parada de um sistema distribuído pode causar a perda de clientes que estiverem executando no momento da parada, algumas operações podem requerer um tempo elevado para serem completadas, dentre outras. Uma composição adaptativa deve ser capaz de executar essas modificações sem a necessidade de reiniciar o sistema.

1.3 Composição Adaptativa de *Web Services*: Estado da Arte

Prover adaptabilidade em uma composição de *web services* é uma tarefa complexa. Através do uso de WS-BPEL [OASIS 2007], um padrão largamente utilizado no desenvolvimento de processos de negócio, informações sobre os serviços participantes devem ser obtidas até o momento em que a composição estiver sendo disponibilizada no servidor Web. Essa obtenção é feita de forma estática, no próprio código-fonte da aplicação. Como prover este tipo de informação e como proceder a modificações em tempo de execução permanece uma discussão aberta no contexto em que o trabalho se apresenta.

Charfi [Charfi 2004] propõe o uso de orientação a aspectos através do projeto AO4BPEL, onde a composição pode sofrer alterações dinâmicas, além de ser conferida uma maior modularidade ao projeto. Em particular, ele assume que, em geral, regras de negócio são exemplos típicos de separação de interesses em composições de *Web services* e considera adaptabilidade um interesse específico (aspecto), podendo inclusive ser ativada e desativada em tempo de execução. Embora tenha a modularidade e a possibilidade de ativação/desativação da adaptabilidade como vantagens potenciais, o seu projeto ainda carece de uma maior maturidade e sua implementação é considerada de elevada complexidade, pois a incorporação de aspectos nos ambientes atuais que suportam composição de *web services* ainda não se encontra em um estágio avançado, e a implementação de diversos módulos associados são requisitados, como será visto posteriormente neste trabalho. Além disso, a forma de desenvolvimento do projeto seria alterada, utilizando-se

de orientação a aspectos, apenas pela necessidade de se conferir adaptabilidade. Isto obriga os programadores a possuírem um largo conhecimento no paradigma orientado a aspectos, o que pode se constituir uma barreira para a sua utilização e aumentar o tempo total do projeto. O ideal seria que o mínimo de esforço e alterações no código fonte da aplicação se fizessem necessários para tornar a adaptação possível.

Karastoyanova [Karastoyanova 2005] propõe uma alteração sintática na especificação padrão do WS-BPEL [OASIS 2007]. Mais especificamente, esta proposta adiciona uma nova primitiva na especificação padrão. Através desta inclusão, o trabalho mostra que a adaptabilidade pode ser obtida por meio da utilização de um mecanismo denominado "*find and bind*". Embora essa abordagem torne possível a mudança de *web services* em tempo de execução, o programador deve modificar o código fonte de sua aplicação tendo em vista a utilização desta primitiva. Além da necessidade de utilizar uma *engine* (software que provê o ambiente necessário à execução da composição) que suporte esta nova primitiva, o projetista deve estar ciente desta nova possibilidade e se adequar ao seu uso. Adicionalmente, esta proposição modifica uma especificação já estável e amplamente difundida, o que pode dificultar a sua proliferação e utilização.

Erradi [Erradi 2005] propõe um *framework* para o suporte de composição adaptativa de *web services* baseado em políticas. Esta proposta apresenta uma nova forma de se realizar adaptações em composição de *web services*, utilizando políticas para determinar como uma composição pode ser modificada. Contudo, assim como [Charfi 2004], os programadores devem conhecer e utilizar o paradigma orientado a aspectos, o que acaba aumentando o nível de dificuldade e o desenvolvimento da composição.

McKinley e seu grupo propuseram uma taxonomia de composição com o intuito de classificar os vários métodos que dão suporte à adaptabilidade composicional [McKinley 2004]. Adicionalmente a esta proposição, eles apresentam um estudo detalhado sobre o estado da arte da adaptabilidade. Com vários conceitos importantes e apresentação de mecanismos, o mesmo acabou por não tratar uma categoria específica, como, por exemplo, *web services*.

1.4 Proposta, Objetivos e Contribuições

Este trabalho apresenta uma abordagem alternativa para composição adaptativa de *web services* em relação a outras já existentes introduzidas na seção anterior, com foco específico no nível de troca de parceiros e de serviços em tempo de execução. Esta idéia se baseia fundamentalmente na alteração semântica de uma das primitivas do WS-BPEL, o *invoke*. Esta primitiva é responsável pela invocação de *web services* dentro da composição. No contexto atual, o *invoke* executa de forma estática: são passados o nome do parceiro (entidade que disponibiliza o *web service*) e os parâmetros para a execução do serviço. Se acontecer qualquer problema com este parceiro, a aplicação interrompe seu funcionamento, necessitando de paradas para possíveis alterações. Através da modificação proposta, o *invoke* passa a ter um comportamento dinâmico, podendo perceber se houve alguma mudança no ambiente e agir para fazer as modificações necessárias.

Para possibilitar esta alteração na semântica do *invoke*, o presente trabalho propõe uma extensão para uma das *engines* mais difundidas e de código aberto que dão suporte ao WS-BPEL (mais especificamente, ActiveBPEL [ActiveBPEL 2006]). Com a incorporação desta extensão adaptável a *engine* original, a mesma irá então adquirir um comportamento adaptável. Esta modificação em nada altera a especificação em BPEL das aplicações; o que será alterado é a execução da *engine*, que apresentará um comportamento adaptável, totalmente transparente aos programadores. A Figura 1.1 apresenta um exemplo de composição em uma *engine* adaptável.

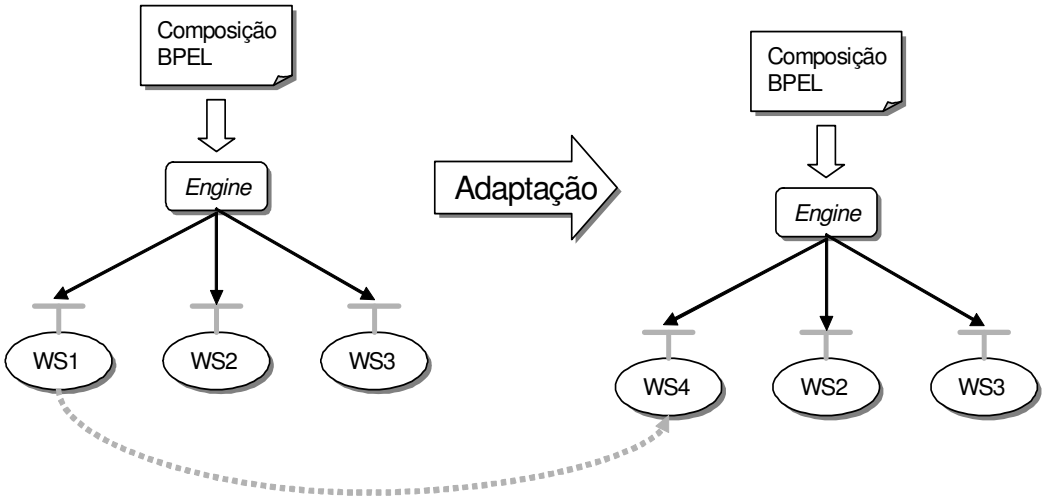

Figura 1.1: Exemplo de execução de uma *engine* adaptável

O exemplo é constituído de quatro *web services* distintos: WS1, WS2, WS3 e WS4. Inicialmente, pode-se supor que a composição é constituída por WS1, WS2 e WS3. Em tempo de execução, um novo *web service* com características funcionais idênticas a WS1 pode ser disponibilizado (WS4), porém com algum aspecto não-funcional melhorado, como segurança ou tempo de execução. Desta forma, assim que a *engine* enxerga este novo serviço no ambiente e percebe que ele é funcionalmente idêntico a WS1, ela pode verificar se ele tem mais "qualidade" do que WS1 (por exemplo, baseado em algum critério não-funcional). Depois que a *engine* verificar que WS4 é, por exemplo, mais seguro do que WS1, ela passa a redirecionar todas as chamadas que iriam para WS1 para WS4.

É importante ressaltar que a adaptação proposta será focada na possibilidade de escolha dinâmica dos *web services*, e não nas regras da composição em si. Na prática, isto significa dizer que não é objeto de estudo deste trabalho a mudança das regras do negócio em si, e sim a possibilidade de troca de parceiros de forma dinâmica. Com a proposição desta abordagem objetiva-se:

- Minimizar o esforço de programação necessário para a disponibilização de um mecanismo adaptável dentro do projeto;

- Preservar padrões pré-estabelecidos, *i.e.*, não propor alterações sintáticas nos padrões;
- Propor um mecanismo adaptável com o maior nível de transparência possível (sem exigir um número excessivamente elevado de intervenções por parte do gerente);
- Preservar as outras características originais da *engine* ActiveBPEL (já amplamente testados e difundidos).

Como foi visto na seção anterior, vários trabalhos propões soluções para a questão da adaptabilidade na composição de *web services*. A principal contribuição deste trabalho é a de fornecer uma alternativa que respeite os padrões já estabelecidos e amplamente difundidos (como o WS-BPEL), que torne a utilização do mecanismo transparente para o projetista, que não exija alterações no código fonte da aplicação e que não exija outros conhecimentos específicos (como orientação a aspectos) aos envolvidos no projeto.

1.5 Estrutura da Dissertação

Além deste capítulo de introdução, o presente trabalho inclui ainda mais cinco capítulos, como descrito a seguir:

Capítulo 2: Neste capítulo serão introduzidos todos os conceitos básicos associados a este trabalho. Os aspectos relacionados à arquitetura orientada a serviços (SOA), aos *web services*, aos processos de negócio (*business processes*) e a composição de serviços serão detalhados. Adicionalmente, os principais aspectos referentes à adaptabilidade de sistemas serão apresentados, que irão nortear a descrição da abordagem proposta neste trabalho.

Capítulo 3: Este capítulo apresenta a abordagem proposta nessa dissertação para a composição adaptativa de *web services*. Inicialmente, os princípios básicos desta metodologia são apresentados, vindo logo em seguida a implementação da extensão A-ActiveBPEL, responsável por prover o mecanismo adaptável. Por fim, é descrito o *Web Service Repository*, estrutura responsável por armazenar informações sobre os *web services* disponíveis no ambiente.

Capítulo 4: Os estudos de caso deste trabalho são apresentados neste capítulo. As aplicações escolhidas são o Somador Distribuído e o Sistema de Aprovação de Crédito. Depois de apresentar uma visão geral sobre as ferramentas utilizadas, este capítulo apresenta aspectos sobre a configuração, a execução e, principalmente, sobre os resultados obtidos em cada um dos estudos de caso.

Capítulo 5: Neste ponto, uma discussão sobre os trabalhos relacionados a este trabalho é apresentada. Duas correntes distintas de trabalho são identificadas, uma propondo alteração sintática no padrão WS-BPEL e a outra sugerindo a utilização de orientação a aspectos para a composição adaptativa de *web services*. Esses trabalhos são apresentados, e uma avaliação crítica sobre os mesmos é feita. Por fim, considerações comparativas sobre esta dissertação e esss trabalhos relacionados são realizadas.

Capítulo 6: Finalmente, este capítulo apresenta as conclusões e os trabalhos futuros associados a esta dissertação. São explicitadas também as contribuições da mesma e as limitações da abordagem apresentada, seguidas de propostas de aperfeiçoamento.

CAPÍTULO

2

Conceitos Básicos

"O que sabemos é uma gota, o que ignoramos é um oceano".

Isaac Newton.

Neste capítulo são introduzidos os conceitos básicos e as tecnologias adotadas neste trabalho. Inicialmente, as principais idéias das arquiteturas orientadas a serviço são expostas. Em seguida, um dos maiores exemplos destas arquiteturas orientadas a serviço, *web service*, é alvo de um estudo mais elaborado. A composição de serviços, em especial de *web services*, também é apresentado neste capítulo. Os processos de negócio (*business processes*) são apresentados, e sua relação com a composição de *web services* é analisada. Por fim, os conceitos básicos de adaptabilidade são introduzidos.

2.1 Arquitetura Orientadas a Serviço (SOA)

Um número crescente de organizações tem implementando sistemas de informação em seus diversos departamentos. Além disso, os mesmos estão cada vez mais ultrapassando os limites da própria empresa. O grande desafio que surge neste contexto é encontrar uma solução que seja extensível, flexível e que seja compatível com os sistemas já existentes. Realizar a substituição de sistemas legados tendo em vista a incorporação de novas tecnologias e requisitos pode trazer riscos ao funcionamento de aplicação, além de ter, em geral, um alto custo associado.

Neste cenário, um antigo conceito [van der Aalst 2003] voltou a ter destaque, o das arquiteturas orientadas a serviços. Com o advento de programas independentes de plataforma (interoperáveis), a idéia de se ter serviços rodando

transparentemente em diversas plataformas passou a ter um atrativo especial, em particular quando se trata de integração de sistemas. O SOA (*Service Oriented Architecture* [Pallos 2001] [Papazoglou 2003] [Tsai 2006]) é uma tecnologia recentemente difundida baseada na orientação a serviços, e tende a revolucionar as formas atuais de desenvolvimento de sistemas. O serviço é a unidade atômica do SOA e os sistemas operam através de uma coleção de serviços independentes, em que os mesmos podem interagir com vários outros serviços para executar determinadas tarefas. Um dos principais pontos desta tecnologia é a reutilização de serviços previamente implementados e testados, reduzindo o tempo total de projeto, os custos associados e diminuindo os riscos em desenvolvimento e manutenção.

Existem diversos exemplos bem sucedidos de sistemas orientados a serviços presentes no mundo atual, tais como, os sistemas de telecomunicações, os serviços financeiros e o sistema elétrico.

Contudo, se este não é um conceito novo, o que garantiria que realmente este é o momento certo para ele voltar à tona? Primeiramente, a larga proliferação dos *web services* (principal exemplo atual da utilização do SOA) possibilitou a volta da discussão acerca de serviços. Segundo, a disseminação do uso da Internet (portais, serviços na Internet, etc.) facilita a idéia de usar serviços disponibilizados na grande rede (em especial, por parceiros/provedores confiáveis). Finalmente, existem plataformas integradas de desenvolvimento que facilitam a tarefa de implementar aplicações orientadas a serviços, tornando acessível este tipo de tecnologia para uma grande fatia do mercado da computação.

A construção de sistemas baseados no SOA difere significativamente dos métodos tradicionais de desenvolvimento, como por exemplo os baseados no paradigma funcional e no orientado a objetos. A utilização de serviços pode parecer simples a primeira vista, contudo existem algumas tarefas adicionais, como a necessidade de se analisar a funcionalidade da aplicação (especificamente, verificando se todos os serviços juntos estão fazendo realmente o que se deseja), montar serviços compostos através de serviços primitivos compatíveis entre si (serviços que têm a mesma funcionalidade podem apresentar interfaces distintas), modelar essa composição para que ela reproduza as regras de negócio desejadas e gerenciar mudanças feitas em serviços disponibilizados por terceiros (provedores externos).

De forma resumida, a topologia básica de um sistema orientado a serviços é composta por três elementos: clientes dos serviços (ou consumidores), provedores dos serviços (ou servidores) e agenciadores (ou serviços de registro/localização). O cliente acessa o provedor através de uma interface padrão. Mesmo que o serviço sofra alterações em sua implementação, o cliente não irá ser afetado, pelo fato de o mesmo apenas enxergar a interface do serviço. A Figura 2.1 apresenta resumidamente a topologia básica de um sistema orientado a serviços.

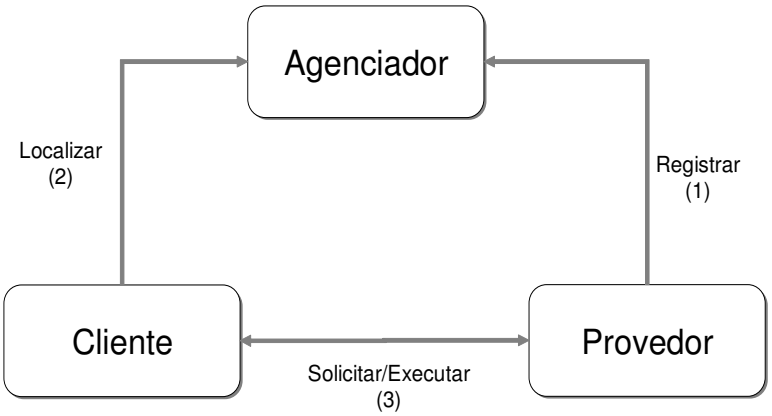

Figura 2.1: Topologia básica de um sistema orientado a serviços

Na topologia apresentada, não apenas os três principais atores do SOA (cliente, provedor e agenciador) são explicitados, mas também os seus relacionamentos. Inicialmente, o provedor do serviço deve registrar o mesmo em algum agenciador (1), elemento responsável por armazenar o nome dos serviços e suas respectivas localizações. Após este registro, o cliente entra em contato com o agenciador para pedir informações, mais especificamente, sobre o endereço do provedor do serviço que deseja utilizar (2). De posse desta informação, o cliente acessa o servidor com o objetivo de executar o serviço desejado (3). Vale ressaltar a importância da configuração básica apresentada na Figura 2.1, aparecendo com destaque em vários projetos de sistemas distribuídos.

Como objetivos gerais do proposição do SOA, podem-se destacar:

- ✓ Independência tecnológica (em especial, interoperabilidade entre cliente/servidor);
- ✓ Fraco acoplamento (especificação/implementação do cliente não deve depender do servidor);
- ✓ Interfaces auto-descritivas com grossa granuralidade;
- ✓ Independência do protocolo de comunicação (a escolha do protocolo não deve influenciar na funcionalidade do serviço);
- ✓ Transparência na invocação de serviços (eles devem ser invocados de forma similar tanto remotamente como localmente);
- ✓ Suporte para comunicação síncrona e assíncrona;
- ✓ Composição de serviços em processos de negócios (importância destacada em EAI – *Enterprise Application Integration* e B2B – *Business-to-Business Integration*).

Uma questão fundamental que surge no contexto de desenvolvimento de sistemas baseados no SOA é a de identificar serviços. Tecnicamente, é possível transformar qualquer tipo de função em um serviço. Contudo, se isto for feito, o número total de serviços será muito elevado.

Para o tratamento desta questão existe uma recomendação que vem sendo utilizada pela comunidade e que reduz o número total de serviços em um sistema [Gupta 2007]. Nesta recomendação, sugere-se que os serviços representem conceitos de negócios tangíveis. Por exemplo, uma função do tipo *getFaturaTotal()*, por ser um conceito de negócio tangível, seria uma forte candidata a ser um serviço. Contudo, não se recomendaria transformar uma função do tipo *convertStringtoInt()* em um serviço, pela razão da mesma não representar um conceito de negócio tangível.

Por fim, vale ressaltar que a orientação a serviços já vem sendo vista na comunidade como um novo paradigma computacional, e definitivamente não é um novo nome para orientação a componentes e muito menos se define como um padrão de projeto. Em seu conjunto, SOA apresenta uma coleção de conceitos com os quais podem-se construir novos tipos de sistemas. Em uma visão bastante otimista, estima-se que o desenvolvimento orientado a serviços mudará a forma de construir e vender sistemas em 80% das empresas [Plummer 2004], o que mostra a relevância desta tecnologia no atual estado da arte da Ciência da Computação. Contudo, é importante deixar claro que nem sempre a utilização do SOA é recomendada. Por exemplo, no caso de sistemas mais simples, nos quais os custos de implementação da nova tecnologia poderiam não ser justificados pela melhoria da qualidade do *software*. Uma boa análise do impacto da utilização desta tecnologia pode diminuir situações embaraçosas e custos desnecessários para as empresas.

2.2 *Web Services*

Há alguns anos, o mundo da computação distribuída foi surpreendido com uma proposta inovadora, baseada em técnicas padronizadas para descrever interfaces para componentes de software, métodos para acessar tais componentes por meio de protocolos interoperáveis e métodos de descoberta que permitem a identificação de importantes provedores de serviço. Além disto, a proposta incluía ainda a independência de linguagens de programação e/ou sistemas operacionais. Segundo estes preceitos, os *web services* foram propostos.

A Figura 2.2 apresenta um exemplo simples de w*eb service* que implementa um serviço de busca na Internet. Este serviço foi publicado em algum servidor na Internet e o cliente pode localizá-lo através de métodos de descoberta, *e.g.*, um repositório UDDI (seção 2.2.1.4).

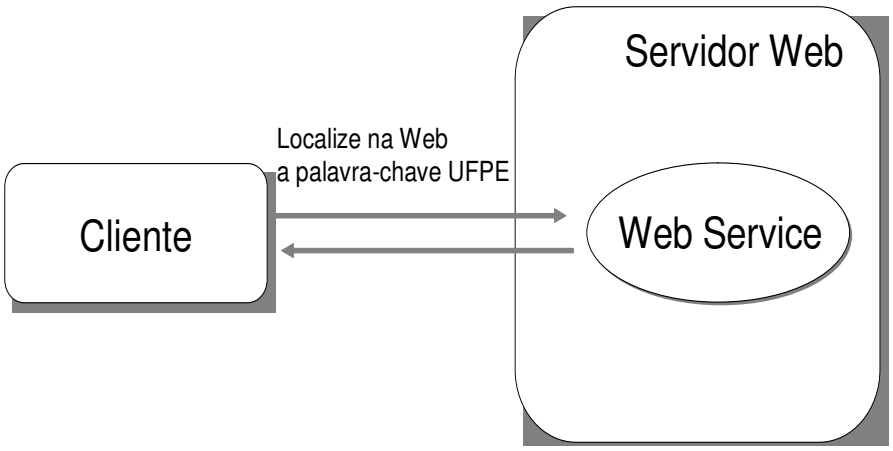

Figura 2.2: Exemplo de *web service*

O cliente, que deseja obter informações sobre a palavra-chave UFPE, envia uma solicitação ao serviço, informando a palavra-chave de interesse. O *web service*, por sua vez, retorna uma resposta ao programa cliente contendo informações sobre a palavra-chave informada. É válido ressaltar que programas, e não humanos, acessam os *web services*.

Como mencionado anteriormente, *web services* nada mais são do que uma nova tecnologia de computação distribuída. Sendo assim, o que a diferencia de outras a ponto de justificar sua proposição e utilização são os seguintes pontos:

1) Independência de plataforma de execução e de linguagens de programação. Esta peculiaridade se justifica pelo uso de XML, que é uma linguagem padrão para troca de dados.

2) Utilização de HTTP para transmissão de mensagens. A grande vantagem da utilização deste padrão reside na questão da acessibilidade. Por questões de segurança, muitos administradores de sistemas bloqueiam o acesso às suas redes internas através de *firewalls*, os quais acabam também inviabilizando a utilização de tecnologias que não utilizam a porta 80 do HTTP.

Por outro lado, a portabilidade dos *web services* tem um preço a ser pago. Obviamente, transmitir dados no formato XML ocasiona um *overhead* muito maior, pois como uma meta-linguagem, o volume do tráfego aumenta, tendo em vista que não apenas os dados estarão trafegando, como também as definições desses dados.

A Figura 2.2 mostra uma visão bastante simplificada da invocação de um *web service* por um cliente. Na prática, existe um conjunto de passos que devem ser seguidos, como mostrado na Figura 2.3. Inicialmente, o cliente deve saber onde se encontra o serviço desejado. Para isso, ele acessa um serviço que possui o nome e localização de vários serviços. Em *web services*, utiliza-se o UDDI (*Universal Description, Discovery and Integration*), especialmente desenvolvido para organização e registro de *web services*.

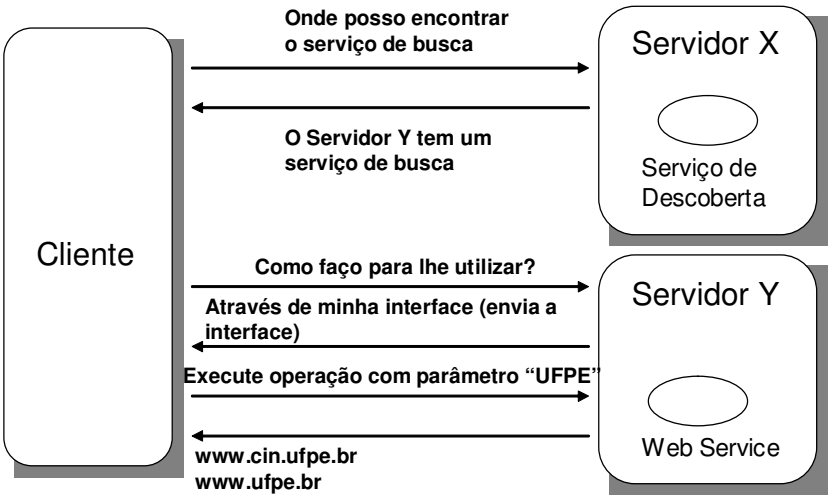

Figura 2.3: Invocação de um *web service* por um cliente

Após descobrir a localização de um serviço, o cliente se depara com outra questão: como acessar os serviços deste *web service*. Para isso, o cliente acessa a interface do mesmo, a qual é descrita pela linguagem WSDL (*Web Service Description Language*) [Christensen 2001], linguagem padrão para a descrição de interface de serviços. Por ser um padrão baseado em XML, o uso de WSDL faz com que o acesso ao *web service* seja independente de linguagens de programação e de ambientes de desenvolvimento.

Após analisar a interface do *web service* (especificada em WSDL), o cliente invoca o serviço desejado. Em geral, a invocação de um serviço envolve passagem de mensagens entre o cliente e o serviço. Em *web services*, essa passagem de mensagens é realizada através do protocolo SOAP (*Simple Object Access Protocol*), que especifica como fazer requisições para o servidor e como o mesmo deve formatar suas respostas. Em teoria, outras linguagens de invocação de serviços podem ser usadas. Contudo, SOAP é amplamente utilizada em *web services*, em especial por utilizar como protocolo de transporte o HTTP e o SMTP.

2.2.1 Protocolos e tecnologias associadas

Como mencionado antes, os *web services* são baseados em um conjunto de padrões amplamente utilizados e aceitos. Esta larga aceitação possibilita que clientes e serviços se comuniquem através de uma ampla variedade de plataformas (como diversos tipos de servidores Web e sistemas operacionais) e linguagens (como Java e .NET). Esta seção apresenta os protocolos e as tecnologias que constituem os padrões de *web services*.

2.2.1.1 XML

A linguagem XML (*Extensible Markup Language*) [W3C 2006] se tornou o padrão para a descrição de dados. Como o próprio nome diz, XML é uma linguagem de marcação (assim como HTML) que objetiva descrever o conteúdo de um documento. Para marcar o documento, *tags* são utilizadas. Uma *tag* no XML identifica a informação em um documento, e identifica também a estrutura desta informação. Adicionalmente, XML é uma linguagem que serve

para transportar dados, e não para processá-los ou formatá-los (caso específico da linguagem HTML).

A Figura 2.4 apresenta um exemplo de especificação XML. O exemplo apresentado, bastante tradicional, é o de uma locadora de filmes.

```
(1)<?xml version="1.0" encoding="ISO-8859-1"?>
(2) <locadora>
(3) <dvd>
(4)   <titulo> O Conde de Monte Cristo </titulo>
(5)   <autor> Alexandre Dumas </autor>
(6)   <genero> Suspense </genero>
(7)   <fornecedor> Europa Filmes </fornecedor>
(8)   <quantidade> 10 </quantidade>
(9)   <duracao> 120 </duracao>
(10)   <preco moeda="real"> 10.00 </preco>
(11) </dvd>
(12) </locadora>
```

Figura 2.4: Exemplo de uma especificação XML

A linha (1) fornece as informações básicas do arquivo XML, provendo a sua versão e o tipo de codificação que está sendo utilizado. Nas linhas seguintes (2-12), é especificado o corpo do documento, que contém uma instância do tipo DVD e a especificação de suas características.

Um documento XML é especificado basicamente através de elementos e atributos e é processado por um *parser* (*parser* XML). Na Figura 2.4, são apresentados diversos elementos, como `<titulo>` e `<autor>`. Cada elemento tem um significado e funções especiais que navegadores diferentes entendem. No caso do atributo, pode-se afirmar que ele tem a função de ajudar a descrever um elemento, *e.g.*, o elemento `<preco>` tem um atributo que especifica que a moeda utilizada é o real.

Os elementos de um documento XML podem ser classificados, segundo o seu conteúdo, em quatro categorias básicas:

- ***Empty content*:** elementos pertencentes a esta categoria não contêm nada e normalmente não têm uma *tag* final. Exemplo: *<autor nome="Alexandre Dumas">*;
- ***Simple content***: nesta categoria, elementos possuem apenas texto em seu conteúdo. Exemplo: *<fornecedor> Europa Filmes </fornecedor>*;
- ***Element content***: possui apenas outros elementos em seu conteúdo. Exemplo: *<dvd> <quantidade> 10 </quantidade> </dvd>*;

- ***Mixed content***: possui elementos e textos em seu conteúdo. Exemplo: <dvd> O Conde de Monte Cristo <autor> Alexandre Dumas</autor> </dvd>.

Para a etapa da validação do documento XML, é necessário saber quais *tags* são permitidas em um determinado contexto e as suas relações umas com as outras. Esses conjuntos de *tags* permitidas são descritos através de um esquema XML (XML *Schema*). O documento XML é comparado com o esquema XML para garantir que as *tags* definidas no mesmo estejam de acordo com as *tags* definidas no esquema. Essa validação é feita por um *parser* XML. O esquema XML é a nova versão, atualizada e aperfeiçoada, das antigas DTDs (*Document Type Definitions*).

Finalmente, aparece o conceito de *namespace* (ou espaço de nomes). Um mesmo nome de uma *tag* poderia ser definido em vários contextos, causando um grande conflito de nomes. Além disso, como um documento XML pode conter outros documentos XML, se faz necessário garantir que nenhuma das *tags* do documento incluído seja idêntica a um dos documentos principais. Esse mecanismo é o *namespace*, lugar onde todos os nomes são únicos. Por exemplo, não podemos ter dois elementos chamados `<autor>` em um mesmo *namespace*. O atributo `xmlns` informa ao *namespace* cujos nomes dos elementos devem ser únicos. Por exemplo, suponha que na Figura 2.4 o elemento `<dvd>` fosse escrito como `<dvd xmlns:http://www.cin.ufpe.br/~faal2>`. Neste caso, estariam sendo utilizados todos os elementos pertencentes ao *namespace* `www.cin.ufpe.br/~faal2`.

2.2.1.2 SOAP

O SOAP (*Simple Object Access Protocol*) [W3C 2003] foi criado inicialmente para possibilitar a invocação remota de métodos através da Internet. O SOAP surgiu numa época em que as poucas opções para a invocação remota de métodos eram o DCOM [Grimes 97], o CORBA [OMG 2002] e o RMI [Sun 2003]. Contudo, para a invocação remota na Internet, essas opções se mostraram inadequadas por algumas razões: uma delas é que, em segurança da informação, é uma prática habitual configurar o *firewall* para deixar acessíveis para usuários externos apenas serviços essenciais como o HTTP (porta 80). O problema de segurança surge na medida em que esses outros padrões (DCOM, RMI e CORBA) requisitam uma porta específica (diferente da porta 80) para ser utilizada, e com o agravante de objetivar a realização de invocações remotas de métodos dentro do servidor. Este tipo de conduta praticamente inviabilizou a utilização do DCOM, de CORBA e de RMI para a invocação remota de métodos na Internet, e acabou favorecendo a proliferação do SOAP.

A grande utilização do SOAP é na computação distribuída baseada no estilo RPC (*Remote Procedure Call*), contudo ele pode ser empregado também para outros fins. Qualquer protocolo de transporte pode ser utilizado para carregar mensagens SOAP, embora apenas o HTTP e o SMTP sejam descrito na especificação do padrão.

A unidade básica de transmissão SOAP é constituída pelo seu envelope, utilizado para descrever o conteúdo da mensagem e fornecer informações sobre como processar esse conteúdo. No arquivo XML, as *tags* que abrem e fecham o

código do envelope delimitam a mensagem SOAP. O envelope pode ser dividido em duas partes distintas, o corpo e o cabeçalho. Dentro do cabeçalho encontramos informações de controle, enquanto que no corpo encontramos o corpo da mensagem.

A simplicidade do SOAP carrega ainda mais um benefício associado, a representação dos tipos de dados. De forma resumida, o SOAP permite que dados sejam passados sem informações sobre os tipos de dados. Nesse caso, o tipo padrão *string* é utilizado. Caso se faça necessário outro tipo de dado, ele pode ser incluído simplesmente adicionando um atributo que indique o tipo do dado. Desta forma, o SOAP tira proveito do único formato que a maioria dos computadores, de diferentes marcas e arquiteturas, pode compartilhar facilmente uns com os outros: texto. O HTTP, assim como o SMTP, pode facilmente apanhar um arquivo de texto de um computador e transferi-lo para outro sem perda de dados. O SOAP acabou repassando a responsabilidade da conversão de dados ao programador do *web* service, que vai transformar os dados descritos na interface do serviço nos dados locais da sua plataforma servidora, e ao programador do cliente, que vai converter os dados locais de sua aplicação original para os tipos requisitados pelos *web services*. Maiores detalhes sobre aspectos sintáticos do protocolo SOAP podem ser obtidos em [W3C 2003].

2.2.1.3 WSDL

Uma das grandes vantagens dos *web services* sobre métodos mais tradicionais reside no fato de que eles podem ser formalmente descritos em um documento definido por um padrão denominado WSDL (*Web Services Description Language*) [Christensen 2001]. Este padrão, baseado em XML, é utilizado para a especificação das interfaces dos *web services*, servindo como base para informar a todos os possíveis clientes sobre, por exemplo, o que o serviço pode fazer, como invocá-lo, qual o formato das respostas e outras informações requeridas para a sua correta invocação. WSDL permite que os programadores do serviço forneçam informações cruciais acerca do mesmo, possibilitando que clientes o utilizem corretamente.

Em termos práticos, WSDL define um esquema XML[1] para descrição de *web services*. Para um cliente invocar um serviço, ele deve antes acessar a interface WSDL do serviço e verificar a forma com que o mesmo deve ser acessado. Para fazer esta chamada, o cliente precisa localizar onde a interface WSDL do serviço está disponibilizada. Uma forma de descobrir é através do registro do serviço (o implementador de um serviço pode registrá-lo em algum "agenciador", no caso específico dos *web services* no UDDI visto na Seção 2.2.1.4), o qual pode ter informações acerca da localização (URL) da interface. Outra forma possível é o próprio implementador do serviço passar diretamente (ou deixar exposto em algum lugar acessível para o programador do cliente) o endereço da interface WSDL.

É importante ressaltar que a interface WSDL descreve apenas a sintaxe que deve ser utilizada para o serviço ser acessado. Não existe uma preocupação direta em explicitar aspectos semânticos do serviço. Ela se limita expressamente a dizer o que o serviço faz apenas em um nível mais estrutural,

[1] Disponível em http://schemas.xmlsoap.org/wsdl/)

sem entrar em detalhes sobre o que o serviço faz com os dados que foram enviados. Existe atualmente um grande esforço para possibilitar a especificação da semântica de *web services*, especialmente com o advento da *Semantic Web* [McIlraith 2001]. Neste caso, anotações semânticas podem ser associadas aos serviços, dando pelo menos uma idéia geral sobre como os mesmos executam. Embora a semântica tenha sido omitida da proposta inicial do WSDL, ela ainda pode ser utilizada graças à natureza extensível da linguagem de descrição de *web services*.

WSDL tem sido utilizada em duas situações distintas:

- **Descrição de serviços para clientes**: situação mais usual. O documento WSDL descreve um serviço para os seus clientes. O objetivo principal é possibilitar que um cliente utilize o serviço de forma correta e eficaz. Após o desenvolvimento, o programador do serviço produz a interface WSDL do *web service* e a disponibiliza; e
- **Descrição de um modelo para programadores**: neste caso, o documento WSDL serve como base para a implementação do serviço. Ele pode servir como um contrato entre os usuários e os programadores do serviço, especificando previamente a forma com a qual o mesmo deve ser disponibilizado. Isto obriga os programadores a tomarem como base na implementação a interface previamente acordada. Com a utilização desta metodologia, o documento WSDL funciona como uma "especificação de requisitos", amarrando a implementação do *web service* a interface anteriormente acordada. Contudo, essa idéia pode levar a um maior custo de implementação e tempo de projeto (nem sempre se considera uma tarefa trivial moldar um programa a uma interface já definida). Neste caso, a interface WSDL pode até ser considerada um documento de projeto.

O padrão WSDL possui algumas particularidades importantes para o projeto de um *web service*:

- **Extensibilidade**: é fato sabido que devemos diminuir ao máximo a especificação de um padrão, colocando apenas os aspectos mais importantes. WSDL seguiu este preceito, porém adotou a noção de extensibilidade. Caso seja desejado acrescentar mais alguma informação (ex.: aspectos de qualidade de serviço), o programador pode propor a sintaxe adequada e inseri-la no documento WSDL;
- **Interoperabilidade**: como WSDL é baseado em XML, um documento WSDL pode ser acessado por um grande número de máquinas e sistemas operacionais;
- **Suporte a diversos protocolos**: WSDL suporta diversas alternativas para a interação com *web services*. Embora o SOAP seja a opção mais utilizada, um cliente enviando mensagens RMI/IIOP também pode invocar o serviço. O segredo está na primitiva <binding> do WSDL, que informa ao cliente como invocar o serviço utilizando um protocolo de comunicação específico;

- **Falta de ordenamento**: a descrição do padrão não prevê a ordem da execução das operações. O próprio cliente deve procurar informações a respeito dessa ordem, talvez tentando inferir através do nome das operações. O aspecto positivo é que, como essa ordem não é definida, o cliente pode configurá-la da forma que achar mais conveniente.

Um documento WSDL é composto, basicamente, por dois grupos de definições: concreta e abstrata (também chamados, respectivamente, de descrições funcionais e não-funcionais). A parte abstrata descreve o que o serviço faz em função das mensagens que ele envia e recebe, contudo sem endereçar preocupação de como e onde o serviço é oferecido. A parte concreta é resposável por vincular fisicamente o cliente ao serviço (similar a IDL de CORBA). Na totalidade, WSDL (Versão 1.1) é composta por seis elementos, a saber: *definitions*, *types*, *message*, *portType*, *binding* e *service*. A Figura 2.5 apresenta a estrutura básica de um documento WSDL, incluindo o relacionamento entre os seus principais elementos.

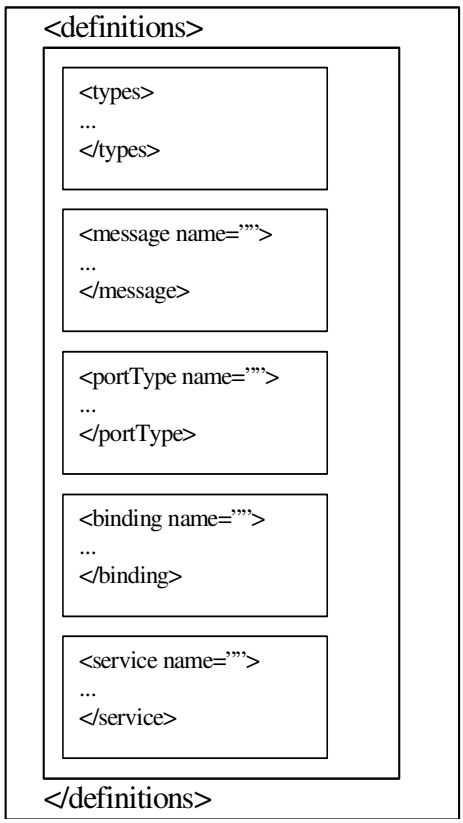

Figura 2.5: Estrutura básica de um documento WSDL

O elemento raiz em um documento WSDL é o *definitions* (*wsdl:definitions*). Ele contém todos os elementos presentes na Figura 2.5, bem como outros para especificar o *namespace* (ver Seção 2.2.1.1) alvo do documento e diversos *namespaces* auxiliares para evitar o conflito de nomeação. Os *namespaces* que são definidos neste elemento são visíveis em todo o documento WSDL.

O elemento *types* (`wsdl:types`), por sua vez, é utilizado para declarar um tipo WSDL. Esses tipos são usados posteriormente para a definição das mensagens que um serviço envia e recebe. Um tipo de dados definido pelo *types* é, basicamente, uma variável composta de tipos primitivos (semelhante às estruturas da linguagem C).

O próximo elemento apresentado é o *message* (`wsdl:message`). A sua função é a de descrever logicamente as mensagens trocadas pelos computadores distribuídos. Uma mensagem pode usar tipos padrões do esquema XML ou tipos definidos através do elemento *types*. O elemento *message* define o nome da mensagem e contém elementos *part*, que se referem aos parâmetros de entrada ou aos valores de retorno.

O elemento *portType* (`wsdl:portType`) agrupa diversos elementos *operation*, que são simplesmente um agrupamento de um conjunto de mensagens relacionadas que são trocadas. De forma resumida, o *portType* é o conjunto de todas as operações suportadas por um determinado *web service*. Através do *operation*, um *portType* pode combinar uma mensagem de requisição com uma mensagem de resposta, caracterizando uma determinada operação. É importante ressaltar que o *portType* pode, e geralmente define, vários elementos *operation*. Adicionalmente, podem-se dividir os tipos de operações em quatro categorias básicas:

- **One-way**: o cliente envia uma mensagem para o serviço, e o mesmo não produz nenhuma mensagem como resposta;
- **Request-response**: o cliente envia uma mensagem para o serviço, que é posteriormente respondida pelo mesmo;
- **Solicit-response**: o próprio serviço solicita uma mensagem, e a recebe posteriormente; e
- **Notification**: o serviço envia uma mensagem, e nada recebe como resposta (envia apenas para notificar o cliente de alguma situação).

Os elementos *types*, *message* e *portType* descrevem o que o *web service* faz, baseado essencialmente nas mensagens trocadas. A principal finalidade do elemento *binding* (`wsdl:binding`) é agrupar as informações necessárias para se conectar fisicamente ao serviço. WSDL não assume uma maneira padrão para formatar mensagens. Ao invés disso, ele define como trocar mensagens utilizando SOAP ou HTTP. Fazer a conexão através do SOAP é mais usual, especialmente porque as operações HTTP *Get* e *Post* não podem representar facilmente todas as estruturas XML necessárias. Uma boa discussão sobre este tema está presente em [Hendricks 2002].

Por fim, o elemento *service* (`wsdl:service`) é a parte final da descrição do *web service* e é responsável por descrever informações acerca da localização do serviço. Ele é composto por um ou mais elementos *port*, que especificam onde o serviço pode ser localizado, fornecendo a URL e a porta do mesmo.

Maiores detalhes sobre o padrão WSDL podem ser obtidos em [Christensen 2001].

2.2.1.4 UDDI

Com a implementação do *web service* finalizada, é natural que o programador tenha a intenção de registrar o mesmo em algum lugar, com o intuito de disponibilizá-lo para outros usuários. Ao invés de enviar o documento WSDL para cada cliente em potencial, ele pode registrar as informações sobre o serviço em um repositório geral que esteja disponível publicamente para todos os clientes interessados. Além disso, não é interessante que todas as pessoas utilizem tecnologias diferentes para realizar este registro; desta forma, o problema da interoperabilidade não estaria sendo resolvido pelos *web services*, mas apenas sendo deslocado para um outro nível, o nível de registro de serviços.

Este ambiente propiciou o surgimento do UDDI (Universal *Discovery, Description and Integration*) [OASIS 2004]. O UDDI é um mecanismo padronizado e transparente para descrever serviços, disponibilizar formas simples para a solicitação dos mesmos e prover registros centrais, que são acessíveis por todos os usuários. Na prática, os servidores UDDI duplicam as informações sobre os *web services* entre si, com o objetivo de garantir que todos os serviços possam ser acessados a partir de um único servidor (e de tal sorte que ele seja constantemente atualizado). Adicionalmente, os dados UDDI são estruturados de maneira semelhante a de um catálogo telefônico, onde informações sobre proprietário e telefone são armazenadas. As informações são compostas de três tipos de entradas:

- **Páginas brancas**: contém informações básicas de contato, como nome do desenvolvedor, endereço e telefone. Outras informações, legíveis para leigos, também podem ser aqui colocadas para subsidiar uma melhor escolha do serviço;
- **Páginas amarelas**: contém listagens comerciais baseadas essencialmente nos tipos dos negócios. Desta forma, o UDDI possibilita que se registre entradas baseadas no tipo do negócio ou até mesmo pelo segmento industrial que ele opera; e
- **Páginas verdes**: são utilizadas principalmente para fornecer detalhes técnicos do serviço para subsidiar o desenvolvimento dos clientes.

A estruturação acima apresentada pode ser utilizada tanto para o registro do *web service* quanto para a consulta dos dados no UDDI.

Um outro elemento de extrema importância no mecanismo UDDI é o seu modelo técnico (conhecido como tModel) [OASIS 2004]. O tModel representa uma especificação técnica em um registro UDDI. Esse elemento pode representar vários elementos, tais como a interface WSDL de um *web service* ou até mesmo a URL do seu esquema XML). Portanto, se um novo serviço for criado, e for colocada a interface WSDL do mesmo no UDDI, ela será armazenada como um tModel.

Por fim, uma questão interessante no contexto do UDDI é o tipo de descoberta. Em tese, existem dois tipos: o primeiro é a descoberta em tempo de projeto, onde o serviço é descoberto ainda durante o desenvolvimento do cliente. Neste tipo, o processo é feito de modo manual, com a descoberta e configuração manuais. O outro tipo é a descoberta em tempo de execução, que

envolve a descoberta automática do serviço desejado através da especificação do tipo do *web service* desejado. Embora mais prático, este tipo de descoberta carece de uma maior segurança, pois o ambiente em que esta busca ocorre é ainda muito inseguro.

2.3 Composição

Uma das principais idéias defendidas pelo SOA é a de reuso, onde sistemas complexos podem ser desenvolvidos utilizando serviços mais básicos. Neste contexto, surge novamente o conceito de composição de serviços. Este conceito não representa uma idéia totalmente nova. Em um passado recente foram realizadas tentativas de se desenvolver sistemas incorporando serviços pré-existentes [Plummer 2004]. Contudo, alguns pontos impediram o sucesso deste conceito:

- Sistemas demasiadamente complexos e de baixo nível;
- Grande esforço necessário para desenvolvimento (especialmente em sistemas distribuídos e heterogêneos);
- Falta de padrões.

Graças ao advento dos *web services*, não apenas o SOA voltou a ser tópico de extremo interesse, mas também a composição de serviços. Baseados em um conjunto de padrões suficientemente aceitos e de alto nível, os *web services* propiciam um ambiente favorável à composição de serviços.

Como exemplo de composição de serviços, pode-se imaginar um tradutor de línguas capaz de traduzir textos do Português para o Árabe. Se não houver disponível um tradutor direto para as duas línguas, mas sim um tradutor de Português para Inglês e outro de Inglês para Árabe, o serviço pode ser criado através da composição dos outros dois pré-existentes.

Os serviços podem ser compostos de maneira estática ou dinâmica [Benatallah 2002]. Esta escolha depende do tipo do processo que está sendo composto. Se os serviços que serão compostos têm uma natureza fixa, ou mudam raramente as suas interfaces e semântica, a composição estática satisfaz as necessidades. Contudo, um processo pode conter um conjunto de funções fracamente definidas a ser realizado, ou então ele tem que ter a possibilidade de se adaptar dinamicamente à mudanças não previstas no ambiente. Para estes casos, a composição estática pode não ser a abordagem mais adequada, pois alterações em um sistema baseado em composição estática requer a interrupção da continuidade do processo. Composição dinâmica é uma alternativa importante neste caso, dado que existem processos críticos que não podem sofrer interrupções.

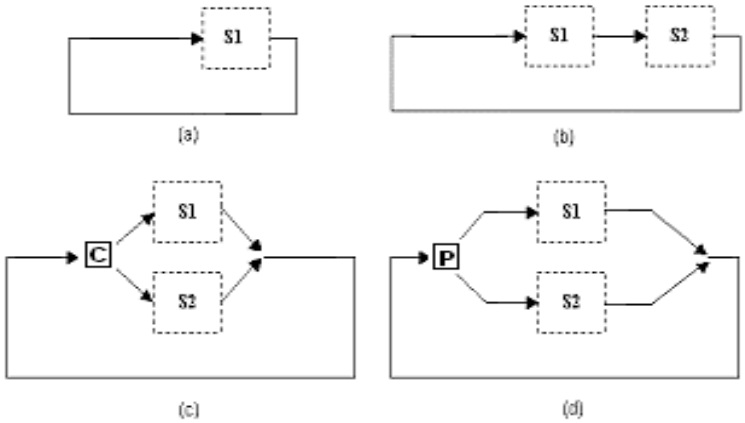

Figura 2.6: Tipos básicos de composições

Além da natureza estática e dinâmica da composição, os *Web services* podem ser compostos de diversas formas. Maneiras tradicionais de composição incluem: composição iterativa, composição seqüencial, composição com escolha e composição paralela. Em uma composição iterativa, um serviço S1 é invocado várias e consecutivas vezes, como mostrado na Figura 2.6(a). Já na Figura 2.6(b), é exposto uma composição seqüencial: um serviço S1 seguido de um serviço S2 significa que S1 deve terminar sua execução antes de S2 começar a sua. Já na composição com escolha, dois ou mais serviços (por exemplo, S1 e S2) são compostos, contudo apenas um deles é escolhido para ser executado. Esta escolha é feita por um critério específico (incluindo critérios randômicos). Este caso se encontra mostrado na Figura 2.6(c), onde C representa o operador de escolha. Finalmente, na composição paralela, dois ou mais serviços executam ao mesmo tempo e de forma independente. A Figura 2.6(d) apresenta esta situação (onde P representa o operador de paralelismo). Neste tipo específico de composição, os serviços podem interagir ou não (ou seja, pode ou não haver comunicação entre os mesmos). Estes diferentes tipos de composição podem ser incorporados em linguagens de composição utilizando definições de operadores e combinadores [Cardeli 1999].

2.3.1 Composição de *Web Services*

A utilização de um único serviço em geral não é suficiente para a implementação de lógicas de negócio complexas. Logo, com o objetivo de completar as promessas da proposição da arquitetura orientada a serviços, *web services* devem ser compostos com o objetivo de se construir aplicações maiores e de significativa complexidade. Como visto anteriormente, implementar sistemas complexos baseados no uso de serviços mais simples facilita o trabalho de desenvolvimento e promove a reusabilidade.

Várias tecnologias podem ser utilizadas na área de composição de serviços (não necessariamente *web services*). Contudo, uma forte tendência está direcionando os esforços de pesquisa e implementação para a implementação de *web services* compostos. Alguns fatores contribuem para este fato:

- *Web services* possuem interfaces bem definidas. Através do padrão WSDL, os mesmos possuem interfaces padronizadas, nas

quais podem ser extraídas as informações funcionais do serviço sem grandes dificuldades;

- *Web services* estão normalizados (descritos em WSDL invocado através de XML dentro de mensagens SOAP);
- Linguagens tradicionais não foram desenvolvidas tendo a composição em mente, com isso, para o seu uso, outros aspectos devem ser avaliados, como por exemplo: conversão de dados, criação de mensagens e ligação dinâmica de componentes (*dynamic binding*) durante a composição. O projeto em si teria uma duração mais longa por causa da análise (e implementação) desses aspectos.

A composição de *web services* se preocupa com a implementação interna das operações dos serviços. O desenvolvimento de um serviço composto de uma empresa é mantido privado, mesmo que sejam utilizados outros *web services* externos. E, mais importante ainda, a execução do serviço composto ocorre de forma transparente para os clientes, ou seja, eles não sabem (e nem precisam saber) que um serviço na verdade invoca diversos outros para a produção do resultado requisitado.

Uma das propriedades mais importantes de uma composição de *web services* é a de que a composição é também um *web service*. Esta propriedade é de fundamental importância no contexto do SOA, na medida em que os próprios *web services* compostos podem ser utilizados como simples *web services* participantes em composições e lógicas de negócio ainda mais complexas. Assim como os *web services* mais simples, os compostos utilizam WSDL (Seção 2.2.1.3) para a especificação de sua interface, o SOAP (Seção 2.2.1.2) para o envio e recebimento de informações e o UDDI (Seção 2.2.1.4) para se registrarem e serem localizados.

Contudo, de fato, não existe atualmente um padrão para especificar e implementar composições de *web services*. A maneira mais usual na atualidade de se realizar esta tarefa é a criação de processos de negócio (*business processes*) utilizando, em especial, o padrão WS-BPEL [OASIS 2007]. Estes serão detalhados nas próximas seções.

2.3.2 Processos de Negócio (*Business Processes*)

Os processos de negócio (*business processes*) visam especificar um conjunto de atividades realizadas pelas organizações [OASIS 2007]. Estão associados a este conceito as informações por ele manipuladas, a utilização de recursos (*resources*) e elementos da estrutura organizacional da empresa. Aqui, o foco principal é no negócio, e neles são baseadas a especificação e a implementação dos processos. Exemplos típicos de processos de negócio são: fabricação de produto, agendamento de consulta, efetuar compra, dentre vários outros exemplos.

Teoricamente, pode-se afirmar que os processos de negócio são uma coleção de invocações de serviços e atividades relacionadas que produzem um resultado de negócio, tanto para uma única empresa ou diversas outras. Os *business processes* visam, em especial, a integração de sistemas, parceiros comerciais e usuários corporativos através de processos empresarias flexíveis,

com o intuito de reduzir custo, melhorar a eficiência operacional e criar novas oportunidades empresarias. Conforme a tecnologia da informação continua a se desenvolver, um número cada vez maior de sistemas conectáveis e de usuários empresarias estão na grande rede. Isto faz com que tantos os benefícios quanto os desafios de implementação de soluções sejam cada vez evidentes. Graças à complexidades e custo envolvidos no desenvolvimento e gerenciamento de conexões em todos estes sistemas e pessoas, o processos de negócio continua sendo um objetivo difícil para muitas empresas.

Há muitos anos, existem soluções de integração no mercado. Contudo, muitas soluções acabam onerando a tarefa de integração ao adicionar tecnologias defasadas e de difícil implementação. Em geral, essas soluções colocam *overhead* desnecessário na aplicação, não lida com todos os aspectos das necessidades do cliente e não interagem de forma transparente com outras tecnologias também importantes, tais como plataformas e sistemas operacionais diversos.

Neste cenário foi proposto o padrão WS-BPEL [OASIS 2007]. Basicamente, este padrão é uma linguagem focada na execução de processos de negócio, que permitem que vários *web services* trabalhem de forma conjunta com o objetivo de executar uma tarefa relacionada ao negócio. No momento atual, WS-BPEL não é apenas um dos principais padrões existentes que possibilitam a criação de processos de negócio, mas possivelmente o principal a efetuar, de fato, composição de *web services*.

Desta forma, atualmente, a composição de *web services* está estreitamente relacionada com a criação e execução de processos de negócio. Como conseqüência, as formas atuais de execução de composição de *web services* estão mais ligadas aos processos de negócio em si, e não têm relação direta com padrão de composição de serviços, como, por exemplo, LOTOS [Bolognesi 1987]. Este fato é decorrente da inexistência de um padrão específico para a composição de *web services*; em verdade, se utiliza um padrão voltado a negócio e a *workflow* (BPEL) como principal caminho para se realizar composições de *web services*, padrão este que não foi projetado com objetivos específicos de possibilitar diversos tipos de composições.

2.3.3 WS-BPEL

Embora não exista nenhum padrão consensual na área de processos de negócio e composição de *web services*, WS-BPEL se tornou o padrão de *fato* para estas tecnologias. Ela surgiu da combinação das linguagens WSFL (IBM *Web Service Flow Language*) e XLANG.

WS-BPEL fornece uma linguagem para a especificação dos processos de negócio e dos estados dos mesmos, descrevendo como ocorre o relacionamento entre os diversos *web services* participantes da composição. Neste contexto, deve ser especificado como um processo de negócio interage com os *web services* participantes com o objetivo de obter o resultado desejado e também como especificar o próprio *web service* composto. De certa maneira, WS-BPEL é similar a algumas linguagens de programação já existentes, na medida em que ela oferece determinados tipos de construções como *loops*, desvios condicionais, variáveis e atribuições. Este fato acaba possibilitando um tratamento algorítmico do processo de negócio. Contudo, WS-BPEL oferece

esses recursos de uma maneira bastante simplificada, com o intuito de facilitar o aprendizado e a utilização da mesma.

Um processo de negócio especificado em WS-BPEL tem sua interface baseada no padrão WSDL, troca informações com outros *web services* através do padrão SOAP e pode ser cadastrado em um UDDI para fins de busca e consulta. Ou seja, na prática, um *business process* baseado em WS-BPEL se comporta igual a um *web service*, tornando verdadeira a afirmação de que, inclusive na prática, a composição de *web services* se constitui um *web service*. Adicionalmente, o padrão WS-BPEL assume que os serviços a serem compostos estão descritos usando WSDL.

Especificações baseadas em WS-BPEL usam XML para descrever aspectos do processo: parceiros nas trocas de mensagens, serviços disponíveis, a orquestração [Juric 2006], o formato das mensagens, dentre outros aspectos.

2.3.3.1 Arquitetura da Execução: Orquestração *versus* Coreografia

Uma questão importante em WS-BPEL é a arquitetura da execução de um processo de negócio. De forma geral, existem dois tipos de arquiteturas de execução: orquestração e coreografia.

Na orquestração, uma figura central (orquestrador), que pode ser tanto a própria *engine* (ver Seção 2.3.4) ou um outro serviço, tem controle sobre os *web services* envolvidos e coordena as mais diversas operações entre os mesmos. Os *web services* participantes da composição não sabem que eles fazem parte de uma composição por meio de um *business process*, e nem precisam ter sido implementados objetivando diretamente essa questão. Apenas o orquestrador sabe disto, e deve tomar todas as providências para executar as invocações na ordem correta e as operações necessárias para a obtenção do resultado desejado. Este tipo de arquitetura é utilizado extensamente em processos de negócio privados (ou seja, restritos a uma única empresa ou conjunto de empresas). A Figura 2.7 apresenta um exemplo de orquestração.

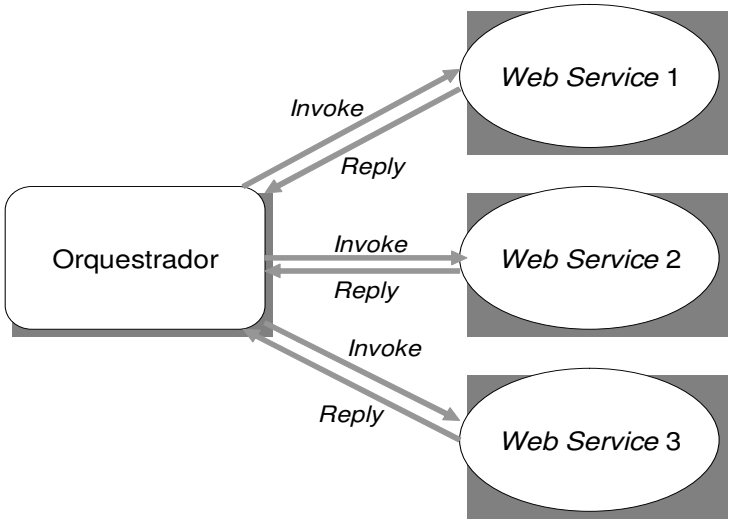

Figura 2.7: Exemplo de orquestração

Neste tipo de arquitetura, pode-se especificar com exatidão todos os detalhes requisitados para a execução do processo de negócio. Processos deste gênero são denominados processo de negócios executáveis.

Já no caso da coreografia, a arquitetura de execução do processo de negócio (e da composição) não é baseada em uma figura centralizada. Cada *web service* envolvido na coreografia deverá saber a hora exata que deverá ser executado e com quem deverá interagir. Todos os serviços participantes da composição deverão estar atentos a questões no negócio, que operações executar, que mensagens trocar, dentre diversos outros detalhes. Este tipo de arquitetura de execução é focado em processos de negócios públicos. A Figura 2.8 apresenta um exemplo de coreografia.

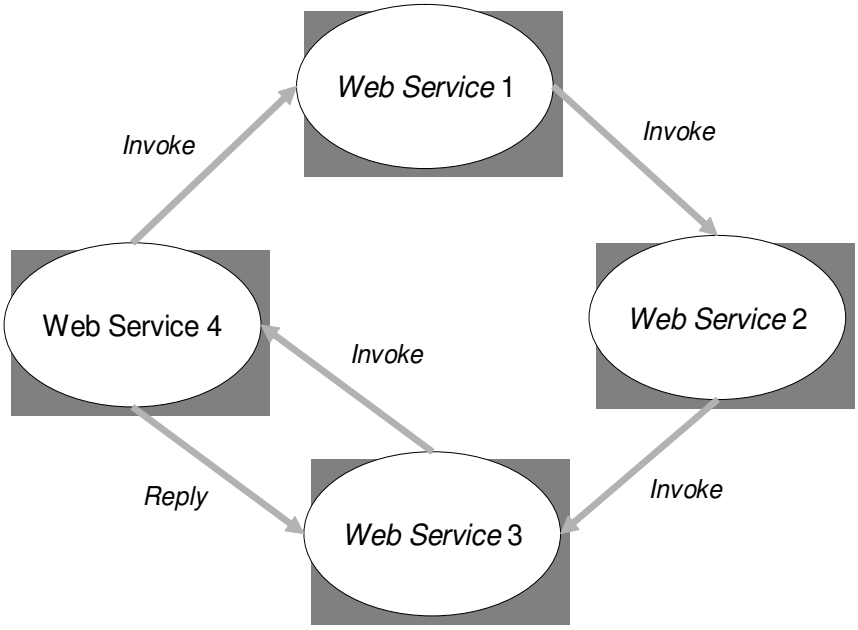

Figura 2.8: Exemplo de coreografia

Em contraste à orquestração, a coreografia especifica apenas a troca de mensagens entre os participantes, não incluindo detalhes internos do fluxo do processo e não é executável. Processos deste tipo são denominados processos de negócio abstratos.

Em termos de composição de *web services*, a orquestração apresenta duas grandes vantagens em relação à coreografia:

- Novos *web services*, mesmo que não saibam que vão ser utilizados em uma composição de serviços, podem vir a ser incorporados;
- Existe um responsável central pela execução da tarefa, que serve de ponto inicial para a busca de falhas e possíveis alterações.

No cenário atual do SOA e dos *web services*, a orquestração e os processos executáveis vêm sendo extensamente utilizados, enquanto a coreografia e os processos abstratos são raramente usados, e o seu cenário de uso mais comum é o de modelo para a definição de processos executáveis. WS-BPEL é focado na orquestração, enquanto não existe um padrão forte na área de

coreografia; o CDL4WS (*Choreogray Description Language for Web Services*) vem sendo discutido pela W3C, mas até o presente momento nenhum documento de referência sobre tal proposta foi disponibilizado.

2.3.3.2 Especificação de Processos em WS-BPEL

Nesta seção, são discutidas as principais partes de uma especificação em WS-BPEL. A Figura 2.9 apresenta os principais componentes de tal documento. Em seguida, segue-se uma breve explanação sobre os mesmos.

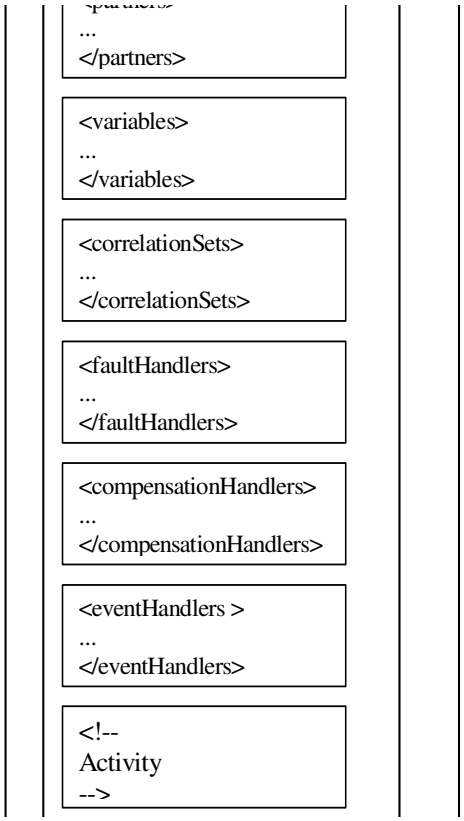

Figura 2.9: Estrutura básica de uma especificação BPEL

De forma genérica, a *tag* `<partner>` especifica quem são os serviços participantes da composição. Tipicamente, nesse local são armazenados os endereços dos servidores que estão disponibilizando os *web services*. Um outro termo bastante utilizado neste contexto é o de *partner links* (`<partnerLink>`), o qual se refere a qualquer recurso disponível, como grande exemplo o próprio *partner*.

A *tag* `<variable>` representa uma das grandes vantagens do uso deste padrão, que é a possibilidade de armazenamento de estados durante a execução da composição. Variáveis podem ser inicializadas, terem seus valores enviados

para os serviços participantes e até mesmo serem alteradas dependendo da resposta dos mesmos.

Adicionada recentemente [OASIS 2007] ao padrão WS-BPEL, a *tag* `<correlationSet>` identifica para qual instância de processo uma nova mensagem recebida deve ser roteada. Este elemento é também responsável por não permitir que existam duas interações distintas de *web services* caso elas sejam oriundas do mesmo parceiro (*tags* `<partner>` e `<partnerLink>`), utilizem a mesma operação e possuam o mesmo conjunto de correlações (`<correlationSet>`).

O objetivo da *tag* `<faultHandlers>` é capturar falhas (eventos inesperados, erros, etc.) e tentar reverter a situação, a fim de que o trabalho seja finalizado. Em um processo de negócio, várias situações podem gerar problemas, desde a indisponibilidade momentânea de serviços até erros na própria lógica do processo.

Em muitos processos de negócios, não se pode terminar toda a execução dos mesmos em uma única transação atômica, principalmente porque os processos exigem longo tempo de execução no qual várias transações processadas criam efeitos persistentes antes do processo ser completado. No WS-BPEL, a *tag* `<compensationHandlers>` reverte atividades completadas. Ele é bastante utilizado associado ao `<faultHandler>`, situação na qual ele possibilita que atividades já realizadas (inclusive com resultados armazenados) em um processo de negócio que apresentou algum tipo de erro em sua execução sejam canceladas e retorne ao estado inicial.

Adicionalmente, WS-BPEL fornece meios de se tratar eventos (em especial, assíncronos) concorrentemente à execução do processo através do uso da *tag* `<eventHandlers>`. Como exemplos de eventos assíncronos, podem-se citar o estouro de tempo de execução de uma determinada operação e mensagens enviadas pelo cliente para a monitoração do progresso da execução do processo de negócio. Uma propriedade importante desta *tag* é que a mesma pode conter uma atividade associada ao evento, ou seja, o processo de negócio é capaz de responder de alguma forma a novos eventos.

Por fim, um processo BPEL é composto de atividades (*activity*). As atividades podem ser divididas em duas categorias: básicas (ou primitivas) e estruturadas. Uma apresentação completa de todas estas atividades seria demasiadamente extensa, então optou-se por agrupar as mesmas em tabelas, associando o nome da atividade com a sua principal funcionalidade. A Tabela 2.1 apresenta as atividades básicas utilizadas no WS-BPEL.

Tabela 2.1: Atividades básicas

Atividade	Principal Função
`<invoke>`	Invocação de *web services* de parceiros (*partners*)
`<receive>`	Recebe a invocação por parte do cliente

<assign>	Atualização de variáveis
<reply>	Envia a resposta do processamento para o cliente
<throw>	Usada para sinalizar uma falha interna de maneira explícita (opcionalmente, fornece uma variável para armazenar o erro correspondente)
<terminate>	Finaliza imediatamente a execução de uma instância de um *business process*
<wait>	Especifica um atraso na execução do processo por um certo período
<empty>	Atividade que não realiza nenhuma tarefa (usada em <faultHandling>, por exemplo)

Processos de negócio podem ser formados pela utilização de apenas três destas atividades básicas: <receive>, <invoke> e <reply>.

As atividades estruturadas são responsáveis por especificarem a ordem em que uma coleção de atividades deverá ser executada. A Tabela 2.2 explicita as atividades estruturadas apresentadas na especificação do WS-BPEL [OASIS 2007].

Tabela 2.2: Atividades estruturadas

Atividade	Principal Função
<sequence>	Processa atividades de forma seqüencial
<switch>	Executa desvios condicionais (similar às linguagens de programação convencionais)
<pick>	Espera a ocorrência de um evento e executa uma atividade associada ao mesmo
<flow>	Executa atividades de forma concorrente
<link>	Expressa dependências de sincronização associadas à utilização do <flow>
<while>	Executa laços na execução das atividades (também similar às linguagens de programação convencionais)
<scope>	Define um novo escopo para

	variáveis, <faultHandlers>, <compensationHandlers> e <eventHandlers>

Alguns trabalhos [ActiveBPEL 2006] criam uma outra categoria, denominada atividades especiais, e classificam as atividades <scope> e <terminate> como integrantes dessa nova categoria. Contudo, a especificação do padrão WS-BPEL só apresenta duas categorias, e ela é tomada como base no contexto deste trabalho.

De todas as atividades apresentadas, a atividade básica <invoke> é a mais importantes do WS-BPEL, pois ela é responsável pela invocação dos *web services* participantes. Por esse motivo, ela é descrita com mais detalhes a seguir.

2.3.3.3 Primitiva *Invoke*

WS-BPEL suporta invocações de *web services* de forma síncrona e assíncrona. Na invocação síncrona (requisição/resposta), o processo envia uma requisição para o serviço participante e fica aguardando por uma resposta. As invocações são bloqueantes, ou seja, se um processo de negócio invocar um determinado *web service*, ele deverá esperar o envio da resposta do mesmo para continuar a sua execução. Esta operação tem início com o envio da requisição e finaliza com o recebimento da resposta. Por outro lado, a invocação assíncrona normalmente é usada para operações que podem demorar um longo período de tempo para finalizarem o seu processamento. Neste contexto, o *web service* que invoca a operação pode continuar processando instruções durante a execução do *web service* participante. Neste caso, em conjunto com a *tag* <invoke> (que se preocupa neste caso apenas com o envio da solicitação) deverá ser utilizada uma *tag* <receive>, a fim de receber a resposta por parte do serviço. Entre o <invoke> e o <receive>, pode-se realizar outras atividades. Adicionalmente, a associação entre a resposta e o <receive> é realizada através de <correlationSets> [OASIS 2007].

Em ambos os tipos de invocação, <invoke> precisa das informações dos parceiros (obtidas através do <partnerLink>) para os quais vai mandar a requisição. Duas ou mais invocações podem ocorrer tanto de forma seqüencial como de forma concorrente: as atividades estruturadas <sequence> e <flow> devem ser utilizadas para prover estas funcionalidades.

A Figura 2.10 apresenta a estrutura básica de uma invocação baseada no padrão WS-BPEL.

```
<invoke partnerLink="pl_nome" portType="pt_name" operation="op_name"
 inputVariable="iv_nome" outputVariable="ov_nome">
...
</invoke>
```

Figura 2.10: Estrutura básica de uma operação <invoke>

Como se pode perceber, o parceiro alvo da invocação é especificado através do atributo `partnerLink`. Desta forma, o parceiro é associado estaticamente, no próprio código do processo de negócio. Ou seja, com o processo de negócio já especificado e implementado, é impossível mudar o alvo de uma requisição sem parar o sistema e reescrever o seu código. Esta compreensão é de vital importância para este trabalho, à medida que o mesmo é baseado na "quebra" desta rigidez, permitindo que os parceiros sejam trocados em tempo de execução.

2.3.4 *Engines*

Para a execução de um processo de negócio baseado no padrão WS-BPEL, é necessária a existência de um ambiente de execução específico para esta tarefa. As *engines* WS-BPEL são responsáveis por disponibilizarem tal ambiente. Pode-se afirmar que a *engine* é o motor que executa a lógica de negócio especificada na aplicação, invocando as operações definidas pela mesma. Cada execução de um serviço composto é denominada uma instância da composição. Podem existir diversas instâncias ao mesmo tempo, e cabe também a *engine* o gerenciamento dessa questão.

Existe um número extenso de *engines* no mercado atual, e Juric [Juric 2006] apresenta uma lista extensa das mesmas. Como exemplo de *engine*, será aqui apresentada a utilizada neste trabalho, ActiveBPEL [ActiveBPEL 2006]. Por este motivo, ela será alvo de um aprofundamento em alguns aspectos práticos.

A *engine* ActiveBPEL [ActiveBPEL 2006] é um ambiente de execução de processos de negócio (*business processes*) baseada no padrão WS-BPEL. Ela é escrita em Java, e é de código aberto, o que possibilita que desenvolvedores possam investigá-la e, inclusive, alterá-la para atender alguma necessidade específica.

Sob um ponto de vista arquitetural, a *engine* gerencia a execução de um ou vários processos WS-BPEL. Por sua vez, os processos são compostos basicamente por atividades, que implementam a funcionalidade desejada do negócio. O foco desta *engine* é a execução de processos executáveis.

Para a execução de processos de negócio no ActiveBPEL, o seguinte conjunto de arquivos é utilizado:

- ✓ ***.bpr:** este é o arquivo que armazena as informações do processo de negócio. Na verdade, ele é um "repositório" de outros arquivos, tais como: o documento que contém o código BPEL propriamente dito (extensão .bpel), o descritor de implantação do processo (*Process Deployment Descriptor* – extensão .pdd), a interface WSDL do serviço (extensão .wsdl) e todos os arquivos de parceiros necessários para a implantação do mesmo (opcional);
- ✓ ***.pdd:** este tipo de arquivo informa a *engine* detalhes importantes para a execução do processo de negócio, como a descrição dos <partnerLinks> e dos arquivos WSDL necessários;
- ✓ **wsdlCatalog.xml:** serve como um catálogo dos documentos WSDL, provendo para a *engine* uma forma de encontrar esses arquivos no aquivo descritor de implantação (.bpr). Este arquivo é

utilizado no caso de as interfaces WSDL dos *web services* participantes se encontrarem dentro do próprio projeto, e não na Internet.

Baseada na documentação oficial do ActiveBPEL, é apresentada na Figura 2.11 a estrutura física (em termos de diretórios e arquivos) de um arquivo descritor de implamantação de um processo (*.bpr). Nesta figura, destacados com letras maiúsculas se encontram os diretórios. Através da mesma, uma idéia geral da estruturação de um projeto nesta *engine* pode ser obtida.

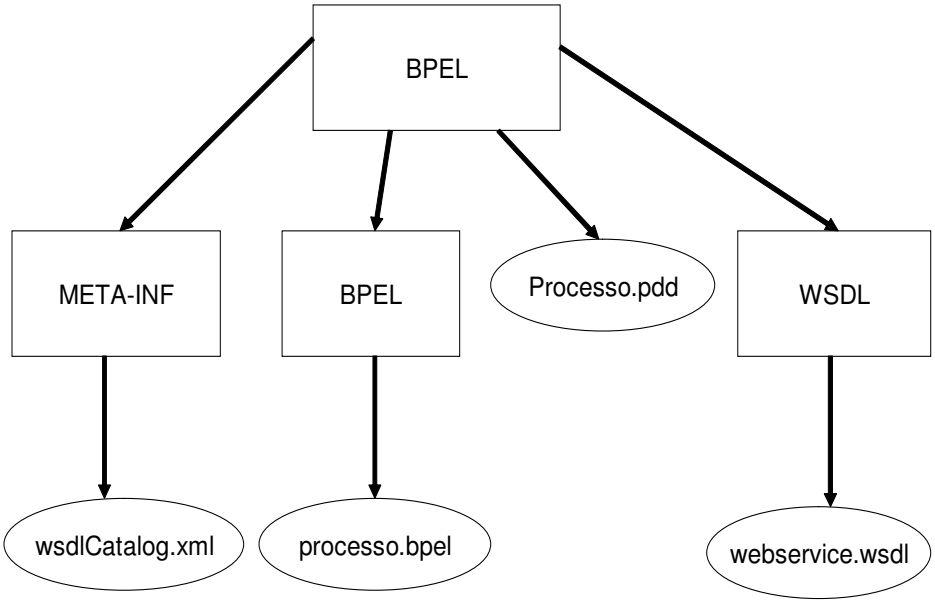

Figura 2.11: Estrutura geral de um arquivo .bpr

Por fim, a *engine* ActiveBPEL gerencia questões como: persistência de dados (importante para transações de longa duração), filas, alarmes e diversos outros detalhes inerentes a execução de processos de negócio.

Maiores detalhes sobre esta *engine*, incluindo questões de configuração de dependência, instalação, execução e especificação dos arquivos necessários (.bpr, .pdd e wsdlCatalog.xml) pode ser encontrada na documentação oficial sobre a mesma [ActiveBPEL 2006].

2.4 Adaptabilidade

O interesse em sistemas adaptativos tem crescido ao longo dos anos. Devido à disseminação da computação móvel, ubíqua e autônoma, diversas questões relacionadas à adaptabilidade vem sendo estudadas [Agha 2002]. No contexto dos *web services*, a adaptabilidade decorre do fato de que novos *web services* são disponibilizados de maneira muito freqüente. Esses novos serviços poderiam ser tratados de uma maneira automática, sem a necessidade de se parar todo o sistema para a inclusão dos mesmos. Adicionalmente, a Internet

não provê um ambiente de comunicação totalmente confiável, e os servidores que disponibilizam os *web services* podem passar por diversos problemas. Neste contexto, a importância da utilização de mecanismos de adaptabilidade aparece com forte destaque.

Adaptações podem ser estáticas ou dinâmicas, manuais ou automáticas e proativas ou reativas [McKinley 2004] [Courbis 2005]. Adaptações estáticas são aquelas que sofrem modificações ainda no código-fonte da aplicação, em fase de projeto, enquanto que as dinâmicas geralmente modificam as características do software em tempo de execução. Adaptações manuais ocorrem através de intervenções direta no sistema, enquanto que as automáticas podem ser realizadas pelo próprio sistema. Finalmente, adaptações proativas ocorrem anteriormente a um determinado evento, enquanto que a reativa ocorre após este determinado evento. Por exemplo, o sistema pode perceber que um dos serviços que ele utiliza não está disponível e tomar atitudes antes da realização do mesmo (caracterizando uma adaptação proativa). Se ele tomar atitudes apenas depois da chamada a este serviço e apresentar um erro, estaremos tratando de uma adaptação reativa.

Em termos de implementação de sistemas adaptativos, segundo McKinley [McKinley 2004], existem duas metodologias principais: adaptação parametrizada e adaptação composicional.

A adaptação parametrizada, de forma geral, trabalha modificando variáveis que determinam o comportamento do sistema. Um determinado algoritmo do programa pode receber diversos parâmetros, e o sistema adaptativo poderia controlar o valor desses parâmetros a depender das condições no ambiente. Embora relativamente mais simples de implementar do que a adaptação composicional, esta metodologia não permite que novas funcionalidades e/ou características sejam adicionados à aplicação após a mesma ter sido implementada.

Por sua vez, a adaptação composicional possibilita a adição de novas funcionalidades e/ou características ao sistema através da troca de componentes do mesmo. Esta metodologia é bastante poderosa, à medida que necessidades que não foram implementadas em tempo de projeto podem ser inseridas no sistema sem a necessidade de reescrita de código e, até mesmo, de paralização dos serviços oferecidos. Contudo, esta flexibilidade provoca uma maior dificuldade (em relação à adaptação parametrizada) de implementação e descoberta/teste de erros. Pelo exposto, a adaptação composicional tem atraído um maior interesse por parte dos pesquisadores nos últimos anos.

Atualmente, existe um fator limitante, que impede a completa disseminação da adaptação relacionada à composição de *web services*. As principais tecnologias que suportam composição de *web services* (ex.: WS-BPEL) fazem a conexão dos parceiros durante o processo de implantação do serviço, em termos de portTypes (ver Seção 2.2.1.3). Isto é feito de forma estática, e na utilização padrão destas tecnologias não existe uma maneira direta de se efetuar mudanças nos parceiros em tempo de execução [Karastoyanova 2005]. Quando um processo WS-BPEL é implementado, os arquivos WSDL de todos os serviços participantes da composição devem ser conhecidos e todos os possíveis parceiros declarados. Não são permitidas mudanças dinâmicas.

Engines como ActiveBPEL [ActiveBPEL 2006] tentam driblar esta limitação de forma indireta, contudo apresentam outros problemas decorrentes da própria proposta inicial do WS-BPEL. Por exemplo, na *engine* ActiveBPEL, utilizando a estratégia *dynamic partner binding* [ActiveBPEL 2006] (que permite que ocorra a adaptação desde que os parceiros sejam especificados na fase de desenvolvimento do sistema), é suportada adaptação dinâmica de parceiros em tempo de execução. Contudo, utilizando-se esta estratégia, novos parceiros não estariam englobados nesta proposta (e, neste contexto, essa seria a necessidade mais evidente).

2.4.1 Onde, quando e como?

McKinley [McKinley 2004] propõe uma taxonomia para a estruturação de um sistema adaptativo. Segundo ele, o mesmo pode ser descrito baseando-se em três pontos básicos: "como", "quando" e "onde".

O ponto "como" se refere ao mecanismo utilizado que possibilita a adaptação. Estes mecanismos devem ser baseados em alguma técnica [McKinley 2004] que deve ser implementada diretamente no sistema (ou nos artefatos que dão suporte a ele). Um número expressivo de técnicas atua causando uma indireção entre as principais entidades do sistema, principalmente através de interceptações (transparente para a aplicação) ou por intermédio de um *middleware* integrado (deve ser invocado explicitamente pela aplicação).

Por sua vez, o ponto "quando" se refere ao momento em que a adaptação deve ser implementada, mais especificamente em tempo de desenvolvimento do software, em tempo de compilação, em tempo de implantação ou em tempo de execução. Idealmente, a adaptação deve ocorrer o mais tarde possível. Contudo, é importante observar que a checagem da consistência do sistema (associada com as adaptações) é facilitada nos estágios menos avançados, especialmente nos estágios de desenvolvimento e de compilação. Uma adaptação é classificada como estática ou dinâmica tendo como base o tempo em que a mesma ocorre. Se a mesma é executada em tempo de desenvolvimento ou de compilação, a mesma é denominada estática. Caso contrário, se for executada em tempo de execução, é dita dinâmica.

Finalmente, o ponto "onde" se refere ao local do ambiente onde será inserido o código referente à adaptabilidade. Este código poderia, por exemplo, ser inserido no sistema operacional, na *engine* que executa a composição ou até mesmo no próprio código-fonte da aplicação.

2.4.2 *Late Binding*

Em termos da adaptação composicional e dinâmica, surge a estratégia *late binding* [McKinley 2004]. De forma direta, esta estratégia permite que componentes compatíveis possam ser incorporados no sistema em tempo de execução. Esta incorporação pode ser executada através de interfaces bem descritas, permitindo que o sistema possa checar a compatibilidade deste componente.

Esta estratégia tem uma importância relevante na disponibilização da adaptação composicional, na medida em que se pode incluir e excluir componentes de uma aplicação em tempo de execução, ao passo que na

adaptação parametrizada pode-se apenas combinar esses componentes e, no máximo, no estágio de compilação. Adicionalmente, esta estratégia estimula a reutilização de *software*.

2.5 Considerações Finais

Este capítulo apresentou conceitos básicos usados na dissertação. Inicialmente, noções sobre *web services* e composição de *web services* foram apresentadas, destacando-se suas características e protocolos, cujo entendimento será importante para uma análise mais profunda dos próximos capítulos. O padrão WS-BPEL foi introduzido, explicitando visões e características que serão importantíssimas não apenas para o entendimento dos capítulos posteriores, mas também para uma correta análise das contribuições deste trabalho. Por fim, não se poderia deixar de trabalhar o conceito de adaptabilidade, o qual foi apresentado através de uma abordagem simples, mas poderosa, baseada em três pontos essenciais: “onde”, “quando” e “como”.

CAPÍTULO

3

Composição Adaptativa de *Web Services*

"Sei que meu trabalho é uma gota no oceano.
Mas, sem ele, o oceano seria menor."

Madre Tereza de Calcutá.

Este capítulo detalha a proposta para o tratamento da adaptação em composição de *web services*. Inicialmente são apresentados os princípios adotados na proposta, considerando os aspectos de quando, onde e como a adaptabilidade é provida. Seguindo estes princípios, é apresentada a arquitetura da *engine* A-ActiveBPEL que realiza os princípios definidos. Por fim, são apresentados detalhes da implementação da *engine* A-ActiveBPEL e um exemplo de seu uso.

3.1 Princípios Básicos

Como mencionado anteriormente, este trabalho visa prover adaptabilidade em tempo de execução de maneira transparente e preservando os padrões já consolidados da área (como, por exemplo, WS-BPEL). Para atender estes requisitos, a utilização da presente abordagem requer a satisfação de dois requisitos:

- Tendo em vista a troca de *web services* parceiros da composição em tempo de execução, serviços com diferentes implementações devem possuir a mesma interface. Logo, assume-se que os *web services* podem ter implementações diferentes e podem ter sido

implementados por pessoas distintas, mas precisam ter a mesma interface WSDL.

- O projetista da composição pode prover a qualquer instante, inclusive em tempo de execução, os novos *web services* existentes. Desta forma, quando surgir um novo *web service* que o projetista acredite ser útil para a composição, ele deve informar a *engine* a localização (URL) do mesmo, a operação desejada e suas propriedades não-funcionais. A *engine* adaptativa se responsabilizará pelo resto.

Considerando estes dois pontos principais, detalhamos a seguir os princípios básicos adotados na proposta.

3.1.1 Quando, Como e Onde

Na Seção 2.4.1 foram apresentados três aspectos a serem considerados em projetos relacionados à adaptabilidade: como, quando e onde a adaptação é executada.

O primeiro aspecto, "como", é tratado no contexto de WS-BPEL da seguinte forma: quando a *engine* vai executar uma especificação, ela é analisada e cria-se a estrutura utilizada em tempo de execução para o processamento da composição. Depois da leitura da especificação, mudanças na mesma não surtem efeito em tempo de execução, a menos que o sistema seja reinicializado. Isto acontece basicamente porque a estrutura para a chamada de *web services* é estática e os parceiros são previamente definidos na própria chamada através do `<invoke>` (Seção 2.3.3.2). Contudo, a semântica associada à invocação pode ser modificada. A idéia principal é procurar mudanças em um repositório específico imediatamente antes de se efetuar cada invocação. Se for observada alguma modificação, um mecanismo adaptativo específico é ativado antes da chamada ser direcionada para o *web service* participante. Caso contrário, a execução do processo de negócio segue seu curso normal. Em termos mais práticos, esta abordagem intercepta a chamada, examina os *web services* disponíveis e escolhe um deles baseado em um determinado critério (Seção 3.1.3).

A Figura 3.1 apresenta um exemplo desta idéia. Inicialmente, existe uma composição baseada em WS-BPEL composta de três *web services* (WS1, WS2 e WS3). A composição é passada para a *engine* adaptativa (1) que começa sua execução. Quando uma primitiva `<invoke>` é encontrada na composição indicando a invocação de uma operação, a *engine* verifica se houve mudanças no repositório (2), recebe o resultado desta checagem (3) e executa a invocação (4). Quando um novo *web service* é desenvolvido (WS4), disponibilizado (colocado no repositório), tem a mesma interface e funcionalidade de WS1, mas sob algum critério de qualidade é "melhor" do que WS1, a *engine* atua e a requisição correspondente é redirecionada para WS4 (5).

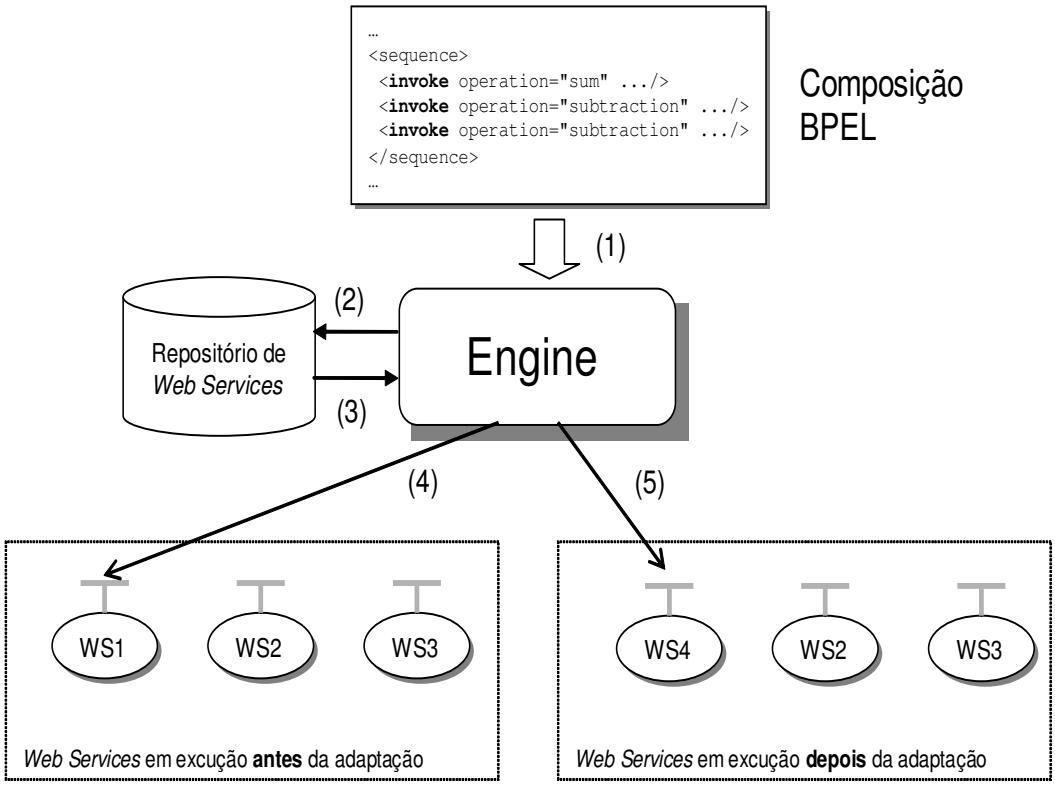

Figura 3.1: Visão Geral da Proposta de Adaptação

Como mencionado anteriormente, o segundo aspecto básico é "quando" executar a adaptação. A abordagem proposta adota uma estratégia específica que possibilita o acoplamento de serviços compatíveis (com diferentes implementações, mas com semântica equivalente e interfaces WSDL iguais) em tempo de execução, baseada em parte na estratégia *late binding* (Seção 2.4.2). As mudanças podem ocorrer a qualquer momento, contudo elas só serão percebidas pela *engine* adaptativa e só terão efeito nos instantes anteriores à chamada ao *web service* participante, quando o mecanismo adaptativo é acionado (caracterizando a adaptação dinâmica). É importante observar que a utilização da idéia do *late binding*, nesta situação, se torna essencial, em especial nos casos em existe um número elevado de chamadas de *web services* e/ou quando as mesmas requisitam um longo tempo para o término de sua execução.

Estendendo esta idéia, este trabalho propõe que mudanças ocorridas durante a execução de uma determinada chamada possam ser avaliadas imediatamente antes das chamadas dos *web services* participantes. Desta forma, são contempladas mudanças ocorridas antes da execução de um determinado processo de negócio e mudanças quando a execução já está em andamento. Supondo que uma composição seqüencial de dois *web services* (WS1 e WS2) está sendo executada, se ocorrerem mudanças no ambiente como, por exemplo, o aparecimento de um novo *web service* WS3 (com interface e semântica equivalentes a WS2), contudo mais seguro, e considerando que este outro *web service* seja mais apropriado para o processo de negócio do que WS2 (por ser mais seguro, se segurança for o critério adotado para as adaptações), a *engine*

adaptativa será capaz de verificar esta mudança e atuar para redirecionar o alvo da chamada se a mesma ainda não tiver sido executada.

Finalmente, o último aspecto a ser considerado é "onde" o mecanismo adaptativo deve ser inserido. Na abordagem proposta para este trabalho, o código deste mecanismo é inserido dentro da própria *engine*, com a mesma passando a ser chamada de *engine* adaptativa. Desta forma, as sintaxes das primitivas do padrão WS-BPEL, em especial o `<invoke>`, não precisam ser alteradas.

3.1.2 Políticas

No contexto da proposta deste trabalho, vários *web services* podem estar disponíveis em um dado instante, e a *engine* adaptativa deverá escolher quais serão utilizados no processo de negócio. Desta forma, a escolha de quem irá ser utilizado é uma decisão importante que a *engine* adaptativa deverá efetuar.

Nesta abordagem, os *web services* participantes e a própria composição têm uma política, especificando o que eles requisitam e como os serviços são oferecidos. Estas políticas contem informações fundamentais para os clientes dos *web services* e da composição. A política da composição pode especificar a qualidade do serviço que é disponibilizado pela própria composição, e pode também expressar os requisitos necessários para a execução do serviço na maneira esperada por outras composições (vale lembrar que uma composição de *web services* é considerada um *web service*, sendo assim possível que a mesma participe de outras composições).

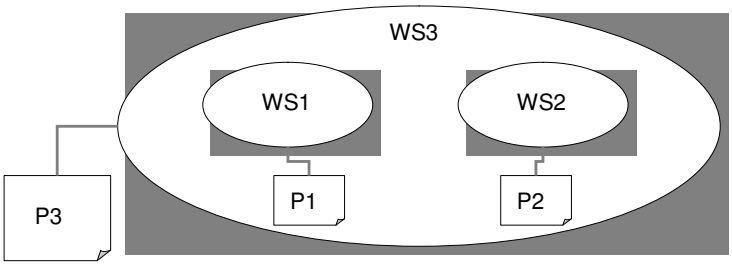

Figura 3.2: Políticas associadas com os *web services*

O exemplo da Figura 3.2 contém dois *web services* (WS1 e WS2), a própria composição (WS3) e as políticas associadas (P1, P2 e P3). As políticas definem propriedades não-funcionais sobre os serviços para o *Web Service Repository*. Desta maneira, a *engine* adaptativa é capaz de saber se algum *web service* satisfaz o critério adotado. Baseado na idéia de que uma composição é um *web service*, ela terá sua própria política, descrevendo a qualidade do seu serviço e os requisitos para provê-lo com essa qualidade. Adicionalmente, como esta própria composição pode até fazer parte de uma outra composição, a sua política pode ser avaliada por outras composições com o objetivo de verificar a qualidade que ela oferece seus serviços, assim como o que ela requisita também para poder oferecer esta qualidade.

Sob um ponto de vista mais prático, políticas podem ser armazenadas de várias formas, tanto como num simples arquivo texto (ligado diretamente à interface do *web service*) ou até mesmo inseridas diretamente no *Web Service*

Repository. Neste trabalho, informações sobre políticas foram armazenadas diretamente no repositório. Uma política pode ser descrita em termos de atributos, que são utilizados para subsidiar o processo de sua análise.

3.1.3 Critérios de Adaptação

Diversos *web services* podem estar disponíveis em um dado instante, e cabe a *engine* adaptativa escolher quais deles irão ser executados pela composição. A qualidade do processo de negócio será influenciada diretamente pelas escolhas da mesma, ou seja, é fundamental fazer uma boa escolha. A *engine* adaptativa pode tomar a decisão baseada em requisitos não-funcionais impostos pela aplicação e que devem ser satisfeitos pelos *web services* participantes da composição. Por exemplo, uma aplicação pode definir que uma determinada composição deve ser segura. Desta forma, mudanças realizadas nos *web services* participantes devem preservar este requisito. Caso contrário, não se deve permitir que essas mudanças sejam realizadas. Em geral, esses tipos de requisitos se estendem a toda composição, ou seja, se um simples *web service* participante falhar ao tentar atender um requisito específico, toda a composição é afetada. Outras propriedades não-funcionais como tolerância a falhas, desempenho [Silva 2006], custo, disponibilidade e várias outras [Zeng 2004] podem ser consideradas em uma adaptação.

Em termos práticos, o critério adotado na adaptação deve ser passado para a *engine* adaptativa, e as informações sobre as propriedades não-funcionais dos *web services* participantes devem ser armazenadas no *Web Service Repository* e anexadas ao serviço correspondente. Em particular, as propriedades não funcionais que pode ser associadas a *web services* e à composição incluem estas definidas em [O'Sullivan 2005]:

- **Provedor de serviços**: esta categoria captura informações sobre o nome do serviço e o provedor do mesmo;
- **Modelo temporal**: especifica conceitos temporais que serão necessários para a descrição de serviços (por exemplo, para se fazer uma reserva em um hotel, o cliente pode precisar saber o formato da hora que é gerado pelo serviço);
- **Modelo de localização**: apresenta as propriedades não-funcionais relativas à localização (por exemplo, informações como telefone, endereço eletrônico e endereço completo com CEP);
- **Disponibilidade de serviços**: é composto pela junção dos modelos temporal e de localização, e se destina a especificar onde e quando um cliente pode utilizar o serviço;
- **Modelo de obrigações**: captura as obrigações tanto do servidor como do cliente. A idéia é que estas propriedades estejam disponibilizadas para serem analisadas pelas partes interessadas, facilitando o processo de escolha.
- **Modelo de custos**: esta categoria visa definir o preço da utilização de um serviço. Este preço deve ser definido pelo provedor do serviço, enquanto os clientes devem estar cientes do mesmo para a sua utilização;

- **Modelo de pagamento**: este modelo visa especificar os diversos tipos de pagamentos disponíveis, como por exemplo cartão, saque em conta corrente e cheque;
- **Modelo de descontos**: categoria reponsável por oferecer atrativos para os clientes, em especial relacionados aos meios de pagamento;
- **Modelo de penalidades**: utilizado principalmente pelo provedor do serviço, especificando o que irá ocorrer se o cliente não atender determinado requisito ou obrigação;
- **Modelo de direitos**: captura essencialmente os direitos associados aos provedores de serviços e aos clientes dos mesmos para a execução de operações (pode ser usado em dois contextos: de acordo com o preço estabelecido pode existir a disponibilização de direitos e quando uma obrigação não for satisfeita, direitos podem ser revogados);
- **Modelo de linguagens**: categoria utilizada para especificar informações relativas a linguagens (em especial, a linguagem falada/escrita e a linguagem de programação/descrição utilizada);
- **Modelo de confiança:** modela aspectos de confiança entre o provedor de serviço e o cliente (embora o conceito de confiança seja utilizado de diversas formas, exemplos de utilização incluem, por exemplo, a segurança do meio de pagamento e o nível de confiabilidade do serviço atestado por um terceiro reconhecidamente confiável);
- **Modelo de qualidade**: categoria bastante abrangente, que procura capturar todos os aspectos relativos a qualidade do serviço. Pode conter elementos numéricos, comparações com outros padrões e até mesmo comparações com *benchmarks*;
- **Modelo de segurança**: abrange todas as informações relativas à segurança no serviço, em especial relacionadas a identificação (comprovar a identidade do mesmo) e a confidencialidade (dados trafegados entre serviço/provedor não devem ser acessíveis por terceiros).

De forma geral, qualquer uma destas propriedades pode ser utilizada como critério de adaptação.

3.2 Arquitetura da A-ActiveBPEL

Existem diversas *engines* que se baseiam na especificação WS-BPEL e executam processos de negócios. Como exemplo podem-se citar a *engine* ActiveBPEL [ActiveBPEL 2006] e a *Oracle BPEL Process Manager* [Oracle 2006]. Em geral, elas diferem em aspectos como: maneira de organizar os arquivos necessários à composição e à própria especificação, forma com que tratam aspectos relacionados à qualidade de serviços (QoS), se são de código aberto ou não, o seu público alvo e, finalmente, se a mesma implementa aspectos como persistência, alarmes, filas, gerenciadores de falhas, dentre outras características.

A extensão proposta, A-ActiveBPEL (*Adaptive* ActiveBPEL), é uma extensão adaptativa da *engine* ActiveBPEL. A adoção desta *engine* foi baseada

em alguns critérios. Primeiramente, embora a proposta deste trabalho seja independente de qualquer *engine* particular, uma deve ser escolhida como exemplo de implementação. Em segundo lugar, a maioria das *engines* disponíveis na atualidade não tem código-fonte aberto. Como conseqüência, não existe a possibilidade de alteração do código-fonte tendo em vista a incorporação da abordagem proposta. Em terceiro lugar, dependendo da *engine*, diferentes linguagens e plataformas são utilizadas, o que impõe diversas dificuldades para a realização de uma implementação mais genérica. Finalmente, ActiveBPEL é vastamente utilizada na academia e na indústria, é escrita em Java e é bem documentada.

A arquitetura geral da extensão A-ActiveBPEL é apresentada na Figura 3.3.

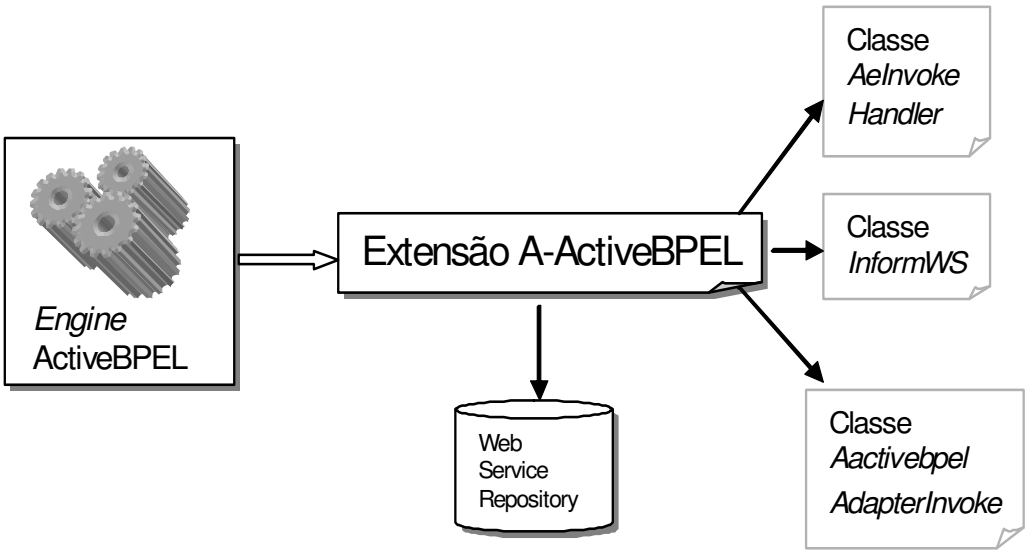

Figura 3.3: A extensão A-ActiveBPEL (arquitetura geral)

A extensão proposta é composta por três classes, a saber: classe AeInvokeHandler, classe InformWS e classe AactivebpelAdapterInvoke. A primeira é a reimplementação da classe de mesmo nome existente na *engine* ActiveBPEL, tendo em vista incorporar o mecanismo adaptável na mesma. Adicionalmente, a extensão inclui também um repositório, neste trabalho denominado Web Service Repository. Todos estes artefatos serão alvos de estudo e detalhamento na próxima seção.

3.2.1 Implementação da A-ActiveBPEL

Foi visto anteriormente que o padrão WS-BPEL especifica diversas primitivas com o intuito de permitir várias formas de composição e de estruturação do negócio: primitiva de sequenciamento (`<sequence>`), primitiva de concorrência (`<flow>`) e primtiva de iteração (`<while>`). Operações de entrada e saída são especificadas com o uso de duas primitivas, `<receive>` e `<while>`, respectivamente. Como mencionado anteriormente, a abordagem proposta se concentra na primitiva `<invoke>` que realiza invocações aos *web services* participantes da composição. O código abaixo

apresenta um exemplo de especificação em BPEL, onde a primitiva de invocação é destacada.

```
<process name="Sum" ...>
 <partnerLinks>
 ...
 </partnerLinks>
  <receive ...> ... </receive>
  <reply ...> ... </reply>
  <sequence>
   <invoke operation="sum" .../>
   <invoke operation="subtraction" .../>
  </sequence>
</process>
```

A sintaxe da primitiva de invocação continua a mesma, contudo a extensão define uma nova semântica para essa primitiva: antes de realizar a chamada para os *web services* participantes, o <invoke> verifica se existe alguma modificação no *Web Service Repository* e no critério de adaptação (Seção 3.1.3).

A extensão A-ActiveBPEL é um conjunto de arquivos Java colocados juntos com o código-fonte da *engine* ActiveBPEL. Modificações foram feitas na classe AeInvokeHandler do ActiveBPEL. Esta classe é responsável por gerenciar o processo de invocação dentro da *engine*. A criação da chamada é feita com esta classe, e parâmetros são passados tendo em vista a configuração desta chamada. Antes da chamada ser criada, pode-se manipular estes parâmetros. O método específico chamado, setTargetEndpointAddress(), pode ser invocado para atribuir um novo endereço. Adicionalmente, duas classes adaptiveInvoke e informWS, foram introduzidas. A Figura 3.4 apresenta uma visão prática da *engine* adaptativa. O método adaptiveInvoke() é chamado imediatamente antes da invocação do *web service* (1). Após esta etapa, o método adaptiveInvoke realiza uma busca no *Web Service Repository* com o intuito de verificar se existem mudanças (2) e pegar informações sobre os *web services* candidatos para execução (se necessário) (3). Finalmente, o método adaptiveInvoke configura todas as variáveis de ambiente necessárias para redirecionar a chamada para o endereço do *web service* que melhor satisfaz o critério adotado (4).

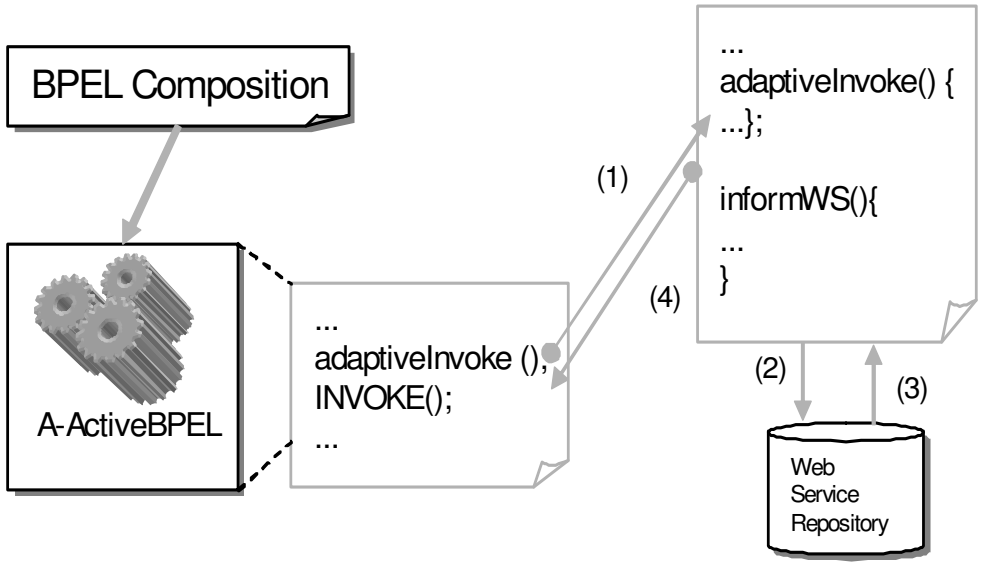

Figura 3.4: Visão geral da implementação

A código a seguir apresenta parte do código da implementação da extensão A-ActiveBPEL, presente na classe AeInvokeHandler.

```
(1) import org.activebpel.aactivebpel.AactivebpelAdapterInvoke;
(2) import org.activebpel. aactivebpel.InformWS;
(3) public IAeWebServiceResponse handleInvoke
(4)  {
(5)  ...
(6)   Call call = createCall (wsdlService, operation,
(7)                           endpointReference,
(8)                           addressHandling);
(9)   String url = call.getTargetEndpointAddress();
(10)   String operacao = call.getOperation().getName();
(11)   InformWS ws =
  AactivebpelAdapterInvoke.getInstance().adapterInvoke(url,
                                                    operacao);
(12)   call.setTargetEndpointAddress(ws.getURL());
(13)   call.setOperation(ws.getOperation());
(14) ...
(15) }
```

Inicialmente, a chamada começa a ser construída através do construtor `Call (6)`, que se baseia nas variáveis de ambiente da própria *engine* para

configurar os seus parâmetros. As *strings* url e operacao são declaradas, com o intuito de verificar quais as opções que foram originalmente ofertadas no código-fonte da aplicação, descrito no padrão WS-BPEL. Essas *strings* são passadas como parâmetros para o método `adapterInvoke()` da classe `AactivebpelAdapterInvoke`. O retorno deste método especifica o *web service* eleito pelo mecanismo adaptativo, que é armazenado em objeto do tipo `InformWS`, criado pela extensão especialmente para armazenar os *web services* eleitos. O código da classe `InformWS` é mostrado a seguir.

```
(1)package org.activebpel.aactivebpel;
(2)
(3)public class InformWS {
(4)  private String url;
(5)  private String operation;
(6)  public InformWS(String url, String operation){
(7)       this.url = url;
(8)       this.operation = operation;
(9)       }
(10)      public String getURL(){
(11)      return url;
(12) }
(13)      public String getOperation(){
(14)      return operation;
(15) }
(16)}
```

Esta classe armazena os dois atributos relevantes para a chamada de um *web service*: o seu endereço (`url`), na linha (4), e a sua operação (`operation`), na linha (5).

Finalmente, a classe `AactivebpelAdapterInvoke` será agora detalhada. Esta classe executa uma das tarefas fundamentais no contexto da extensão A-ActiveBPEL: executa a seleção dos *web services* participantes, baseada em algum critério.

Inicialmente, é desejado que exista apenas uma instância desta classe em um determinado instante execução, pois várias instâncias podem ser criadas (especialmente porque podem existir várias chamadas a *web services* participantes em uma aplicação, já que a mesma pode ser executada diversas vezes), sendo imprescindível que apenas a correta seja invocada. Para isto, esta classe foi implementada adotando-se o padrão de projetos *Singleton*, que é utilizado quando for necessária a existência de apenas uma instância de uma classe. Desta forma, ele mantém um ponto de acesso global a este objeto.

Contudo, como garantir que haverá apenas uma instância? Uma delas é restringindo o acesso ao construtor, tornando-o um método privado. Uma outra é sempre utilizar um método `getInstance()`, que faz o controle da instanciação, criando um novo objeto apenas quando necessário.

```
(1) package org.activebpel.aactivebpel;
(2)
(3)import java.util.Iterator;
(4)import java.io.File;
(5)import org.jdom.Document;
(6)import org.jdom.Element;
(7)import org.jdom.JDOMException;
(8)import org.jdom.input.SAXBuilder;
(9)
(10)public class AactivebpelAdapterInvoke {
(11)
(12)  private final String ATRIBURL = "url";
(13)  private final String ATRIBOPERATION = "name_operation";
(14)  private final String CRITERION = "version";
(15)  private static String FILENAME =
(16) "C:\\temp\\BPEL\\tomcat-activepel20\\bin\\WebServices.xml";
(17)  private static SAXBuilder parser = null;
(18)  private static Document document = null;
(19)  private static AactivebpelAdapterInvoke singleton = null;
(20)  private static File xmlFile ;
(21)  private static long lastModified = 0;
(22)
(23)  protected AactivebpelAdapterInvoke(){}
(24)
(25)  public static AactivebpelAdapterInvoke getInstance(){
(26)        if (lastModified < xmlFile.lastModified()) {
(27)              try {
(28)                    document = parser.build(FILENAME);
(29)                    lastModified = xmlFile.lastModified();
(30)              } catch (JDOMException e) {
(31)                    // TODO Auto-generated catch block
(32)                    e.printStackTrace();
(33)              }
(34)        }
(35)        return singleton;
(36)  }
```

```
(37)
(38)  static {
(39)        singleton = new AactivebpelAdapterInvoke();
(40)      parser = new SAXBuilder();
(41)      parser.setValidation(false);
(42)        xmlFile = new File(FILENAME);
(43)        try {
(44)              document = parser.build(FILENAME);
(45)              lastModified = xmlFile.lastModified();
(46)        } catch (JDOMException e) {
(47)              // TODO Auto-generated catch block
(48)              e.printStackTrace();
(49)        }
(50)  }
(51)
(52)public InformWS adapterInvoke(String url, String operation){
(53)      Element groups = document.getRootElement();
(54)        Iterator itGrupo = groups.getChildren().iterator();
(55)        while (itGrupo.hasNext()) {
(56)              Element group= (Element) itGrupo.next();
(57)             Iterator itWS = group.getChildren().iterator();
(58)            while(itWS.hasNext()) {
(59)               Element WS = (Element) itWS.next();
(60) if(WS.getAttributeValue(ATRIBURL).equalsIgnoreCase(url))
(61)               {
(62)
(63)if(WS.getAttributeValue(ATRIBOPERATION).equalsIgnoreCase(oper
ation)       return chooseWS(group);
(64)               }
(65)            }
(66)}
(67)         return new InformWS(url,operation);
(68)  }
(69)
(70)  private InformWS chooseWS(Elementgroup){
(71)      Iterator itWS = group.getChildren().iterator();
(72)        Element greaterWS = (Element) itWS.next();
(73)        while(itWS.hasNext()){
(74)          Element WS = (Element) itWS.next();
(75)if(Integer.parseInt(greaterWS.getAttributeValue(CRITERION)) <
Integer.parseInt(WS.getAttributeValue(CRITERION))){
```

```
                    greaterWS = WS;
                  }
(76)      }
(77)return new InformWS(greaterWS.getAttributeValue(ATRIBURL),
(78) greaterWS.getAttributeValue(ATRIBOPERATION));
(79)  }
(80)
(81)  public static void main(String[] args)
(82)        {}
(83)}
```

Ao se analisar a estrutura geral da classe, pode-se perceber que o método `getInstance()`, linha 25, funciona efetivamente segundo o padrão *Singleton*: inicialmente, a primeira instância da classe é criada, e depois o `getInstance()` só faz atualizá-la, através da análise do *Web Service Repository*. Como será descrito na Seção 3.3, este repositório é construído através de um arquivo texto, escrito em um formato compatível com o padrão XML. Desta forma, `getInstance()` avalia se este arquivo foi recentemente alterado (caso em que novos *web services* podem ter sido adicionados); se tiver, ele carrega as novas informações para análise de qual *web service* será eleito; caso contrário, ele simplesmente mantém a instância como está atualmente.

Finalmente, o método `adapterInvoke()`, linha 52, é responsável por sair percorrendo as informações obtidas através do repositório, em especial as propriedades não-funcionais relacionadas com os *web services* descritos.O critério utilizado está especificado no atributo `CRITERION` (linha 14), que no caso específico é o `version`.

3.3 *Web Service Repository*

Como mencionado anteriormente, a extensão A-ActiveBPEL tem um repositório que provê informações sobre os *web services* participantes (como endereço, operações e propriedades não-funcionais). Na prática, o repositório pode usar diferentes formas de armazenar essas informações. Sistemas gerenciadores de banco de dados (SGBDs) como Oracle e MySQL podem ser utilizados neste contexto, em especial quando uma grande quantidade de informações estiver disponível. Contudo, este repositório pode ser tão simples como um arquivo de texto.

3.3.1 Estratégia Adotada

Na implementação corrente, a estratégia adotada utiliza um padrão baseado em XML para realizar a descrição dos dados em arquivo texto. Com o objetivo de tornar a tarefa de atualização mais fácil para o usuário, *web services* que tiverem a mesma funcionalidade são colocados em um grupo. Quando a *engine* adaptativa analisa a especificação WS-BPEL, ela identifica a qual grupo o *web service* pertence. Quando o grupo tem dois ou mais *web services*, o critério adotado é utilizado para decidir para qual deles será repassada a chamada.

Nesta estratégia, existe a necessidade de especificar a localização do *web service*, o nome da operação que será invocada (um *web service* pode ter diversas operações) e propriedades não-funcionais sobre o serviço, que irão caracterizar a sua política. O conjunto dessas propriedades não-funcionais constitui a política deste *web service*. Todo *web service* inserido no repositório deve ter a sua política especificada, descrevendo a qualidade que é oferecida por ele e provendo elementos com o objetivo de facilitar a escolha que a *engine* deverá fazer. As políticas dos serviços são especificados em um arquivo texto baseado no padrão XML (Seção 2.2.1.1), que representa o *Web Service Repository*.

O código abaixo apresenta um exemplo de uma especificação baseada na estratégia apresentada nesta seção. As políticas trazem como informação não-funcional a versão dos *web services*, que é representada através do atributo `version`, em um ambiente onde o critério de adaptação escolhido poderia ser a versão mais recentemente disponibilizada (versão mais atual).

```
(1) <GROUPS>
(2) <GROUP name="Add">
(3)   <WS url="http://172.16.100.58:8080/Ws1.jws" name_operation="add"
version="7"/>
(4) <WS url="http://172.17.115.206:8080/Ws1.jws" name_operation="add"
version="6"/>
(5) </GROUP>
(6) <GROUP name="Sub">
(7) <WS url="http://172.16.100.58:8080/Ws2.jws" name_operation="sub"
version="6"/>
(8) <WS url="http://172.17.115.206:8080/Ws2.jws" name_operation="sub"
version="4"/>
(9) </GROUP>
(10) </GROUPS>
```

Este exemplo especifica dois *web services* com as operações *add* (soma) e *sub* (subtração). As interfaces são equivalentes, mas os serviços tem implementação distintas e estão localizados em diferentes servidores (consequentemente, em diferentes endereços IP). Na estratégia adotada, é desejado colocar serviços semanticamente equivalentes no mesmo grupo. Dois grupos foram especificados no código acima, denominados Add e Sub. Supondo que um processo de negócio esteja precisando dessas duas operações, a *engine* irá encontrar os grupos correspondentes e, baseado em algum critério não-funcional (como o mais recentemente disponibilizado) escolhe o mais conveniente para a realização da invocação.

Este é uma maneira compacta e intuitiva de especificar informações sobre os *web services* participantes. Se um programador desejar adicionar um

novo *web service*, ele necessita apenas localizar o grupo correspondente e colocar a informação requisitada no formato proposto (opcionalmente com outras informações não-funcionais). Baseado na abordagem proposta, esta informação será disponibilizada em tempo de execução para a *engine*, sem a necessidade de reinicialização do sistema.

É importante observar que esta é apenas uma estratégia possível para a construção do *Web Service Repository*, contudo existem outras que podem ser desenvolvidas e utilizadas dependendo do contexto da aplicação e dos objetivos do projeto.

Uma outra consideração a ser feita é que diversos outros atributos podem ser especificados, e eles serão vistos pela *engine* adaptativa. Para a leitura deste documento XML, utilizou-se o JDOM [Harold 2003], que acaba percorrendo toda a lista de atributos dentro da *tag* <WS>. Mesmo que eles não sejam utilizados, a *engine* acaba os lendo, possibilitando inclusive a avaliação de vários desses atributos em conjunto, a depender das necessidades específicas do projeto. Este fato facilita a extensibilidade da proposta deste trabalho, à medida que situações mais complexas poderão também ser avaliadas, levando em consideração em conjunto, por exemplo, aspectos como segurança, desempenho e custo financeiro.

Finalmente, é importante fazer a ressalva de que a utilização do *Web Service Repository* em aplicações mais robustas pode gerar uma quantidade elevada de informações dentro do mesmo. Sugere-se que um coletor de lixo seja implementado no repositório, tendo em vista eliminar os resíduos provenientes das várias atualizações que surgirem. Em aplicações menos robustas, o projetista pode fazer este trabalho, devido a menor quantidade de intervenções requisitadas.

3.4 Considerações Finais

Este capítulo objetivou apresentar a abordagem proposta neste trabalho para possibilitar a composição adaptativa de *web services*. É importante mencionar que a principal contribuição são os conceitos associados à proposta, e que a implementação do mesmo junto a *engine* ActiveBPEL foi apenas um exemplo de implementação, ou seja, diversas outras *engines* podem se basear nessa abordagem e implementar composições adaptáveis de *web services* com um alto grau de transparência e em tempo de execução.

A presente proposta propicia adaptabilidade não apenas antes da execução de um determinado processo de negócio, mas inclusive durante a execução do mesmo, especificamente no caso em que o *web service* participante a ser trocado ainda não tenha sido executado. Embora ainda não englobe todos os tipos de casos possíveis, este é mais um passo em direção a este objetivo. Restam apenas os casos nos quais o serviço a ser trocado está sendo atualmente executado ou já foi executado e o processo de negócio ainda não finalizou sua execução. A utilização de *web services* com estado [Foster 2005] no contexto destes dois casos é vislumbrada, e será comentada na seção de trabalhos futuros (Seção 6.4).

Adicionalmente, em relação aos possíveis critérios de adaptação, diversos modelos de propriedades não-funcionais, baseado no trabalho de O'Sullivan, foram apresentados, e sua exposição serve como base para subsidiar

a escolha dos critérios, à medida que diversos poderão ser escolhidos, tendo em vista as necessidades mais específicas dos diversos tipos de usuários.

Uma outra contribuição associada é a especificação proposta para o *Web Service Repository*. Projetada tendo em vista especialmente a facilidade de atualização pelo usuário, o agrupamento semântico de *web services* facilita o trabalho de atualização e se mostra uma forma de descrição interessante para o contexto da proposta. Para se adicionar um novo *web service* ao ambiente, basta procurar o grupo a ele associado e efetuar a inserção.

CAPÍTULO

4

Estudos de Caso: o Somador Distribuído e o Sistema de Aprovação de Créditos

"A visão é uma faculdade: o ver é uma arte."

Autor desconhecido.

Este capítulo se destina a apresentar os estudos de casos utilizados para demonstrar a viabilidade da abordagem proposta neste trabalho. Duas aplicações foram escolhidas neste contexto, o Somador Distribuído e o Sistema de Aprovação de Créditos. O somador distribuído foi desenvolvido tendo em vista o aspecto didático, apresentando um caso mais simplificado de composição de *web services*. O Sistema de Aprovação de Créditos, por sua vez, apresenta um sistema real de aprovação de crédito, já utilizado, inclusive, em diversos outros trabalhos.

4.1 Somador Distribuído

Este estudo de caso visa, principalmente, mostrar a adequabilidade da proposta deste trabalho no contexto em que ela se apresenta. Ele foi utilizado no artigo publicado associado a este trabalho [Lins 2006].

Implementada especificamente para a utilização nesse trabalho, o Somador Distribuído é conceitualmente mais simples do que o Sistema de Aprovação de Créditos. O processo de negócio inicialmente é composto por dois *web services*, e cada um é capaz tanto de realizar adições de números como

também de subtrações (embora o nome seja somador, ele também tem essa característica). Esta aplicação receberá dois números quaisquer (X e Y), realizará a soma deles (X + Y), realizará também a sua diferença (X-Y) e retornará os dois resultados obtidos. Embora sejam serviços conceitualmente muito simples, vale ressaltar que a complexidade de implementação dos mesmos não é relevante no contexto deste trabalho, pois esta complexidade não é levada em consideração pela *engine* adaptativa na hora da escolha do serviço mais adequado. Adicionalmente, neste estudo de caso, será utilizado outro critério de adaptação (em relação ao capítulo anterior), com o objetivo de demonstrar a extensabilidade da proposta tendo em vista a utilização de outros aspectos não-funcionais.

4.1.1 Ferramentas

Todos os experimentos executados neste estudo de caso utilizaram o servidor Web Apache-Tomcat [Apache 2006]. Dois servidores foram configurados para a execução deste estudo de caso. A única diferença entre os mesmos é que um deles tem a extensão A-ActiveBPEL instalada, provendo o ambiente necessário para a execução do processo de negócio, e o outro apenas disponibiliza outra versão dos mesmos serviços, através de um servidor Web sem a *engine*/extensão instalada (no caso, o Apache-Tomcat com o módulo SOAP). Desta forma, apenas um deles está apto a executar composições de *web services*, com o outro apenas disponibilizando serviços alternativos. A Figura 4.1 apresenta, de forma compacta, os servidores utilizados, incluindo o conteúdo de cada um deles e os serviços que eles disponibilizam. Por fim, um mesmo *web service*, com quatro diferentes versões, foi armazenado nos mesmos, e suas versões foram denominadas de WS1_v1, WS1_v2, WS1_v3 e WS1_v4.

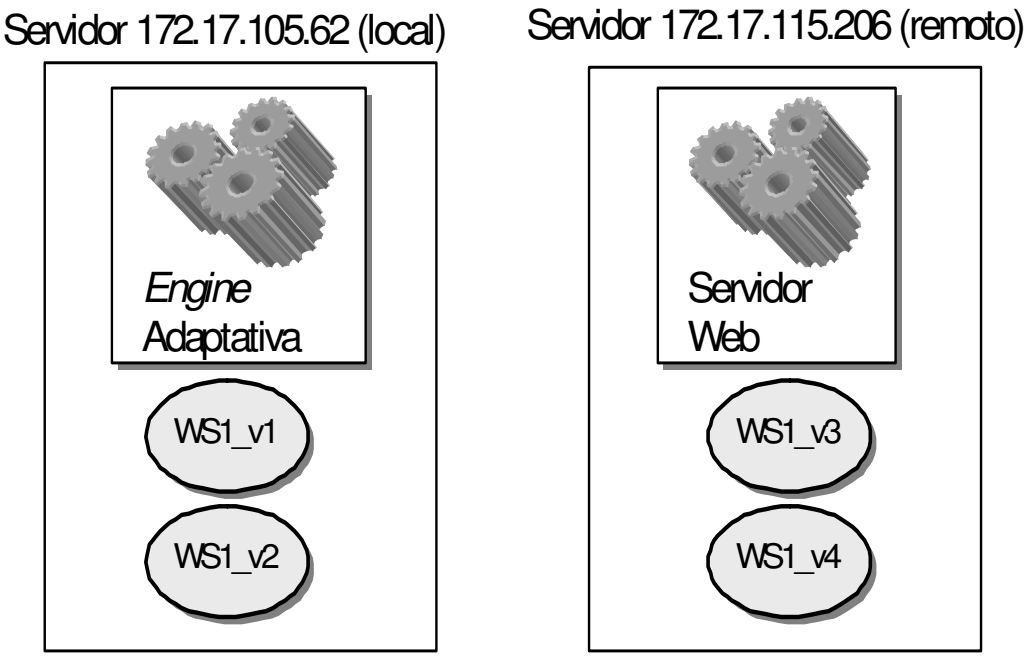

Figura 4.1: Configuração geral dos servidores utilizados

Para o desenvolvimento da aplicação, o *software* ActiveBPEL Designer [ActiveBPEL 2006] foi utilizado. Esta ferramenta CASE auxilia o desenvolvimento do processo de negócio, provendo diversas funcionalidades para a automação de diversas tarefas. Adicionalmente, a ferramenta AXIS, da organização Apache, foi utilizada no processo de desenvolvimento dos *web services* participantes.

4.1.2 Configuração, Execução e Resultados

Como explicado anteriormente, esta aplicação é composta basicamente por duas chamadas a *web services* participantes. Cada serviço especificado no repositório apresenta duas operações possíveis: a soma e a subtração. Desta forma, existe o cenário de ter apenas um *web service* respondendo pelas duas chamadas do processo de negócio. É prudente lembrar que esses *web services* apresentam a mesma sintaxe e possuem semânticas equivalentes, contudo não possuem a mesma implementação.

A configuração física dos servidores e dos *web services* se encontra apresentada na Figura 4.1. Por questões didáticas, os dois *web services* que estão no servidor local serão aqui denominados de WS1_v1 e WS1_v2. Na prática, esses dois serviços têm a mesma interface WSDL, por isso convém dizer que os mesmos são apenas diferentes versões do mesmo serviço. Vale lembrar que mesmo que eles tenham a mesma funcionalidade, eles podem a implementar de formas distintas (com diferentes níveis de qualidade). Desta maneira, os serviços presentes no servidor remoto serão chamados de WS1_v3 e WS1_v4, que também serão diferentes versões do mesmo serviço WS1. Adicionalmente, o endereço IP do servidor local é 172.17.105.62 e o endereço IP do servidor remoto é 172.17.115.206.

A Figura 4.2 apresenta a representação gráfica da implementação da aplicação no ActiveBPEL Designer. Outras atividades, além das primitivas de invocação, foram utilizadas: o <receive> e o <reply>, utilizados para lidar com as operações de entrada e saída; o operador de sequenciamento <sequence> e a atividade <assign>, que é responsável por gerenciar valores de variáveis na execução do processo de negócio.

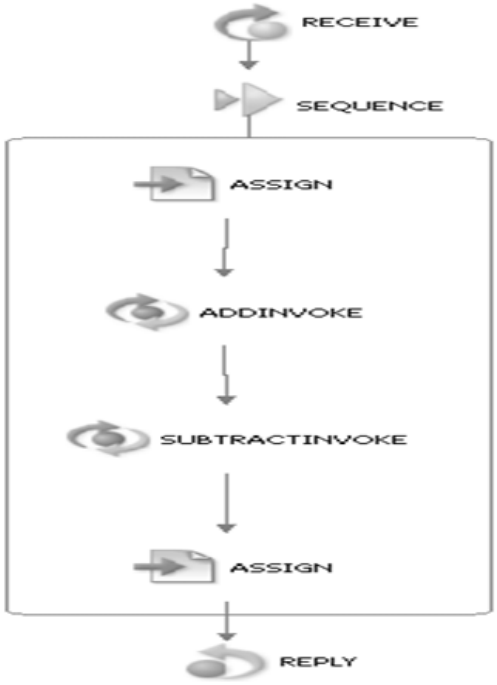

Figura 4.2: Implementação do processo no ActiveBPEL Designer

Neste exemplo, o processo de negócio recebe como entrada dois valores inteiros e retorna como saída a soma e a subtração dos mesmos, respectivamente. Primeiramente, a soma é executada, depois a subtração. A primitiva de invocação <invoke> é executada duas vezes (*addinvoke* e *subtractinvoke*), e inicialmente foi configurado na aplicação que as invocações serão direcionadas, respectivamente, para WS1_v1 e WS1_v2, que se encontram no servidor local. A Figura 4.3 apresenta a configuração inicial do repositório para estas políticas. É conveniente ressaltar que, como esta é a configuração original da aplicação, nem seria necessário configurar o repositório. Contudo, para possibilitar as alterações posteriores em tempo de execução, o repositório já foi aqui inicialmente especificado.

```
<GROUPS>
<GROUP name="Soma">
<WS url="http://172.17.105.62:8080/axis/Ws1.jws"
name_operation="soma" tempo_execucao="1"/>
</GROUP>
<GROUP name="Subtracao">
<WS url="http:// 172.17.105.62:8080/axis/Ws2.jws"
name_operation=“subtracao" tempo_execucao="1"/>
</GROUP>
</GROUPS>
```

Figura 4.3: Configuração inicial do repositório do Somador Distribuído

Neste exemplo, um aspecto não-funcional deve ser escolhido para atuar como critério de adaptação, e foi então selecionado o critério tempo de execução, ou seja, deverá ser eleito o serviço que contiver um melhor tempo de execução.

Tem-se observado que não se constitui uma tarefa trivial mensurar a qualidade de determinados aspectos não-funcionais. Como afirmar, em termos quantitativos, por exemplo, o quão seguro um serviço é? Para solucionar esta questão, uma estratégia pode ser utilizada: a divisão em níveis. Pode-se padronizar que um determinado serviço que satisfaça um número específico de requisitos de segurança tem o nível de segurança 2. Se ele adicionalmente implementar algum protocolo de criptografia, esse nível poderia ser aumentado para 3. Esta estratégia, mesmo que de um alto nível, permite que se tenha uma noção da qualidade de um determinado aspecto não-funcional de um *web service*.

Desta forma, esta divisão por níveis será utilizada para qualificar o aspecto tempo de execução neste estudo de caso. Na Figura 4.3, onde está especificado tempo_execução="1" deve-se ler: "tempo de execução com qualidade correspondente ao nível 1". Adicionalmente, será adotado que quanto maior o nível melhor será a qualidade do serviço.

Neste estudo de caso, configurou-se a *engine* adaptativa para que a mesma imprimisse na tela o endereço do *web service* participante que seria chamado. Com isso, a adaptação poderia ser claramente exposta através de diversas execuções deste processo de negócio e com as possíveis alterações no *Web Service Repository*.

Após a inicialização do sistema, foram executadas diversas execuções do processo de negócio. Primeiramente, a Execução 1 foi realizada logo após a inicialização do sistema. Como esperado, a chamada de soma foi direcionada para WS1_v1 e a de subtração para WS1_v2. Em seguida, por diversas razões, poderia ser desejado que apenas um desses *web services* (por exemplo, WS1_v1) fosse utilizado (ele poderia ter um tempo de execução melhor do que o outro). Desta forma, foi adicionada no repositório uma entrada no grupo subtração para a operação "subtracao" deste serviço, com um nível 2 de tempo de execução. Esta alteração esta explicitada na Figura 4.4. Após esta alteração, a Execução 2 foi realizada, com a chamada sendo redirecionada com sucesso, como pode-se verificar na Tabela 4.1. Neste momento, o processo de negócio está sendo executado com apenas um *web service*. Em algum momento, o servidor remoto com os *web services* WS1_v3 e WS1_v4 poderia ser disponibilizado, e ser também desejada a utilização do serviço soma do WS1_v3. Após a inserção do mesmo no *Web Service Repository* (tempo_execucao=3), a Execução 3 foi realizada, e o sistema se adaptou, como mostra a tabela citada. Por fim, WS1_v4 poderia também ter um melhor tempo de execução para a operação "subtracao" do que o atualmente utilizado, WS1_v1. Desta forma, a sua entrada foi adicionada no repositório, com a Execução 4 sendo processada logo em seguida. Esta configuração final do repositório esta apresentada na Figura 4.5. Por fim, se por algum motivo não existir mais o desejo de se utilizar os serviços disponibilizados no servidor remoto, basta que suas entradas sejam removidas do *Web Service Repository*, retornando-se a configuração anterior a sua inserção. A Execução 5 foi então realizada para verificar se o sistema iria se adaptar a esse novo desejo, e a Tabela 4.1 apresenta a correta adequação do mesmo tendo em vista as mudanças ocorridas.

Tabela 4.1: Resultado das execuções do Somador Distribuído

	Soma	**Subtração**
Execução 1	http://172.17.105.62:8080/axis/Ws1.jws **(WS1_v1)**	http://172.17.105.62:8080/axis/Ws2.jws **(WS1_v2)**
Execução 2	http://172.17.105.62:8080/axis/Ws1.jws **(WS1_v1)**	http://172.17.105.62:8080/axis/Ws1.jws **(WS1_v1)**
Execução 3	http://172.17.115.206/axis/Ws1.jws **(WS1_v3)**	http://172.17.105.62:8080/axis/Ws1.jws **(WS1_v1)**
Execução 4	http://172.17.115.206:8080/axis/Ws1.jws **(WS1_v3)**	http://172.17.115.206:8080/axis/Ws2.jws **(WS1_v4)**
Execução 5	http://172.17.105.62:8080/axis/Ws1.jws **(WS1_v1)**	http://172.17.105.62:8080/axis/Ws1.jws **(WS1_v1)**

```
<GROUPS>
<GROUP name="Soma">
<WS url="http://172.17.105.62:8080/axis/Ws1.jws"
name_operation="soma" tempo_execucao="1"/>
</GROUP>
<GROUP name="Subtracao">
<WS url="http:// 172.17.105.62:8080/axis/Ws2.jws"
name_operation="subtracao" tempo_execucao="1"/>
<WS url="http:// 172.17.105.62:8080/axis/Ws1.jws"
name_operation="subtracao" tempo_execucao="2"/>
</GROUP>
</GROUPS>
```

Figura 4.4: Web Service Repository depois da adição da operação subtracao do *web service* WS1

```
<GROUPS>
<GROUP name="Soma">
<WS url="http://172.17.105.62:8080/axis/Ws1.jws"
name_operation="soma" tempo_execucao="1"/>
<WS url="http://172.17.115.206:8080/axis/Ws1.jws"
name_operation="soma" tempo_execucao="3"/>
</GROUP>
<GROUP name="Subtracao">
<WS url="http:// 172.17.105.62:8080/axis/Ws2.jws"
name_operation="subtracao" tempo_execucao="1"/>
<WS url="http:// 172.17.105.62:8080/axis/Ws1.jws"
name_operation="subtracao" tempo_execucao="2"/>
<WS url="http://172.17.115.206:8080/axis/Ws2.jws"
name_operation="subtracao" tempo_execucao="3"/>
</GROUP>
</GROUPS>
```

Figura 4.5: Configuração final do Web Service Repository

Neste momento, é válido destacar a facilidade de atualização do repositório através do uso do mecanismo proposto na Seção 3.3.1. Para adicionar ou remover serviços do ambiente em execução, basta adicionar uma linha dentro do arquivo XML utilizado para a descrição do *Web Service Repository*. A *engine* adaptativa se encarrega de todo o resto.

Por fim, a Figura 4.6 mostra o *log* de todas estas execuções realizadas.

Execution 1
Call: http://172.17.105.62:8080/axis/Ws1.jws
Call: http://172.17.105.62:8080/axis/Ws2.jws
Execution 2
Call: http://172.17.105.62:8080/axis/Ws1.jws
Call: http://172.17.105.62:8080/axis/Ws1.jws
Execution 3:
Call: http://172.17.115.206:8080/axis/Ws1.jws
Call: http://172.17.105.62:8080/axis/Ws1.jws
Execution 4:
Call: http://172.17.115.206:8080/axis/Ws1.jws
Call: http://172.17.115.206:8080/axis/Ws2.jws
Execution 5:
Call: http://172.17.105.62:8080/axis/Ws1.jws
Call: http://172.17.105.62:8080/axis/Ws1.jws

Figura 4.6: *Log* das execuções do Somador Distribuído

Adicionalmente, apenas a título de confirmação, os *web services* que foram trocados foram desligados, e o processo de negócio processou normalmente suas execuções. Se ainda existisse qualquer dependência, o processo não terminaria sua execução, uma mensagem de erro seria retornada.

Os resultados obtidos com este estudo de caso mostraram a eficácia da abordagem proposta, à medida que as alterações ocorridas no ambiente foram percebidas pela *engine* de maneira bastante transparente, com um mínimo de intervenção por parte dos responsáveis, e sem nenhuma necessidade de reinício de qualquer módulo do sistema.

A título de consulta, o código-fonte referente a este exemplo foi disponibilizado no Apêndice A.1. O mesmo não foi inserido neste capítulo tendo em vista não carregar muito o conteúdo do mesmo, impondo maiores dificuldades para a compreensão por parte do leitor. Neste apêndice se encontram o código da especificação WS-BPEL da composição e das interfaces dos *web services* envolvidos, tanto a do próprio processo de negócio como a dos serviços participantes. Os arquivos são: soma.bpel (especificação WS-BPEL do processo de negócio), soma.wsdl (interface WSDL deste processo de negócio), ws1.wsdl (interface WSDL do serviço ws1) e ws2.wsdl (interface WSDL do serviço ws2). Esta aplicação foi implementada tendo em vista este trabalho, e todo o código-fonte foi desenvolvido neste projeto, com o auxílio de ferramentas CASE que facilitaram determinadas etapas. O código foi disponibilizado no apêndice apenas para fins de consulta.

4.2 Sistema de Aprovação de Créditos

Será apresentado nesta seção o Sistema de Aprovação de Créditos. Esta aplicação é considerada bastante tradicional, sendo apresentada (e com a implementação disponibilizada) em [ActiveBPEL 2006] e utilizada também, por exemplo, em [Silva 2006].

Uma companhia pode estar interessada em verificar se os seus clientes podem ou não efetuar empréstimos. Para a realização desta análise, existe a figura de um assessor e de um aprovador. O primeiro trata dos casos mais simples, enquanto que as requisições de maior valor financeiro devem passar pelo segundo. Se o próprio assessor considerar que a requisição apresenta um alto risco, mesmo que ela tenha sido considerada inicialmente simples, com baixo valor financeiro, ele a repassa para o aprovador.

Em termos de sistema, dois *web services* são utilizados. Eles levam os mesmos nomes das duas figuras anteriormente apresentadas: aprovador (*Approver*) e assessor (*Assessor*). A Figura 4.7 apresenta a visão geral deste sistema.

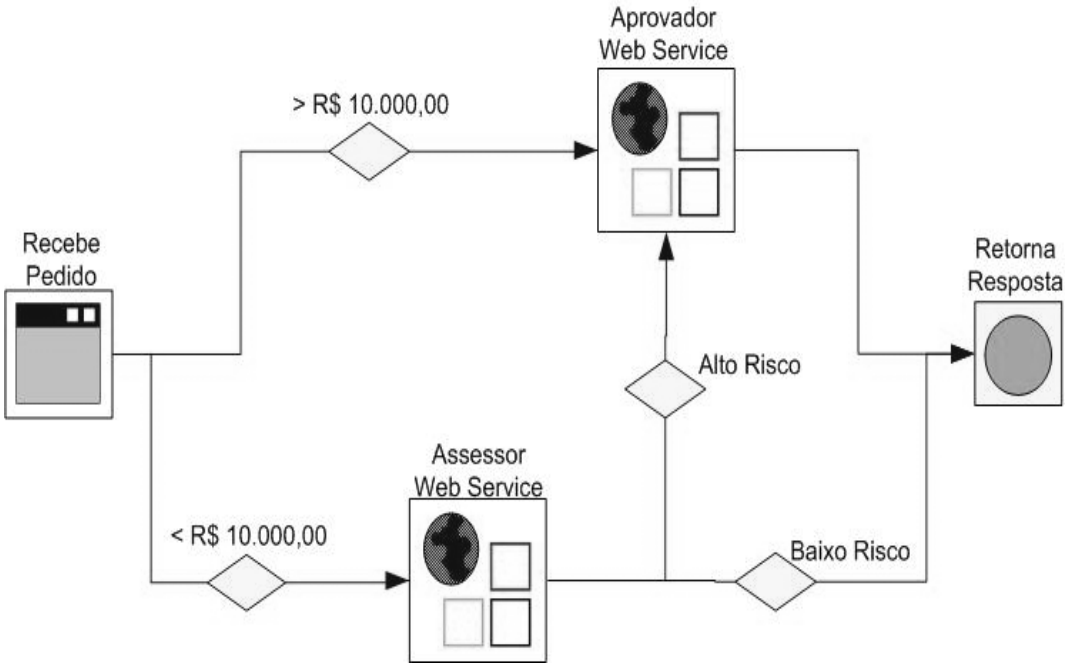

Figura 4.7: Visão geral do Sistema de Aprovação de Créditos

Como pode ser percebido, a depender tanto do valor requisitado pelo cliente e do retorno do assessor, que pode ser dizendo tanto que a requisição apresenta um alto risco como um baixo risco, será definido se a intervenção do aprovador é necessária ou não. A principal razão de se utilizar um assessor ao invés de ser disponibilizado apenas um aprovador é a de não sobrecarregar o mesmo com um número elevado de requisições. Adicionalmente, por ser um serviço que trata com casos mais simples, o assessor pode ter uma implementação menos complexa, o que resulta em um menor tempo de execução.

Em termos de composição, dois casos podem acontecer, a depender do valor de entrada: apenas um dos serviços serem executados (no caso, uma composição com escolha) ou os dois serem executados (caracterizando uma composição seqüencial).

4.2.1 Ferramentas

Todos os experimentos neste estudo de caso foram executados utilizando o servidor Web Apache-Tomcat [Apache 2006]. Dois servidores foram configurados para a avaliação da abordagem proposta neste trabalho, cada qual disponibilizando uma versão dos serviços assessor e aprovador. A única diferença entre os mesmos é que um deles tem a extensão A-ActiveBPEL instalada, provendo o ambiente necessário a execução do processo de negócio, e o outro apenas disponibiliza outra versão dos mesmos serviços, através de um servidor Web sem a *engine*/extensão instalada (no caso, o Apache-Tomcat com o módulo SOAP). Desta forma, apenas um deles está apto a executar composições de *web services*, com o outro apenas disponibilizando serviços alternativos. A Figura 4.1 apresenta, de forma compacta, os servidores utilizados, incluindo o conteúdo de cada um deles e os serviços que eles disponibilizam. Por questões didáticas, o *web service Assessor* será chamado apenas de WS1 e o *web service Approver* será denominado WS2. Como temos

duas versões de cada um, v1 e v2 significam, respectivamente, versão 1 e versão 2.

4.2.2 Configuração, Execução e Resultados

De acordo com a Figura 4.7, a composição é composta por dois *web services* (assessor e aprovador), com o tipo da composição variando entre seqüencial e com escolha, a depender do valor do empréstimo e da resposta do assessor. Além do <invoke>, atividades como <receive> e <reply> também são utilizadas.

Neste exemplo, a primitiva de invocação pode ser utilizada duas vezes na composição (uma para cada *web service*). Cada chamada externa tem duas opções de utilização de serviços: uma se encontra no próprio servidor que possui a extensão A-ActiveBPEL (WS1_v1 and WS2_v1) e a outra opção se encontra no outro servidor, que apenas disponibiliza serviços (WS1_v2 e WS2_v2), conforme disposto na Figura 4.1.

Inicialmente, apenas o primeiro servidor é utilizado, com o segundo sendo adicionado ao *Web Service Repository* em tempo de execução. Cada *web service* tem a sua própria política, especificando as suas propriedades não-funcionais. Estas políticas foram armazenadas diretamente no repositório, segundo a estratégia proposta na Seção 3.3. O critério de adaptação escolhido para este estudo de caso é o "último disponibilizado" (ou seja, eleger sempre a versão mais nova do serviço). Baseado neste critério, o atributo *version* é inserido na política dos *web services* com o intuito de subsidiar a escolha que será realizada pela *engine*.

Depois de configurada e iniciada, A-ActiveBPEL analisa os arquivos da composição e depois a executa. Como mencionado anteriormente, inicialmente apenas os serviços WS1_v1 e WS2_v1 estão disponíveis no *Web Service Repository*. Depois, mudanças são executadas no mesmo com o intuito de adicionar os serviços WS1_v2 e WS2_v2, disponíveis no servidor remoto. Para isto, o arquivo do repositório foi alterado, com a adição dos mesmos (incluindo endereço e operações) em seus respectivos grupos. Além disto, estes *web services* foram armazenados no *Web Service Repository* com indicação de que a versão deles é a mais recente (através do atributo *version*). A Figura 4.8 apresenta a configuração do repositório depois destas alterações. É conveniente lembrar que o endereço IP do servidor local é 172.17.105.62 e o endereço IP do servidor remoto é 172.17.115.206.

Maiores detalhes sobre esta especificação podem ser encontrados na Seção 3.3.

```
<GROUPS>
<GROUP name="Approver">
<WS url="http://172.17.105.62:8080/services/ApproverWebService"
name_operation="approve" version="1"/>
<WS
url="http://172.17.115.206:8080//services/ApproverWebService"
name_operation="approve" version="2"/>
</GROUP>
<GROUP name="Assessor">
<WS url="http:// 172.17.105.62:8080/services/AssessorWebService"
name_operation="check" version="1"/>
<WS url="http://172.17.115.206:8080/services/AssessorWebService"
name_operation="check" version="2"/>
</GROUP>
```

Figura 4.8: Configuração do Web Service Repository com todos os serviços especificados

Com o intuito de visualizar a adaptação, a *engine* adaptativa foi configurada para imprimir os endereços IP das chamadas que são realizadas, o que permite que seja analisada cada invocação realizada. Algumas execuções foram realizadas para avaliar a abordagem proposta neste trabalho. Por razões didáticas, os testes que levam a composições seqüenciais e composições com escolha foram separados. Para o primeiro caso, as requisições de empréstimo continham valores que exigiam a utilização dos dois *web services*, ou seja, levavam a uma composição seqüencial. A Tabela 4.2 expõe os resultados obtidos com as execuções das composições seqüenciais.

Tabela 4.2: Resultado das execuções das composições sequenciais

	Web service Assessor	*Web Service Approver*
Execução 1	http://172.17.105.62:8080/services/AssessorWebService **(WS1_v1)**	http://172.17.105.62:8080/services/ApproverWebService **(WS2_v1)**
Execução 2	http://172.17.115.206:8080/services/AssessorWebService **(WS1_v2)**	http://172.17.115.206:8080/services/ApproverWebService **(WS2_v2)**
Execução 3	http://172.17.105.62:8080/services/AssessorWebService **(WS1_v1)**	http://172.17.115.206:8080/services/ApproverWebService **(WS2_v2)**

Primeiramente, a Execução 1 foi realizada imediatamente após a inicialização do sistema. Depois disto, o *Web Service Repository* foi atualizado de tal forma que os *web services* do servidor remoto foram incluídos no sistema adaptativo (Figura 4.8). Posteriormente, a Execução 2 foi feita e, como pode-se observar na tabela, as chamadas foram redirecionadas para os *web services* que melhor satisfizeram o critério adotado (no caso, os disponibilizados pelo servidor remoto). O sistema se adaptou as novas mudanças, e sem necessidade de reinicializações. Um outro cenário possível se baseia em que os *web services* já estejam especificados no repositório, e apenas as suas políticas sejam alteradas (mais especificamente, alguma de suas propriedades não-funcionais, no caso o *version*), com o intuito de se verificar essas mudanças em tempo de execução. Para avaliar esta situação, na Execução 3 o repositório foi modificado e o *web service* WS1_v1 se tornou o de versão mais recente (version="3"). Como se pode observar na tabela, o sistema novamente se adaptou como desejado.

A Figura 4.9 apresenta o log dessas execuções e também demonstra as mudanças ocorridas nas invocações dos *web services* participantes. Em cada execução, ambos (*Assessor* e *Approver*) são invocados. Através da troca dos locais (endereços) das chamadas, pode-se notar claramente a adaptação sendo realizada.

Execution 1
Call: http://172.17.105.62:8080/services/AssessorWebService
Call: http://172.17.105.62:8080/services/ApproverWebService
Execution 2
Call: http://172.17.115.206:8080/services/AssessorWebService
Call: http://172.17.115.206:8080/services/ApproverWebService
Execution 3:
Call: http://172.17.105.62:8080/services/AssessorWebService
Call: http://172.17.115.206:8080/services/ApproverWebService
Execution 4:
Call: http://172.17.115.206:8080/services/ApproverWebService
Execution 5:
http://172.17.105.62:8080/services/ApproverWebService
Execution 6:
http://172.17.105.62:8080/services/AssessorWebService

Figura 4.9: Log da execução das composições seqüenciais

Com o intuito de se analizar a composição com escolha, outras execuções (Execuções 4, 5 e 6 da Figura 4.9) foram realizadas com valores para as requisições de empréstimo que levava a este tipo de composição (Tabela 4.3). Desta forma, apenas um serviço na composição (*Approver ou Assessor*) é executado, e a escolha do mesmo depende dos valores passados pela requisição. Inicialmente, serão utilizadas requisições com valores superiores a U$ 10.000,00, e apenas o *web service Approver* será requisitado neste caso (Execução 4). Para avaliar a adaptação, e ainda com o sistema em execução, a versão do WS2_v1 é alterada para 3 (version="3"), fazendo com que o mesmo tenha a versão mais recente no ambiente. A Execução 5 foi feita com o objetivo de testar esta alteração. Posteriormente, a Execução 6 foi realizada utilizando

valores de empréstimo menores do que $10.000, considerando-se inclusive que o risco da operação era baixo, especificando também que WS1_v1 agora tem versão 3 (*version*="3"). A *engine* adaptativa percebe esta mudança e redireciona a chamada para o serviço correspondente no servidor local. Existiram, a nível de teste, diversas outras execuções, contudo considerou-se que as aqui apresentadas já davam uma visualização da adaptação.

Tabela 4.3: Resultado das execuções das composições com escolha

	Web Service Assessor	Web Service Approver
Execução 4	X	http://172.17.115.206:8080/services/ApproverWebService **(WS2_v2)**
Execução 5	X	http://172.17.105.62:8080/services/ApproverWebService **(WS2_v1)**
Execução 6	http://172.17.105.62:8080/services/AssessorWebService **(WS1_v1)**	X

Adicionalmente, para evidenciar de uma outra forma que realmente houve a adaptação, o *web service* substituído nas adaptações foi desligado, e a composição continuou funcionando normalmente, o que comprova que a chamada estava sendo corretamente redirecionada (caso contrário, uma mensagem de erro seria retornada afirmando que foi impossível encontrar o serviço requisitado).

A aplicação apresentada nesta seção vem sendo utilizada em diversos trabalhos de composição de *web services* [ActiveBPEL 2006] [Silva 2006], mostrando sua relevância nesta área da computação. Neste trabalho, ela foi utilizada para demonstrar, de uma maneira clara, a viabilidade da proposta deste trabalho. Aplicações comerciais e diversos *benchmarks* poderiam ser testados, contudo a abordagem proposta independe da complexidade da implementação dos *web services* participantes. É razoável pensar que outras aplicações com um largo número de *web services* ou apresentando serviços mais complexos (em termos de implementação) sofrerão as adaptações em tempo de execução, como esperado.

A título de consulta, o código-fonte referente a esta aplicação foi disponibilizado no Apêndice A.2. Nele se encontram o código da especificação BPEL da composição e as interfaces dos *web services* envolvidos (tanto o do próprio processo de negócio como o dos serviços participantes). Os respectivos arquivos são: loanapproval.bpel (especificação do processo de negócio), loanapproval.wsdl (interface do processo de negócio), loanassessor.wsdl (interface do serviço Assessor) e loanapprover.wsdl (interface do serviço approver). Outros arquivos de configuração são necessários ainda para a execução do mesmo, contudo não são importantes para subsidiar o processo de

entendimento deste exemplo no presente trabalho. Eles podem ser encontrados em [ActiveBPEL 2006]. É importante frisar que esta aplicação não foi codificada pelo autor deste trabalho, apenas sendo utilizada como exemplo para a utilização da abordagem proposta. Todo o código-fonte da mesma é de propriedade da empresa Active Endpoints, e todos os créditos são relacionados a esta empresa. O código foi dispobilizado no apêndice apenas para fins de consulta.

4.3 Considerações Finais

Este capítulo apresentou os estudos de caso que validaram a proposta deste trabalho. Dois exemplos foram utilizados, um de larga utilização em diversos trabalhos e outro projetado especificamente para este, com cunho mais didático.

Ambos os estudos de caso mostraram que a abordagem proposta permite a adaptação da composição de *web services* em tempo real, sem a necessidade de reinicialização do sistema e requisitando uma mínima intervenção humana, que é necessária apenas para disponibilizar as informações sobre os novos *web services* em um repositório central. Como será visto na próxima seção, outros trabalhos também apresentam propostas para disponibilizar adaptabilidade no contexto da composição de *web services*, contudo ou eles não respeitam padrões pré-estabelecidos e largamente difundidos (uma das premissas deste trabalho) ou requisita uma intervenção maior por parte do implementador.

Pode-se observar claramente que em momento nenhum o código original da aplicação foi alterado. Esse fato apresenta uma elevada importância, à medida que o programador da aplicação não precisa pensar em como tratar a questão da adaptabilidade, ela já é fornecida automaticamente pelo ambiente (mais especificamente, pela *engine* adaptativa).

A grande questão envolvida na análise dos estudos de caso aqui apresentados é verificar que a *engine* adaptativa conseguiu olhar para o repositório, verificar que existiram mudanças e efetivamente trabalhou para que as mesmas fossem implementadas em tempo de execução.

Como já frisado neste capítulo, a complexidade dos estudos de caso não são fatores determinantes neste trabalho; a capacidade de escolher e mudar a chamada de um *web service* não sofre influência direta da sua complexidade de implementação. Em outras palavras, para avaliar a capacidade adaptativa da *engine*, não importa a complexidade dos serviços participantes, pois tudo o que a *engine* tem a como missão é olhar a interface dos mesmos em conjunto com as suas políticas previamente disponibilizadas e invocar a operação desejada, não importando o que os mesmos façam para retornar o resultado esperado.

CAPÍTULO

5

Trabalhos Relacionados

"Numa discussão, o mais difícil não é defender um ponto de vista, mas entendê-lo."

Autor desconhecido.

Neste capítulo são apresentados os trabalhos relacionados ao tema desta dissertação. Em especial, na área de composição adaptativa de *web services*, duas correntes vemtendo elevado destaque: utilização de orientação a aspectos e alteração sintática no padrão WS-BPEL. Ambas também alcançaram resultados significativos, tendo vantagens e desvantagens a serem apresentadas e discutidas. Adicionalmente, uma visão crítica e comparativa dos mesmos com este trabalho será apresentada.

5.1 Alteração Sintática do Padrão WS-BPEL

Karastoyanova [Karastoyanova 2005] apresenta uma proposta para disponibilizar a adaptação de *web services* compostos em tempo de execução. Esta proposta é baseada na proposição de uma extensão ao padrão WS-BPEL, que implementa um mecanismo adaptativo denominado "*find and bind*". Este mecanismo facilita o controle preciso da seleção de *web services* em tempo de execução e também provê reparo da instância de processos de uma forma simples.

A motivação para esta extensão à WS-BPEL se assemelha em parte a este trabalho: procurar componentes e fazer a ligação dos mesmos em tempo de execução. Esta técnica já foi foi aplicada em diversas tecnologias de *middleware* [Alonso 2003]. A grande questão envolvida é que não existe um suporte disto na especificação padrão de WS-BPEL. Na verdade, a escolha dos serviços é efetuada ainda no próprio código-fonte da aplicação, e esta escolha é estática, sem proporcionar a possibilidade de mudanças durante a execução do processo de negócio.

Em outras palavras, a especificação corrente de WS-BPEL (incluindo as *engines* existentes) não trata a questão da adaptabilidade em instâncias de processos de negócio de forma efetiva. Neste contexto, o mecanismo "*find and bind*" foi proposto.

5.1.1 Mecanismo "*Find and Bind*"

O mecanismo proposto por Karastoyanova é composto, basicamente, por três passos, a saber:

I. Localizar todos os *web services* disponíveis que são compatíveis com o serviço desejado (pode-se utilizar o repositório UDDI para este intento);

II. Selecionar um deles através de um critério pré-escolhido (por exemplo, segurança);

III. Fazer a ligação do *web service* participante eleito com a instância do processo de negócio. Para a execução corrente, este será o *web service* que executará o serviço desejado;

A Figura 5.1 apresenta o esquema gerar deste mecanismo.

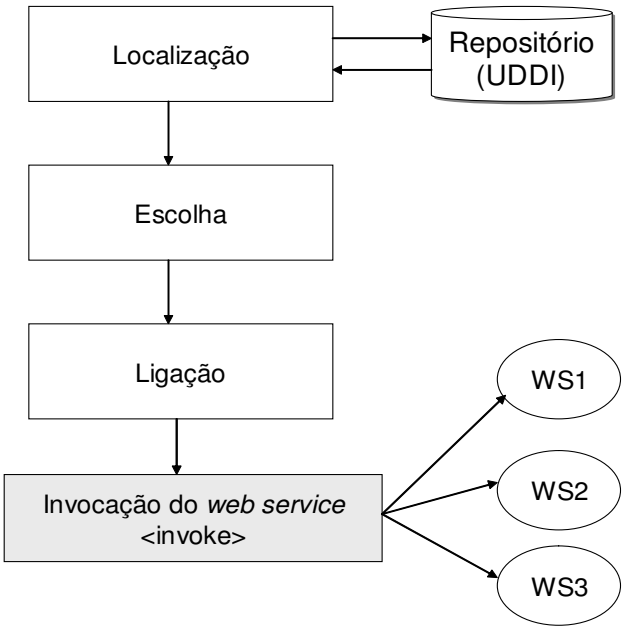

Figura 5.1: Esquema geral do mecanismo "Find and Bind"

Uma observação em especial deve ser feita: a utilização do repositório UDDI neste mecanismo é opcional. Embora o autor sugira a utilização do mesmo, ela não foi implementada no trabalho. Ele utilizou um mecanismo mais simples, baseado em arquivos, o que pode ser benéfico em casos mais simples, contudo empobrece uma especificação mais geral e abrangente da sua proposta. A grande questão envolvida é que o sistema deve ser capaz de interagir com o UDDI e ainda fazer o mapeamento do conteúdo deste repositório no formato

interno utilizado pelo sistema, fato complicador para a implementação do projeto.

Para implementar este mecanismo foi proposta uma alteração sintática no WS-BPEL, com a adição de uma *tag* específica, denominada <*find_bind*>. Ou seja, o programador deve, explicitamente, informar em que ponto pode ocorrer a troca de *web services* participantes e quais os candidatos a serem eleitos pela *engine* através do critério escolhido.

Inicialmente, os programadores não têm a necessidade de se preocupar como o mecanismo é executado e nem como ele pode influir no restante da especificação WS-BPEL do projeto (pois ele não tem relação direta com nenhuma outra *tag* desta especificação, com exceção do <*invoke*>, que é a *tag* de invocação propriamente dita). Eles devem se preocupar primordialmente com o critério de escolha e com a especificação desta nova *tag*.

Através da especificação da *tag* <*find_bind*>, pode-se expressar os requisitos do processo de negócio baseado na seleção de políticas. É permitido ao programador a escolha de valores padrões para a seleção de critérios em substituição a dados durante a implementação do processo (no caso de possíveis modificações reparadoras).

Na prática, as interações dos processos especificados no padrão WS-BPEL com *web services* externos são realizadas através da primitiva <invoke>, motivo pelo qual as *tags* <*find_bind*> se relacionarão diretamente com esta primitiva. Um exemplo deste relacionamento é dado a seguir:

```
(1) <process name=”nome_processo”>
(2) <!--  especificação inicial do processo  -->
(3) <invoke
(4) name= ”requisita_servico”
(5) partnerLink= “web_service_parceiro”
(6) portType = “servico_desejado”
(7) operation = “operacao”
(8) inputVariable = “entrada”
(9) outputVariable = “resultado”>
(10) <find_bind selection_policy = “ID_politica”/>
(11) </invoke>
(12) <!-- especificação final do processo -->
(13) </process>
```

Toda a parte referente ao código do mecanismo foi destacado, tendo em vista explicitar que partes do código não fazem parte da especificação oficial de WS-BPEL.

O atributo `ID_politica` pode tanto especificar a seleção de políticas em termos de uma lista de critérios ou pode indicar o nome de uma política de seleção de serviços participantes diretamente.

A utilização deste mecanismo no caso de reparo de instâncias de processos, mais especificamente quando da indisponibilidade de serviços necessários e requisitados, é também tratado, de forma automática. Se em um dado momento da execução do processo de negócio um *web service* participante da composição está inoperante, o mecanismo "*find and bind*" é capaz de avaliar uma alternativa para o mesmo, caso exista e tenha sido especificada anteriormente. Contudo, se não existirem mais opções especificadas na política adotada na *tag* <*find_bind*>, um reparo manual é requisitado, acarretando a conseqüente paralização do sistema (que é indesejada).

Maiores detalhes sobre a implementação deste mecanismo (em especial, em termos de codificação), podem ser obtidos em [Karastoyanova 2005].

5.1.2 Avaliação Crítica

O trabalho de Karastoyanova [Karastoyanova 2005] apresentou uma proposta relevante para a adaptação de composições de *web services*. Através da utilização do mecanismo "*find and bind*", programadores podem criar alternativas para a chamada de serviços participantes, sendo eleitos aqueles que melhores satisfizerem determinados critérios definidos por eles mesmos.

O trabalho de Karastoyanova apresenta algumas semelhanças com a proposta desta dissertação. Em princípio, a disponibilização de informações sobre *web services* em um repositório para análise é realizada nos dois trabalhos (mesmo que de forma distinta). A idéia de, em tempo de execução, verificar qual destes *web services* disponibilizados melhor satisfaz determinados critérios também é ponto comum. Por fim, a proposição de critérios baseados em aspectos não-funcionais (como, por exemplo, QoS e segurança), não se excluindo outras possibilidades, é proposta em ambos os trabalhos.

Contudo, a proposta de Karastoyanova propõe uma alteração sintática em um padrão já amplamente difundido e aceito. É sabido que modificar padrões já estabelecidos pode levar a inúmeros problemas, dificultando a proliferação da própria proposta. Pelo fato de esta proposta não ter sido incorporada pelas diversas *engines* existentes no mercado, o programador deve utililizar a disponibilizada no contexto do trabalho, o que pode trazer conseqüências indesejáveis e até limitações na utilização efetiva do projeto (a especificação de composição de serviços é dependente da engine utilizada para este fim). Adicionalmente, a especificação da adaptação ocorre de forma explícita no próprio código-fonte da aplicação, onde fica a cargo do programador, no desenvolvimento do projeto, afirmar onde poderá ocorrer adaptação (apenas as chamadas que tiverem a *tag* <*find_bind*> associada serão alvo de adaptação) e quais as opções disponíveis para a execução de determinados serviços.

Adicionalmente, pelo fato da especificação da adaptação ser no próprio código-fonte da aplicação, o nível de adaptabilidade fica prejudicado, na medida em que todos os *web services* candidatos devem ser explicitados neste momento. Se, em tempo de execução, surgirem novas opções, não será possível a adição das mesmas no sistema sem a parada do mesmo para a alteração de sua especificação.

A abordagem proposta neste trabalho, em comparação com o trabalho de Karastoyanova, propõe que a adaptação seja responsabilidade da *engine*,

preservando o padrão WS-BPEL, já bastante difundido e aceito. Uma vantagem direta decorrente deste fato é que o código-fonte da aplicação permanece intacto, sem necessidade de interação do programador no mesmo. Todas as chamadas a *web services* participantes podem ser alvo de adaptação, mesmo que isso não tenha sido inicialmente pensado quando do desenvolvimento da aplicação. Basta atualizar o repositório central (Seção 3.3), que a própria *engine* se encarrega de eleger o serviço mais adequado (segundo os critérios definidos pelo programador) para a composição.

Finalmente, uma outra diferença crucial é que a abordagem proposta neste trabalho possibilita a adição de novos *web services* em tempo de execução. Por propor que a adaptação seja descrita no próprio código-fonte da aplicação, Karastoyanova não permite que novos serviços sejam adicionados em tempo de execução, sendo necessário a parada do sistema e a reescrita de código. Já no caso da proposta desta dissertação, basta atualizar o *Web Service Repository* (inserindo os novos serviços) que, mesmo em tempo de execução, a *engine* adaptativa será capaz de perceber a mudança e agir com o intuito de adequar a composição tendo em vista as novas possibilidades. Inclusive, como descrito anteriormente, mesmo que uma determinada instância do processo de negócio esteja sendo executada, ela ainda poderá ser adaptada, no caso do serviço a ser substituído ainda não ter sido executado.

5.2 Adaptabilidade & Orientação a Aspectos

Existe uma forte corrente atualmente na área de composição de *web services* que vêm propondo a utilização de orientação a aspectos [Charfi 2004] [Erradi 2005] [McKinley 2004]. Este paradigma permite que a especificação de propriedades gerais do sistema seja separada da especificação de sua funcionalidade. Pode-se afirmar que a programação orientada a aspectos (POA) apresenta um impacto relativo maior na adaptação não-funcional, por oferecer linguagens para expressão de aspectos não-funcionais do sistema, tais como concorrência, distribuição e tratamento de exceções [Charfi 2004]. Esta especificação separada pode facilitar a localização e adaptação destes aspectos não-funcionais do projeto do sistema. A grande idéia neste contexto é tratar a adaptabilidade como um aspecto, especificado a parte do código funcional. Desta forma, poderá ser obtida a desejada adaptabilidade, pois aspectos podem ser ativados/desativados em tempo de execução.

A programação orientada a aspectos introduz unidades de modularidade chamadas aspectos. Um aspecto pode conter fragmentos de código (chamados, na literatura, de *advice*) e descritores de localização (*pointcuts*) que informam onde colocar esses fragmentos. Pode-se afirmar que um *advice* é definido de forma similar a um método. A diferença é que ele nunca é invocado explicitamente, apenas quando um *pointcut* relacionado a ele assume o valor booleano *TRUE*. Por sua vez, os chamados *join points* representam um determinado caminho que pode ser tomado dentro do sistema, a depender dos valores de seus *pointcuts*. Um sistema pode ou não ser adaptativo: um determinado *pointcut* poderia determinar a utilização ou não de um determinado fragmento de código (*advice*), que seria responsável por promover a adaptabilidade. A integração de todos estes elementos é chamada de *weaving* (entrelaçamento). Este entrelaçamento pode ocorrer em tempo de execução, o que acaba permitindo o que já foi aqui anteriormente dito: a adaptação poderia

inclusive ser ativada/desativada em tempo de execução, a depender das condições do momento.

Dois trabalhos, em especial, merecem destaque neste "entrelaçamento" entre programação orientada a aspectos e composição de *web services*: [Charfi 2004] e [Erradi 2005].

5.2.1 AdaptiveBPEL: Um *Framework* Para Composição Adaptativa de *Web Services*

A intenção inicial do projeto AdaptiveBPEL [Erradi 2005] é a de experimentar um novo modelo, baseado nos mecanismo orientados a aspectos, com a intenção de:

- Permitir a customização tanto de características funcionais (como, por exemplo, a adição de uma outra operação) como as não-funcionais (por exemplo, inserção de protocolos de autenticação tendo em vista o incremento do nível de segurança) dos serviços utilizados;
- Possibilitar a mudança nas regras do processo de negócio e das políticas que são utilizadas pelo mesmo (em especial, mas não apenas, troca de *web services* participantes); e
- Prover diferentes níveis de qualidade de serviço (em termos, por exemplo, de disponibilidade e tempo de resposta).

Segundo Erradi, este conjunto de intenções poderá permitir que um *web service* acesse um determinado processo de negócio que seja dinamicamente reconfigurável e adaptável de acordo com o perfil e as necessidades do cliente e do ambiente.

No AdaptiveBPEL, o entrelaçamento de aspectos em uma composição de serviços pode se feita tanto estaticamente, através de um descritor de implementação, ou dinamicamente, a partir de uma política negociada em tempo de execução.

Tendo em vista a possibilidade de interação entre serviços, as políticas associadas aos mesmos (que podem expressar tanto capacidades como requisitos) devem ser conciliadas. O casamento destas políticas pode acontecer tanto em uma fase de desenvolvimento como em tempo de execução, especialmente quando as próprias políticas são objeto de mudanças. Para a especificação de tais políticas, o autor propõe a utilização do Ws-Policy [Bajaj 2006], um formalismo proposto para a descrição de políticas de *web services*.

A Figura 5.2 apresenta o esquema geral da proposta de Erradi, baseada nas ilustrações presentes em seu artigo.

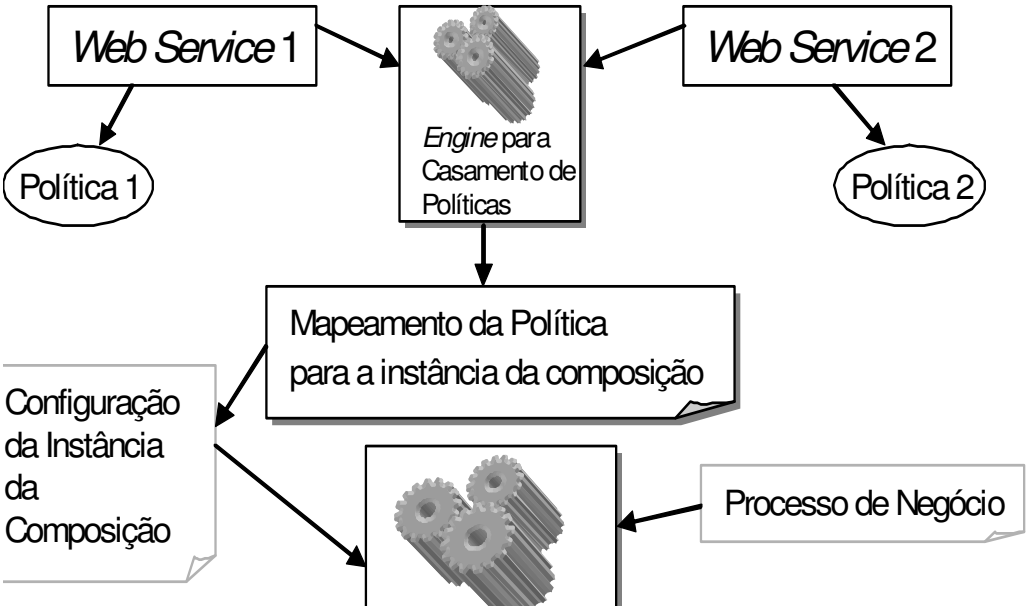

Figura 5.2: Esquema geral da proposta de Erradi

É importante frisar que este esquema apresentado por Erradi é apenas conceitual, não há implementação. Cada *web service* presente deve conter a sua política associada, a qual será alvo de análise pela *engine* responsável pelo casamento destas políticas (objetivando eleger que serviços serão utilizados, baseado na qualidade de serviço desejada). Após esta etapa, um mapeamento entre a política adotada resultante e a instância propriamente dita da composição se faz necessário. Em relação à "configuração da instância da composição", detalhes como a configuração dinâmica do entrelaçamento dos aspectos e possíveis configurações necessárias à chamada dos *web services* participantes, como a configuração das mensagens SOAP para a execução remota dos procedimentos são tratados. Por fim, a *engine* BPEL estendida (apta a lidar com a questão de aspectos, incluindo entrelaçamento de aspectos – *aspect weaving*) executa o processo de negócio.

Como pode ser observado, a utilização de aspectos no contexto de adaptabilidade em composição de *web services* se mostra promissora, promovendo inclusive mudanças na própria lógica do negócio. Este tipo de promessa estrapola os limites apresentados por [Karastoyanova 2005] e pelo trabalho aqui apresentado, que se preocupam especificamente na possibilidade de mudança dos *web services* participantes da composição. Erradi propõe inclusive que os próprios serviços possam ser alvos de mudanças, tudo isto em tempo de execução.

Obviamente, um número tão elevado de requisitos e promessas inferem em uma complexidade de implementação relevante, o que de certa forma

dificultou (ou até mesmo impossibilitou) a implementação deste ambiente. A Seção 5.2.3 irá tratar este aspecto com mais profundidade.

5.2.2 Composição de *Web Services* Orientada a Aspectos: o Projeto AO4BPEL

Charfi [Charfi 2004] também propõe a utilização de orientação a aspectos para a composição de *web services*. O principal argumento é que utilizando aspectos pode-se capturar a composição de *web services* de uma maneira mais modular, tornando a composição mais aberta para mudanças dinâmicas.

Segundo o autor, o padrão WS-BPEL, embora bastante difundido e aceito, apresenta fragilidade em relação a duas questões:

1) Falta de modularidade para a modelagem de separação de interesses (*crosscutting concerns*). Em outras palavras, o próprio processo de negócio é fracamente modularizável, o que dificulta sua manutenção e reusabilidade;

2) Suporte inadequado para a realização de mudanças na composição em tempo de execução (questão já tratada neste trabalho). Quando um processo de negócio especificado em WS-BPEL é desenvolvido, as interfaces WSDL de todos os arquivos participantes deve ser conhecida. Adicionalmente, uma vez que o processo tenha sido implementado, não existe nenhuma maneira de modificá-lo dinamicamente (exceção feita ao mecanismo *dynamic partner binding* [Andrews 2006], que contudo não implementa adaptações em tempo de execução [Charfi 2004]). Esta visão estática é uma herança dos tradicionais sistemas *workflow*, dos quais o modelo de composição de *web services* orientado a processos emergiu. A única forma de implementar mudanças no WS-BPEL, conforme visto anteriormente neste trabalho, é parando o sistema, modificando a especificação da composição e reiniciando o mesmo. Por diversas questões apontadas por Charfi e inclusive colocadas neste trabalho, como perda de clientes, esta forma não constitui a melhor solução para este problema.

Tendo em vista este contexto, Charfi propôs o projeto AO4BPEL, uma extensão orientada a aspectos para o padrão WS-BPEL, na qual aspectos podem ser colocados (ou retirados) dentro da composição em tempo de execução.

Baseado na idéia de que um processo de negócio em BPEL é constituído de um conjunto de atividades, e levando-se em consideração que pode-se considerar pontos bem definidos na execução dos processos como *join points*, cada atividade BPEL se torna um possível *join point*. Os próprios atributos de um processo de negócio (ou uma determinada atividade) podem ser utilizados como base para a escolha de *join points* relevantes. *Pointcuts* são os meios através dos quais essa escolha dos *join points* é realizada. É prudente lembrar que os *pointcuts* é que determinam ou não a utilização desses fragmentos de código (chamados de *advices*).

Desta forma, Charfi propõe a utilização também uma alteração sintática na especificação WS-BPEL, inserindo a *tag* <*pointcut*>, responsável por selecionar as atividades onde a execução de funcionalidades específicas será integrada.

Baseado no AspectJ [Kiczales 2001], AO4BPEL suporta três "tipos" de *advice*: antes, depois e substitutiva. Um *advice*, no contexto de WS-BPEL, segundo Charfi, é uma atividade que deve ser executada antes, depois ou em substituição a uma outra atividade (inclusive uma atividade vazia, que não executa nada). Com a utilização destes tipos, é possibilitado um maior controle sobre a execução do processo baseado em aspectos.

Charfi, no contexto do seu projeto, implementou uma *engine* baseada em aspectos. A Figura 5.3 apresenta a arquitetura geral da mesma.

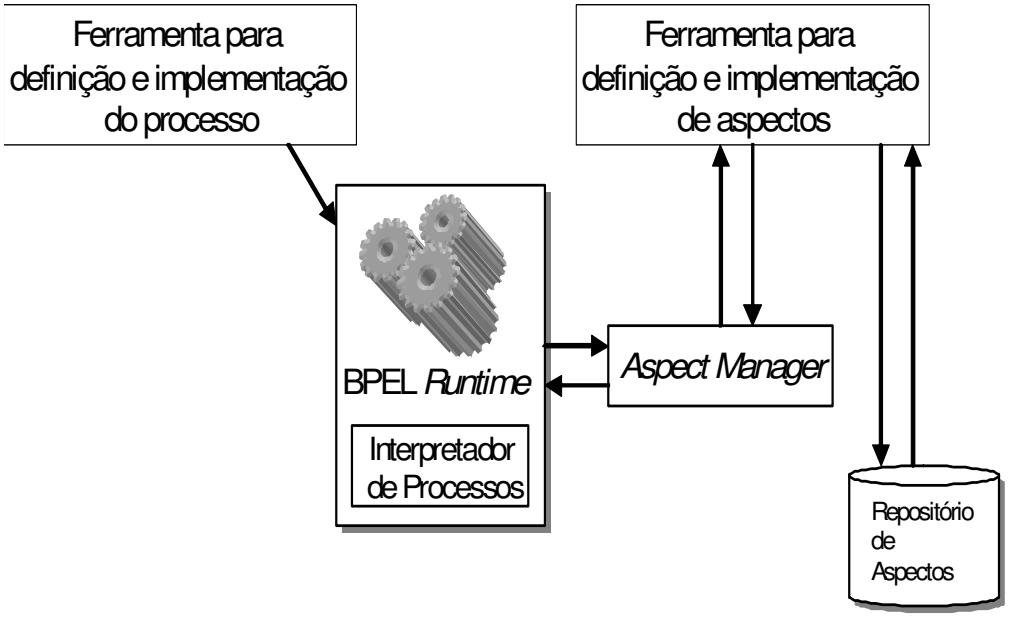

Figura 5.3: Arquitetura geral da *engine* do Projeto AO4BPEL

Desta arquitetura, os principais componentes são o BPEL *Runtime* e o *Aspect Manager*. O BPEL *Runtime* é um interpretador de processos estendido (com suporte as novas *tags* adicionadas em razão da adição de aspectos no projeto). Ele gerencia as instâncias dos processos e os aspectos. O *Aspect Manager* controla a execução do aspecto. Adicionalmente, a ferramenta de definição e implementação de aspectos gerencia o registro e a ativação/desativação de aspectos. Ela utiliza um repositório de aspectos, onde já estariam armazenados estes aspectos previamente implementados.

O princípio de funcionamento desta *engine* é baseado na checagem da existência de algum aspecto antes ou depois da interpretação de cada atividade. Em caso positivo, o *Aspect Manager* executa o *advice* correspondente e retorna o controle para o interpretador de processos. O *overhead* causado é comparativamente muito inferior comparado com o custo de se interagir com *web services* externos, o que é aceitável.

A seguir mostramos como a sintaxe da especificação BPEL é alterada com a utilização do projeto AO4BPEL em um exemplo que consiste em contar

quantas vezes a operação *searchFlight* foi invocada. Todo o código referente à adoção de aspectos no projeto será destacada (ou seja, todo o código que não faz parte da especificação oficial do padrão WS-BPEL). Maiores detalhes sobre este exemplo (inclusive sobre a sintaxe e semântica do mesmo) podem ser encontrados em [Charfi 2004].

```
(1) <aspect name="Counting">
(2) <partnerLinks>
(3)   <partnerLink name="ExecWSLink" .../>
(4) </partnerLinks>
(5) <variables>
(7)   <variable name="invocaMetodoRequisitado" .../>
(8) </variables>
(9) <pointcutandadvice type="after">
(10)  <pointcut name="Invocacoes">
(11)  //process//invoke[@portType ="InvocacoesPT" and @operation=
(12)  "searchFlight"]
(13)  </pointcut>
(14)  <advice>
(15)   <sequence>
(16)    <assign>
(17)     <copy>
(18)     <from>aumentaContador</from>
(19)     <to variable= "invocaMetodoRequisitado" part="nomeMet"/>
(20)     </copy>
(21)    </assign>
(22)    <invoke partnerLink="ExecWSLink" portType="ExecPT"
(23)    operation="metodoInvocado"
(24)    inputVariable="invokeMethodRequest"/>
(25)   </sequence>
(26)  </advice>
(27) </pointcutandadvice>
(28)</aspect>
```

Como se pode perceber claramente, o código-fonte da aplicação original foi bastante alterado, comparativamente até mais do que a proposta do mecanismo "*find and bind*" de Karastoyanova (Seção 5.1.1). Além de propor modificações em um padrão já estabelecido e difundido, a proposta de Charfi obriga que a equipe envolvida no projeto tenha conhecimentos suficientes sobre POA e sobre a sintaxe de sua proposta. O código-fonte original da aplicação foi alterado, e o nível de transparência da proposta de Charfi fica prejudicado pelo

fato de tantos conhecimentos e alterações requisitadas para que a adaptação seja realizada.

Por fugir aos objetivos deste trabalho, a especificação detalhada da implementação do projeto AO4BPEL foi omitida. A mesma pode ser encontrada em [Charfi 2004].

5.2.3 Avaliação Crítica

De início, a proposição de orientação a aspectos para a composição adaptativa de *web services* se mostra promissora. Diversos autores [Bockisch 2004] [Pawlak 2001] [Charfi 2004] reconhecem a programação orientada a aspectos dinâmica como uma técnica poderosa para a adaptação dinâmica de programas. A proposta de Charfi deixa claro que o uso de tal técnica pode viabilizar não apenas a adaptação, mas inclusive a inserção de diversos outras questões neste contexto, como segurança [Charfi 2005].

Embora fundamentadas no mesmo princípio (orientação a aspectos), as propostas de Erradi (AdaptiveBPEL) e Charfi (AO4BPEL) diferem em alguns quesitos. Erradi adiciona a adaptação orientada à políticas e seleção de aspectos para possibilitar customização de serviços específicas para o cliente (segundo o próprio autor). Contudo, Erradi propõe muita coisa, mas não tem implementação nenhuma em seu trabalho. Já Charfi, apresenta uma proposta relativamente mais simples, mais objetiva, e por isso encontrou mais facilidades na implementação de sua proposta.

De forma mais objetiva, as propostas nesta área apresentam deficiências/dificuldades aparentes. Primeiramente, o trabalho de Erradi, embora conceitualmente bem formado e abrangente, incluindo possibilidade de alteração até na lógica do próprio processo de negócio, tem deficiência na questão prática. Como o próprio autor afirma, o seu trabalho foi apenas para "motivar e contextualizar o *framework*", deixando a implementação do mesmo como trabalho futuro. Não existe atualmente nenhuma implementação de *engine* que dê suporte a sua proposta, nem mesmo exemplos ilustrativos de códigos de processos ou configurações que mostrem como funcionaria tal *framework*. É fato que a mesma pode servir como base para possíveis implementações (tanto do próprio autor como de outros pesquisadores). Contudo, a dificuldade da implementação de tal proposta é imensa, pois a mesma altera diversos protocolos e conceitos atuais. Exemplos de dificuldades a serem enfrentadas pelo autor: casamento de políticas através do Ws-Policy, mapeamento de políticas para o formato interno da *engine*, configuração da instância da composição misturando aspectos com criação dinâmica de mensagens SOAP, introdução de suporte a aspectos em alguma *engine* existente (ou então fazer uma própria que disponibilize esse suporte), dentre diversas outras (ver Figura 5.2 para ter noção da complexidade de tal implementação). Adicionalmente, Erradi, não indica como ocorreria tais adaptações, especificamente em que critérios as mesmas seriam baseadas e onde especificar as mesmas. Ele generaliza em demasia ao afirmar que "adaptações serão baseadas na qualidade de serviço desejada". Como especificar tal qualidade e como informar a *engine* que critérios utilizar para avaliar esta qualidade são necessários. Por fim, alterar a lógica de execução de um processo de negócio em tempo de execução se constitui uma árdua tarefa. Esta questão não afeta simplesmente variáveis de

estado, mas altera semânticas em uma instância de processo já em execução. Uma boa referência para ter noção de tal complexidade é [McKinley 2004].

Por sua vez, Charfi direcionou seu trabalho na limitação das linguagens/padrões atuais de composição de *web services* em trabalhar na questão da modularidade e da adaptabilidade dinâmica. Como o objetivo desta dissertação está focado na segunda questão, os comentários irão se ater a mesma, embora seja possível afirmar que efetivamente a especificação de processos em WS-BPEL sofre com a falta de modularidade, e a proposta de Charfi minimiza a deficiência desta questão específica. Por ter focado em um ponto mais concreto e proposto uma metodologia mais simples, o trabalho de Charfi se encontra em um estágio mais avançado, em termos práticos, do que o trabalho de Erradi. Embora, segundo o próprio autor, ele ainda esteja trabalhando em um primeiro protótipo de uma *engine* de orquestração baseada em aspectos, a qual irá gerenciar processos BPEL e aspectos. Inclusive, o próprio autor já coloca exemplos de como ocorreriam as especificações em seu projeto, dando uma idéia aos pesquisadores de como poderia ser utilizada a *engine* do projeto AO4BPEL.

A proposta de Charfi e Erradi tem algumas deficiências em comum. Ambos propõem a extensão do padrão WS-BPEL para a incorporação de aspectos, ou, pelo menos, obrigariam a utilizar o padrão pré-existente associado com primitivas de POA. Alteração de padrões já bastante difundidos e aceitos recai sobre as mesmas considerações feitas sobre o trabalho de Karastoyanova (Seção 5.1.2). Adicionalmente, os programadores envolvidos no projeto do processo de negócio deverão ser capazes (ou então treinados) de desenvolver aplicações orientadas a aspectos, o que pode ser um fator complicador adicional em um projeto. Por fim, a adoção dessas propostas tendo em vista tornar um sistema adaptável não é transparente, a medida que o próprio código-fonte da aplicação deve ser reescrito e diversos artefatos devem trabalhar em conjunto a engine original para possibilitar a desejada adaptabilidade. E conforme o aumento no número destes artefatos, maior *overhead* no tempo de execução do processo de negócio e maior trabalho para a equipe envolvida no projeto (especialmente na configuração dos mesmos), o que geralmente acarreta em um maior tempo de desenvolvimento.

O projeto A-ActiveBPEL, proposto nesta dissertação, torna a tarefa de adaptação muito mais transparente para os programadores do processo de negócio, tendo em vista que nenhuma alteração é proposta no padrão WS-BPEL e o código-fonte do processo de negócio permanece inalterado. A responsabilidade de como prover adaptabilidade é transferida para a *engine*, livrando o programador desta tarefa, requisitando apenas que um repositório central de informações sobre *web services* seja atualizado. Além disto, nenhum novo conhecimento (como orientação a aspectos) é requisitado para a disponibilização da adaptabilidade. Uma consequência direta deste fato é a redução do tempo gasto no desenvolvimento do sistema. Adicionalmente, por não propor a adição de diversas ferramentas para a disponibilização de tal intento (apenas uma extensão, enquanto Charfi, por exemplo, propõe a extensão, o *Aspect Manager* e o repositório de aspectos, dentre outros), é de se esperar que o tempo de execução de uma requisição no A-ActiveBPEL seja menor do que os outros trabalhos. Por fim, a extensão A-ActiveBPEL já se encontra implementada, tendo sido validada com exemplos práticos já

utilizados em diversos outros trabalhos envolvendo composição de *web services* [Silva 2006] [ActiveBPEL 2006]. Os trabalhos relacionados mais atuais relacionados composição adaptativa de *web services* utilizando orientação a aspectos, [Charfi 2004] e [Erradi 2005], ainda se encontram em fase de desenvolvimento de protótipos e validação. Ou seja, ainda não estão disponíveis para uso pela comunidade científica.

5.3 Considerações Finais

Este capítulo apresentou os principais trabalhos relacionados ao tema desta dissertação. Uma visão geral de cada um deles foi apresentada, sendo depois realizada uma avaliação crítica sobre a adequabilidade das mesmas.

De forma resumida, duas correntes distintas foram identificadas na área de composição adaptativa de *web services*: uma liderada por Karastoyanova, em sua proposição do mecanismo "*find and bind*" e a outra trabalhada por diversos pesquisadores, tendo como principais exemplos Charfi e Erradi, que propuseram a utilização de orientação a aspectos.

O trabalho de Karastoyanova [Karastoyanova 2005] tem muitos pontos em comum com a abordagem proposta nesta dissertação. Dando a opção de se interligar um mecanismo adaptável a cada chamada de *web service* participante, verificando neste momento se existe alguma alteração no ambiente e agindo para considerá-la, caso exista, o autor pode disponibilizar adaptabilidade sem a necessidade de utilização de mecanismos mais sofisticados, como orientação a aspectos. Contudo, Karastoyanova propõe uma alteração sintática na especificação padrão do WS-BPEL, o que se constitui um fator complicador para a utilização de sua metodologia, obrigando inclusive o programador a especificar no código-fonte da aplicação onde o mecanismo adaptável poderá ser utilizado.

A outra metodologia sugere orientação a aspectos, tendo em vista prover adaptabilidade em composição de *web services*. Embora surja como uma idéia promissora, ela esbarra em diversos complicadores em termos de implementação, aliado ao fato de que obriga os programadores associados ao projeto a conhecerem o paradigma orientado a aspectos, o que pode se tornar um fator complicador. Por fim, o próprio código-fonte da aplicação deve ser alterado, de forma inclusive menos sutil do que o mecanismo "*find and bind*" proposto por Karastoyanova.

O trabalho proposto nesta dissertação torna a adaptação mais transparente, à medida que preserva o código fonte original da aplicação, passa a responsabilidade de adaptação para a *engine*, e requisita do programador apenas a atualização de um repositório de informações de *web services*, o qual é utilizado pela engine para identificar que serviços estão disponibilizados atualmente e indicar as suas propriedades não-funcionais.

CAPÍTULO

6

Conclusões e Trabalhos Futuros

"Quando sonhamos sozinhos é apenas um sonho. Quando sonhamos juntos é o começo da realidade"

D. Hélder Câmara.

Este capítulo apresenta as conclusões e as principais contribuições do presente trabalho. As limitações da metodologia proposta e os trabalhos futuros a serem realizados dentro desta linha de pesquisa também são destacados.

6.1 Conclusões

A adaptabilidade na área de composição de *web services* ainda é um campo em estudo, onde estão abertas diversas possibilidades de propostas e todas tem os seus méritos, assim como suas deficiências. O que se deve reconhecer é a necessidade de se encontrar uma alternativa viável e suficientemente abrangente, tendo em vista a enorme proliferação dos *web services* e dos chamados processos de negócio (que são baseados em composições desses mesmos *web services*). A vasta proliferação do SOA é visível, e as arquiteturas orientadas a serviços tendem a ocupar uma posição de destaque no mundo da engenharia de *software* e da computação em si. Devido as constantes mudanças que ocorrem com a disponibilização, a modificação e a exclusão de serviços, é extremamanente desejado que os sistemas tenham a capacidade de se adaptar a tais alterações ambientais sem causar maiores danos a continuidade dos serviços.

Este trabalho teve como objetivo propor uma abordagem para a composição adaptativa de *web services*, especialmente em tempo de execução. A proposta se mostrou válida e extremamente viável, sem requisitar, em

nenhum momento, a interrupção de qualquer módulo do sistema e sem alterar qualquer primitiva do padrão WS-BPEL. Outros trabalhos também propõem alternativas para esta questão, contudo eles pecam ao propor alterações sintáticas em padrões já extensamente difundidos e aceitos (como o WS-BPEL), requisitarem alterações no código-fonte original da aplicação (diminuindo a transparência da inserção da adaptabilidade no projeto) e requerendo que os programadores envolvidos no projeto tenham que adquirir conhecimentos que podem não estar ligados diretamente ao desenvolvimento do sistema (como orientação a aspectos).

O ponto central deste trabalho é a abordagem em si, sendo a parte prática apenas exemplo de aplicação. Embora tenha sido utilizada a *engine* de código aberto ActiveBPEL, qualquer outra poderia ser utilizada utilizando as idéias contidas na Seção 3.1 deste trabalho. Outro ponto a ser ressaltado é que todas as características desta *engine* foram preservadas, sendo apenas adicionada mais uma característica, a adaptabilidade. Não houve nenhuma perda de funcionalidade da *engine* em detrimento a adoção da adaptabilidade.

Os resultados obtidos através dos estudos de caso apresentados possibilitam afirmar que a extensão A-ActiveBPEL proposta implementou corretamente a mudança semântica da primitiva de invocação, possibilitando que a mesma pudesse se adaptar ainda em tempo de execução, baseada em critérios pré-estabelecidos pelo programador.

Adicionalmente, a idéia de procurar mudanças imediatamente antes das chamadas dos *web services* participantes acaba permitindo que adaptações sejam realizadas mesmo quando uma determinada instância de composição tenha iniciado sua execução e ainda não a tenha finalizado. Se o *web service* participante que deva ser permutado ainda não tiver sido executado, esta troca ocorrerá antes de sua execução. A probabilidade de uma determinada instância de um processo de negócio ser atualizado em sua execução aumenta consideravelmente, à medida que o mesmo apenas não será adaptado em tempo se o próprio serviço a ser trocado estiver em execução (ou já tiver sido executado). O tratamento do caso específico da troca em instâncias de processos que devam ter seus serviços participantes trocados ainda na execução dos mesmos é discutido na seção de trabalhos futuros.

Tendo em vista o *overhead* causado pela adoção da proposta deste trabalho, deve-se afirmar que estudos preliminares apontaram que existe esse *overhead* na adoção da mesma, contudo ele tem uma tendência a não ser significativo. Fatores que possibilitam este *overhead* reduzido passam pelo fato do *Web Service Repository* se localizar no mesmo servidor da *engine* adaptativa (o que diminui consideravelmente custos de comunicação) e por este repositório se tratar, no caso deste trabalho, de um simples arquivo texto (sistemas gerenciadores de banco de dados mais complexos poderiam inferir um maior custo de acesso). Adicionalmente, outros autores (como exemplo, [Charfi 2004]) também fizeram uma análise do custo (em termos de tempo de execução) da adoção de tais mecanismos adaptáveis na composição de *web services*; a conclusão obtida é que o tempo gasto não chega a ser significativo, em especial quando comparado com as vantagens de se obter um sistema adaptável. De toda forma, é proposta como trabalho futuro uma avaliação mais precisa deste

overhead, em especial utilizando-se redes de petri estocásticas para este fim [Silva 2006].

Por fim, temos a obrigação de destacar que o presente trabalho apresenta ainda algumas deficiências, que irão ser trabalhadas em trabalhos futuros. Tanto essas deficiências como as propostas relacionadas para elas serão especificadas posteriormente ainda neste capítulo.

6.2 Contribuições

A principal contribuição deste trabalho foi propor a mudança semântica da primitiva de invocação do padrão WS-BPEL, permitindo que a mesma adquirisse um comportamento adaptável, sem a necessidade de alterações sintáticas e de utilização de outras tecnologias que poderiam retardar o tempo de desenvolvimento do processo de negócio. A troca de *web services* participantes poderá ser feita em tempo de execução, sem a necessidade de reiniciar sistemas ou de reescrita de código.

Relacionada à contribuição principal, este trabalho apresenta uma forma compacta e intuitiva de especificar serviços no *Web Service Repository*. Sem requisitar a utilização de uma base de dados mais complexa (um arquivo de texto pode ser utilizado), utilizando uma sintaxe simples baseada em XML e agrupando *web services* semanticamente equivalentes, programadores poderão atualizar o repositório sem encontrar maiores dificuldades na realização desta tarefa.

Adicionalmente, foi implementada uma extensão adaptável para a *engine* ActiveBPEL, denominada A-ActiveBPEL, a qual poderá ser utilizada pela comunidade acadêmica e pelo mercado para a disponibilização da adaptabilidade nos processos de negócio, com todas as vantagens já citadas neste trabalho.

6.3 Limitações do Trabalho

Conforme explicitado anteriormente, este trabalho possui algumas limitações. Todas estas limitações terão indicações de aperfeiçoamento na seção de trabalhos futuros.

Inicialmente, a utilização da abordagem proposta neste trabalho tem como requisito que a interface dos *web services* que irão ser mudados sejam sintaticamente equivalentes. Esta limitação se baseia na idéia de que a *engine* adaptativa irá utilizar a mesma sintaxe de mensagens SOAP para a invocação das operações requisitadas. Se a interface usada pelo *web service* que for substituir for radicalmente diferente da sintaxe do atualmente utilizado, a mensagem SOAP não irá conseguir invocar o serviço desejado. O termo equivalente quer dizer, mais especificamente, que essas interfaces deverão ter o mesmo nome para as suas operações e parâmetros.

Por fim, o nível de adaptação proposto neste trabalho se baseia na troca de *web services* participantes da composição em tempo de execução, e não na mudança da lógica de negócio implementada no próprio processo de negócio, o que demanda uma estratégia mais elaborada para a sua efetivação em tempo de execução.

6.4 Trabalhos Futuros

Diversos trabalhos futuros são vislumbrados no contexto deste trabalho, alguns inclusive já iniciados mas ainda não finalizados, motivo pelo qual eles não foram apresentados anteriormente.

Inicialmente, está sendo trabalhada a adoção do padrão WS-Policy [Bajaj 2006], um padrão proposto para a especificação de políticas de *web services*, no contexto da abordagem proposta neste trabalho. Com esta idéia, serviços participantes e a própria composição (que também é um serviço) poderão ter suas políticas baseadas neste padrão, que permite a especificação de políticas de maneira extensível. Desta forma, os desenvolvedores de *web services* poderão descrever a política dos mesmos de uma maneira padronizada, tendo em vista possibilitar a escolha por parte da *engine* adaptativa. Adicionalmente, através do padrão Ws-Policy Attachment [Box 2006], uma política pode ser ligada diretamente a interface WSDL de um *web service*, através de um atributo dentro do mesmo informando a sua localização. Esta linha de trabalho tem sido adotada neste trabalho e em [Lins 2006], e será continuado através da adoção do WS-Policy.

Um requisito para a utilização da abordagem proposta neste trabalho é que os *web services* podem (e inclusive devem) ter implementações diferentes, contudo eles devem ter interfaces equivalentes. Para se tratar serviços com mesma semântica e diferentes interfaces, faz-se necessário algum formalismo capaz de expressar a semântica dos mesmos. Neste contexto, se vislumbra a utilização de uma solução que vem sendo proposta nesta área, denominada *Semantic Web Services* [McIlraith 2001]. Através de anotações semânticas, uma *engine* adaptativa poderia identificar a equivalência semântica entre dois *web services*, mesmo que eles apresentem interfaces visivelmente distintas. Através deste mecanismo, seria eliminada a necessidade de equivalência sintática das interfaces. Adicionalmente, em se tratando de composição automática de serviços, é requirido busca e seleção automática de serviços semanticamente equivalentes, o que pode ser provido por tecnologias de *Semantic Web Services*.

Outro ponto discutido neste trabalho é que a adaptação pode ser realizada inclusive no momento em que a instância do processo de negócio está sendo executada, no caso do *web service* participante a ser trocado ainda não ter sido chamado. Contudo, para tratar esse caso específico, em que justamente ele estiver sendo executado, faz-se necessário salvar de alguma forma o contexto atual da instância para que o serviço possa ser trocado e, consequentemente, utilizado com o contexto atualizado. Isto recai em uma questão clássica dos *web services*, onde os mesmos são *stateless* (ou seja, sem estado), pelo menos em sua versão padrão (é salutar relembrar que uma composição de *web services* também é um *web service*). Isso, de princípio, inviabilizaria o salvamento do contexto atual da execução da composição. Contudo, um grupo liderado por Ian Foster vem trabalhando em um projeto denominado *WS-Resource Framework* [Foster 2004], que introduz a noção de estado aos *web services*. São declarados recursos (*resources*), que são associados aos mesmos, provendo assim uma forma de disponibilizar ao mundo externo as suas variáveis de estado. Com posse deste tipo de informação, uma *engine* adaptativa seria capaz de acessar essas variáveis de estado e utiliza-las com o intuito de parar a execução da instância em execução, efetuar a troca do serviço e retornar a sua execução

normalmente. Propõe-se como trabalho futuro a incorporação de tal tecnologia, tendo em vista englobar o maior número possível de possibilidades de adaptação.

Finalmente, está sendo planejada também uma avaliação de desempenho (mais especificamente, levando-se em consideração tempo de execução) da abordagem proposta. É vislumbrado provar que tal abordagem não adiciona um *overhead* significativo ao tempo total de execução do processo de negócio. Esta avaliação será feita baseado no trabalho de [Silva 2006], que sugere a utilização de redes de petri estocásticas para a avaliação de desempenho da composição de *web services*.

Referências

[ActiveBPEL 2006] ActiveBPEL, L.L.C. “The Open Source BPEL Engine”. Disponível em http://www.activebpel.org (última visita: 02/09/2006).

[Agha 2002] Agha, G. A. “Adaptive Middleware”, em: Communications of the ACM, vol. 45, no. 6, pp. 30-32. 2002.

[Alonso 2003] Alonso, G. *et al*. “Web Services: Concepts. Architectures and Applications”. Springer-Verlag. Alemanha. 2003.

[Andrews 2006] Andrews, T. *et al*. “Business Process Execution Language for Web Services, version 1.1”, http://www.ibm.com/developerworks/library/ws-bpel/ (última visita: 04/09/2006).

[Apache 2006] Apache Software Foundation. “Apache Tomcat 6.0”. Disponível em http://tomcat.apache.org/. Acessado em 05/01/2006.

[Bailliez 2006] Bailliez, S. *et al*. “Apache ANT User Manual”. Disponível em http://ant.apache.org/manual. Acessado em 04/01/2007.

[Bajaj 2006] Bajaj, S. *et al*. “Web Services Policy Framework (Ws-Policy)”. Versão 1.2. Disponível em www.ibm.com/developerworks/library/ws-polfram. Acessado em 22/12/2006.

[Benatallah 2002] Benatallah, B., Dumas, M., Fauvet, M., and Rabhi, F. “Towards Patterns of Web Services Composition”, em: Patterns and Skeletons for Parallel and Distributed Computing. Springer Verlag, UK. 2002.

[Bockisch 2004] Bockisch, C. *et al*. “Virtual Machine Support for Dynamic Join Points”, em Proceedings of the 3th AOSD Conference. Lancaster, UK. 2004.

[Bolognesi 1987] Bolognesi, T. e Brinksma, T. “Introduction to the ISO Specification Language LOTOS”, em: Computer Networks and ISDN Systems, 14(1), pp. 25-29. 1987.

[Box 2006] Box, D. *et al*. “Web Services Policy Attachment”. Versão 1.2. Disponível em www.ibm.com/developerworks/library/ws-polatt. Acessado em 22/12/2006.

[Cardeli 1999] Cardeli, L. e Davies, R. “Service Combinators for Web Computing”, em: IEEE Transactions of Software Engineering, v. 25(3), pp. 309-316. 1999.

[Charfi 2004] Charfi, A. e Mezini, M. “Aspect-Oriented Web-Service Composition with AO4BPEL”, em: Proceedings of the European Conference on Web Services (ECOWS), v. 3250, pp. 168-182 (LCNS). Erfurt, Germany. 2004.

[Charfi 2005] Charfi, A. e Mezini, M. “Using Aspects for Security Engineering of Web Service Compositions”, em: Proceedings of the IEEE International Conference on Web Services (ICWS’05), pp. 59-66. Orlando, USA. 2005.

[Christensen 2001] Christensen, E. *et al*. “Web Service Description Language (WSDL) 1.1”. W3C Note. Disponível em http://www.w3.org/TR/wsdl. 2001.

[Courbis 2005] Courbis, C. e Finkelstein, A. “Towards Aspect Weaving Applications”, em: 27th International Conference on Software Engineering – ICSE. St. Louis, Missouri, USA. 2005.

[Czajkowski 2004] Czajkowski, K. *et al*. “The WS-Resource Framework Version 1.1”. Especificação técnica. Disponível em http://www.globus.org/wsrf/specs/ws-wsrf.pdf. Acessado em 22/12/2006.

[D’Hondt 2004] D’Hondt, M. e Jonckers, V. “Hybrid Aspects for Weaving Object-Oriented Functionality and Rule-Based Knowledge”, em Proceedings of the 3th AOSD Conference. Lancaster, UK. 2004.

[Erradi 2005] Erradi, A., Maheshwari, P. e Padmanabhuni, S. “Towards a Policy-Driven Framework for Adaptive Web Service Composition”, em: International Conference on Next Generation Web Services Practices (NweSP’05), pp. 33-38. 2005.

[Foster 2002] Foster, I. *et al*. “Grid Services for Distributed System Integration”, em: IEEE Press (Computer), vol. 35, no. 6, pp. 37-46. 2002.

[Foster 2004] Foster, I. *et al*. "The Ws-Resource Framework". Disponível em http://www.globus.org/wsrf/specs/ws-wsrf.pdf. 2004.

[Harold 2003] Harold, E.R. "Processing XML with Java". Addison-Wesley. ISBN: 0201771861. 2003.

[Hendricks 2002] Hendricks, M. *et al*. "Java Web Services". Alta Books. 2002.

[Grimes 97] Grimes, R. "Professional DCOM Programming". Wrox Press. Olton, Birmingham, Canada. 1997.

[Juric 2006] Juric, M.B. "*Business Process Execution Language for Web Services*". Packt Publishing, Mumbai, India. 2006.

[Karastoyanova 2004] Karastoyanova, D. e Buchmann, A. "Extending Web Service Flow Models to Provide for Adaptability", em: Proceedings of OOPSLA '04 Workshop on Best Practices and Methodologies in Service-oriented Architectures'. Vancouver, Canadá. 2004.

[Karastoyanova 2005] Karastoyanova, D. *et al*. "Extending BPEL for Run Time Adaptability", em: Proceedings of the 2005 Ninth IEEE International Enterprise Computing Conference. Holanda. 2005.

[Kiczales 2001] Kiczales, G, *et al*. "An overview of AspectJ", em Proceedings of the ECOOP 2001. Budapeste, Hungria. 2001.

[Lins 2006] Lins, F.A.A., Junior, J.C.S. e Rosa, N.S. "Policy-Driven Adaptive Web Service Composition", em: Proceedings of the OOPSLA '4th International Workshop on SOA and Web Services Best Practices', pp. 40-51. Portland, Oregon, Estados Unidos. 2006.

[McIlraith 2001] McIlraith, S.A., Son, T.C. e Zeng, H. "Semantic Web Services", em: IEEE Intelligent Systems and Their Applications, v. 16, pp. 46-53. 2001.

[McKinley 2004] McKinley, P.K. *et al*. "Composing Adaptive Software", em: Computer, v.37, n.7, pp. 56-64. 2004.

[Milanovic 2004] Milanovic, N. and Malek, M. "Current Solutions for Web Service Composition", em: IEEE Internet Computing, vol.8, n.6, pp.51-59. 2004.

[O'Sullivan 2005] O'Sullivan, J., Edmond, D. e Hofstede, A.H. "Formal Description of non-functional service properties". Technical report. Disponível em http://www.service-description.com. Acessado em 25/12/2006. 2005.

[OASIS 2004] OASIS. "UDDI Version 3.0.2". Disponível em http://uddi.org/pubs/uddi-v3.0.2-20041019.htm. Acessado em 06/02/2007. 2004.

[OASIS 2007] OASIS. “Web Services Business Process Execution Language Version 2.0”. Disponível em http://docs.oasis-open.org/wsbpel/2.0. Acessado em 01/03/2007. 2007.

[OMG 2002] OMG. “Common Object Request Broker Architecture: Core Specification (CORBA 3.0)”. 2002.

[Oracle 2006] Oracle. “Oracle BPEL Process Manager”. Disponível em: http://www.oracle.com/technology/products/ias/bpel. Acessado em 30/12/2006.

[Pallos 2001] Pallos, M. “Service-oriented Architecture: A Primer”, em: eAI Journal, pp. 32-35. 2001.

[Papazoglou 2003] Papazoglou, M.P. e Georgakopoulos, D. “Service Oriented Computing”, em: Communications of ACM, vol. 46, no. 10, pp. 25-28. 2003.

[Pawlak 2001] Pawlak, R. *et al*. “JAC: A Flexible Solution for Aspect-Oriented Programming in Java”, em: Proceedings of the 3th International Conference on Metalevel Architectures and Separation of Crosscutting Concerns. Japão. 2001.

[Plummer 2004] Plummer, D. “Service-Oriented Everything: Now is the Time”. Disponível em http://www.ftponline.com/weblogicpro/2004_05/magazine/features/dplummer/. Acessado em 17/12/2006. 2004.

[Gupta 2007] Gupta, S. “Service Oriented Architecture – Part 1”. Disponível em http://javaboutique.internet.com/tutorials/serv_orient/index-2.html. Acessado em 01/03/2006. 2007.

[Silva 2006] Silva, A.N., Lins, F.A.A., Junior, J.C.S., Rosa, N.S., Quental, N.C. e Maciel, Paulo R.M.M. “Avaliação de Desempenho da Composição de Web Services Usando Redes de Petri”, em: Simpósio Brasileiro de Redes de Computadores (SBRC 2006). Curitiba, Paraná, Brasil. 2006.

[Sun 2003] Sun Microsystems. “RMI Specification”. Disponível em http://java.sun.com/j2se/1.4.2/docs/guide/rmi/spec/rmiTOC.html. Acessado em 19/12/2006. 2003.

[Tsai 2006] Tsai, W.T. *et al*. “Architecture Classification for SOA-based application”, em: Proceedings of the Ninth International Symposium on Object and Component-Oriented Real-Time Distributed Computing (ISORC 2006), pp. 295-302. Gyeongju, Korea. 2006.

[W3C 2003] W3C. “SOAP Version 1.2”. Disponível em http://www.w3.org/TR/soap. 2003.

[W3C 2005] W3C. “Document Object Model”. Disponível em http://www.w3.org/dom. Acessado em 04/01/2007. 2005.

[W3C 2006] W3C. “Extensible Markup Language (XML 1.0”. Disponível em http://www.w3.org.xml. Acessado em 06/02/2007. 2006.

[van der Aalst 2003] van der Aalst, W.M.P., Dumas, M. e Hofstede. “Web Service Composition Languages: Old Wine in New Bottles?”, em: Proceedings of the IEEE 29th EUROMICRO Conference on New Waves in System Architecture, pp. 298-305. Turquia. 2003.

[Zeng 2004] Zeng, L. *et al*. “QoS-aware Middleware for Web Services Composition”, em: IEEE Transactions on Software Engineering, vol. 30(5), pp. 311-327. 2004.

A

Apêndice

Este apêndice se destina os códigos utilizados neste trabalho e que podem ser necessários para o entendimento do mesmo.

A.1 Código-fonte do Somador Distribuído

A.1.1 Especificação do Processo de Negócio (soma.bpel)

```
<!--                                                                    -->
<!-- BPEL Process Definition                                            -->
<!-- Edited using ActiveBPEL(tm) Designer Version 2.0.0 (http://www.active-
endpoints.com) -->
<process xmlns="http://schemas.xmlsoap.org/ws/2003/03/business-process/"
xmlns:bpws="http://schemas.xmlsoap.org/ws/2003/03/business-process/"
xmlns:ns1="http://172.17.105.62:8080/axis/Ws1.jws"
xmlns:ns2="http://172.17.105.62:8080/axis/Ws2.jws"
xmlns:ns3="http://172.17.105.62:8080/soma/soma"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" name="Soma"
suppressJoinFailure="yes" targetNamespace="http://Soma">

<partnerLinks>
<partnerLink name="somaPartnerLink"
partnerLinkType="ns3:somaPartnerLinkType" partnerRole="somaRole"/>
<partnerLink name="subtracaoPartnerLink"
partnerLinkType="ns3:subtracaoPartnerLinkType"
partnerRole="subtracaoRole"/>
<partnerLink myRole="requestRole" name="requestPartnerLink"
partnerLinkType="ns3:requestPartnerLinkType"/>
</partnerLinks>
```

```
<variables>
<variable messageType="ns1:somaRequest" name="somaRequest"/>
<variable messageType="ns3:respostaMessage" name="somaResponse"/>
<variable messageType="ns2:subtracaoRequest" name="subtracaoRequest"/>
<variable messageType="ns2:subtracaoResponse"
name="subtracaoResponse"/>
<variable messageType="ns3:parametrosMessage"
name="parametrosMessage"/>
<variable messageType="ns3:respostaMessage" name="respostaMessage"/>
<variable messageType="ns1:somaResponse" name="somaResponse1"/>
<variable messageType="ns1:subtracaoRequest" name="subtracaoRequest1"/>
<variable messageType="ns1:subtracaoResponse"
name="subtracaoResponse1"/>
</variables>

<flow>

<links>
<link name="L1"/>
<link name="L2"/>
</links>

<receive createInstance="yes" name="Receive" operation="request"
partnerLink="requestPartnerLink" portType="ns3:requestPT"
variable="parametrosMessage">
<source linkName="L1"/>
</receive>

<reply operation="request" partnerLink="requestPartnerLink"
portType="ns3:requestPT" variable="respostaMessage">
<target linkName="L2"/>
</reply>

<sequence>
<target linkName="L1"/>
<assign>

<copy>
<from part="argumento1" variable="parametrosMessage"/>
<to part="valor1" variable="somaRequest"/>
</copy>

<copy>
<from part="argumento2" variable="parametrosMessage"/>
<to part="valor2" variable="somaRequest"/>
</copy>

<copy>
<from part="argumento1" variable="parametrosMessage"/>
<to part="valor1" variable="subtracaoRequest1"/>
```

```
</copy>

<copy>
<from part="argumento2" variable="parametrosMessage"/>
<to part="valor2" variable="subtracaoRequest1"/>
</copy>

</assign>

<invoke inputVariable="somaRequest" name="SomaInvoke" operation="soma"
outputVariable="somaResponse1" partnerLink="somaPartnerLink"
portType="ns1:Ws1"/>

<invoke inputVariable="subtracaoRequest1" name="SubtracaoInvoke"
operation="subtracao" outputVariable="subtracaoResponse1"
partnerLink="somaPartnerLink" portType="ns1:Ws1"/>

<assign>
<source linkName="L2"/>
<copy>
<from expression="concat( bpws:getVariableData('somaResponse1',
'somaReturn') , bpws:getVariableData('subtracaoResponse1', 'subtracaoReturn')
)"/>
<to part="resposta_soma" variable="respostaMessage"/>
</copy>
</assign>

</sequence>
</flow>
</process>
```

A.1.2 Especificação das Interfaces dos *Web Services*

A.1.2.1 Interface do Processo de Negócio (soma.wsdl)

```
<definitions
targetNamespace="http://172.17.105.62:8080/soma/soma"
xmlns="http://schemas.xmlsoap.org/wsdl/"
xmlns:plnk="http://schemas.xmlsoap.org/ws/2003/05/partner-link/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:lns="http:// 172.17.105.62:8080/soma/soma"
xmlns:asns="http://172.17.105.62:8080/axis/Ws1.jws"
xmlns:apns="http://172.17.105.62:8080/axis/Ws2.jws"
xmlns:somadef="http://tempuri.org/services/somadefinitions">

<import namespace="http://172.17.105.62:8080/axis/Ws1.jws"
location="http://172.17.105.62:8080/axis/Ws1.jws?wsdl"/>
<import namespace="http://172.17.105.62:8080/axis/Ws2.jws"
location="http://172.17.105.62:8080/axis/Ws2.jws?wsdl"/>
```

```
<import namespace="http://tempuri.org/services/somadefinitions"
location="http://172.17.105.62:8080/axis/somadefinitions.wsdl"/>

<message name="parametrosMessage">
<part name="argumento1" type="xsd:integer"/>
<part name="argumento2" type="xsd:integer"/>
</message>

<message name="respostaMessage">
<part name="resposta_soma" type="xsd:string"/>
</message>

<message name="errorMessage">
<part name="errorCode" type="xsd:integer"/>
</message>

<portType name="requestPT">
<operation name="request">
<input message="lns:parametrosMessage"/>
<output message="lns:respostaMessage"/>
<fault name="unableToHandleRequest"
message="lns:errorMessage"/>
</operation>
</portType>

<plnk:partnerLinkType name="requestPartnerLinkType">
<plnk:role name="requestRole">
<plnk:portType name="lns:requestPT"/>
</plnk:role>
</plnk:partnerLinkType>

<plnk:partnerLinkType name="subtracaoPartnerLinkType">
<plnk:role name="subtracaoRole">
<plnk:portType name="apns:Ws2"/>
</plnk:role>
</plnk:partnerLinkType>

<plnk:partnerLinkType name="somaPartnerLinkType">
<plnk:role name="somaRole">
<plnk:portType name="asns:Ws1"/>
</plnk:role>
</plnk:partnerLinkType>

<service name="somaService"/>
</definitions>
```

A.1.2.2 Interface dos *Web Service* Ws1 (ws1.wsdl)

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
<wsdl:definitions targetNamespace="http://172.17.105.62:8080/axis/Ws1.jws"
xmlns:impl="http://172.17.105.62:8080/axis/Ws1.jws"
xmlns:intf="http://172.17.105.62:8080/axis/Ws1.jws"
xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:apachesoap="http://xml.apache.org/xml-soap"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">

<wsdl:message name="subtracaoRequest">
<wsdl:part name="valor1" type="xsd:integer"/>
<wsdl:part name="valor2" type="xsd:integer"/>
</wsdl:message>

<wsdl:message name="somaRequest">
<wsdl:part name="valor1" type="xsd:integer"/>
<wsdl:part name="valor2" type="xsd:integer"/>
</wsdl:message>

<wsdl:message name="subtracaoResponse">
<wsdl:part name="subtracaoReturn" type="xsd:string"/>
</wsdl:message>

<wsdl:message name="somaResponse">
<wsdl:part name="somaReturn" type="xsd:string"/>
</wsdl:message>

<wsdl:portType name="Ws1">
<wsdl:operation name="soma" parameterOrder="valor1 valor2">
<wsdl:input name="somaRequest" message="impl:somaRequest"/>
<wsdl:output name="somaResponse" message="impl:somaResponse"/>
</wsdl:operation>
<wsdl:operation name="subtracao" parameterOrder="valor1 valor2">
<wsdl:input name="subtracaoRequest" message="impl:subtracaoRequest"/>
<wsdl:output name="subtracaoResponse" message="impl:subtracaoResponse"/>
</wsdl:operation>
</wsdl:portType>

<wsdl:binding name="Ws1SoapBinding" type="impl:Ws1">
<wsdlsoap:binding xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/"
style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>

<wsdl:operation name="soma">
<wsdlsoap:operation xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/"
soapAction=""/>

<wsdl:input name="somaRequest">
<wsdlsoap:body xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://DefaultNamespace" use="encoded"/>
```

```
</wsdl:input>

<wsdl:output name="somaResponse">
<wsdlsoap:body xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://172.17.105.62:8080/axis/Ws1.jws" use="encoded"/>
</wsdl:output>
</wsdl:operation>
<wsdl:operation name="subtracao">
<wsdlsoap:operation xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/"
soapAction=""/>

<wsdl:input name="subtracaoRequest">
<wsdlsoap:body xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://DefaultNamespace" use="encoded"/>
</wsdl:input>

<wsdl:output name="subtracaoResponse">
<wsdlsoap:body xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://172.17.105.62:8080/axis/Ws1.jws" use="encoded"/>
</wsdl:output>

</wsdl:operation>
</wsdl:binding>
<wsdl:service name="Ws1Service">
<wsdl:port name="Ws1" binding="impl:Ws1SoapBinding">
<wsdlsoap:address xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/"
location="http://172.17.105.62:8080/axis/Ws1.jws"/>
</wsdl:port>
</wsdl:service>
</wsdl:definitions>
```

A.1.2.3 Interface dos *Web Service* Ws2 (ws2.wsdl)

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions targetNamespace="http://172.17.105.62:8080/axis/Ws2.jws"
xmlns:impl="http://172.17.105.62:8080/axis/Ws2.jws"
xmlns:intf="http://172.17.105.62:8080/axis/Ws2.jws"
xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:apachesoap="http://xml.apache.org/xml-soap"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">

<wsdl:message name="somaRequest">
<wsdl:part name="valor1" type="xsd:integer"/>
<wsdl:part name="valor2" type="xsd:integer"/>
</wsdl:message>
```

```
<wsdl:message name="somaResponse">
<wsdl:part name="somaReturn" type="xsd:string"/>
</wsdl:message>

<wsdl:message name="subtracaoResponse">
<wsdl:part name="subtracaoReturn" type="xsd:string"/>
</wsdl:message>

<wsdl:message name="subtracaoRequest">
<wsdl:part name="valor1" type="xsd:integer"/>
<wsdl:part name="valor2" type="xsd:integer"/>
</wsdl:message>

<wsdl:portType name="Ws2">
<wsdl:operation name="soma" parameterOrder="valor1 valor2">
<wsdl:input name="somaRequest" message="impl:somaRequest"/>
<wsdl:output name="somaResponse" message="impl:somaResponse"/>
</wsdl:operation>
<wsdl:operation name="subtracao" parameterOrder="valor1 valor2">
<wsdl:input name="subtracaoRequest" message="impl:subtracaoRequest"/>
<wsdl:output name="subtracaoResponse" message="impl:subtracaoResponse"/>
</wsdl:operation>
</wsdl:portType>

<wsdl:binding name="Ws2SoapBinding" type="impl:Ws2">
<wsdlsoap:binding xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/"
style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>

<wsdl:operation name="soma">
<wsdlsoap:operation xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/"
soapAction=""/>
<wsdl:input name="somaRequest">
<wsdlsoap:body xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://DefaultNamespace" use="encoded"/>
</wsdl:input>
<wsdl:output name="somaResponse">
<wsdlsoap:body xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://172.17.105.62:8080/axis/Ws2.jws" use="encoded"/>
</wsdl:output>
</wsdl:operation>

<wsdl:operation name="subtracao">
<wsdlsoap:operation xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/"
soapAction=""/>
<wsdl:input name="subtracaoRequest">
```

```
<wsdlsoap:body xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://DefaultNamespace" use="encoded"/>
</wsdl:input>
<wsdl:output name="subtracaoResponse">
<wsdlsoap:body xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://172.17.105.62:8080/axis/Ws2.jws" use="encoded"/>
</wsdl:output>
</wsdl:operation>
</wsdl:binding>

<wsdl:service name="Ws2Service">
<wsdl:port name="Ws2" binding="impl:Ws2SoapBinding">
<wsdlsoap:address xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/"
location="http://172.17.105.62:8080/axis/Ws2.jws"/>
</wsdl:port>
</wsdl:service>
</wsdl:definitions>
```

A.2 Código-fonte do Sistema de Aprovação de Créditos

A.2.1 Especificação do Processo de Negócio (loanapproval.bpel)

```
<!-- BPEL Process Definition                                    -->
<!-- Edited using ActiveWebflow(tm) Designer version 1.0.0
(http://www.active-endpoints.com) -->
<process xmlns="http://schemas.xmlsoap.org/ws/2003/03/business-process/"
xmlns:apns="http://tempuri.org/services/loanapprover"
xmlns:asns="http://tempuri.org/services/loanassessor"
xmlns:bpws="http://schemas.xmlsoap.org/ws/2003/03/business-process/"
xmlns:lns="http://loans.org/wsdl/loan-approval"
xmlns:loandef="http://tempuri.org/services/loandefinitions"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
name="loanApprovalProcess" suppressJoinFailure="yes"
targetNamespace="http://acme.com/loanprocessing">

<partnerLinks>
<partnerLink myRole="approver" name="customer"
partnerLinkType="lns:loanApprovalLinkType"/>
<partnerLink name="approver" partnerLinkType="lns:loanApprovalLinkType"
partnerRole="approver"/>
<partnerLink name="assessor" partnerLinkType="lns:riskAssessmentLinkType"
partnerRole="assessor"/>
</partnerLinks>

<variables>
<variable messageType="loandef:creditInformationMessage" name="request"/>
```

```
<variable messageType="asns:riskAssessmentMessage"
name="riskAssessment"/>
<variable messageType="apns:approvalMessage" name="approvalInfo"/>
<variable messageType="loandef:loanRequestErrorMessage" name="error"/>
</variables>

<faultHandlers>
<catch faultName="apns:loanProcessFault" faultVariable="error">
<reply faultName="apns:loanProcessFault" operation="approve"
partnerLink="customer" portType="apns:loanApprovalPT" variable="error"/>
</catch>
</faultHandlers>

<flow>

<links>
<link name="receive-to-approval"/>
<link name="receive-to-assess"/>
<link name="approval-to-reply"/>
<link name="assess-to-setMessage"/>
<link name="assess-to-approval"/>
<link name="setMessage-to-reply"/>
</links>

<receive createInstance="yes" name="receive1" operation="approve"
partnerLink="customer" portType="apns:loanApprovalPT" variable="request">
<source linkName="receive-to-approval"
transitionCondition="bpws:getVariableData('request', 'amount')>=10000"/>
<source linkName="receive-to-assess"
transitionCondition="bpws:getVariableData('request', 'amount')<10000"/>
</receive>

<invoke inputVariable="request" name="invokeapprover" operation="approve"
outputVariable="approvalInfo" partnerLink="approver"
portType="apns:loanApprovalPT">
<target linkName="receive-to-approval"/>
<target linkName="assess-to-approval"/>
<source linkName="approval-to-reply"/>
</invoke>

<invoke inputVariable="request" name="invokeAssessor" operation="check"
outputVariable="riskAssessment" partnerLink="assessor"
portType="asns:riskAssessmentPT">
<target linkName="receive-to-assess"/>
<source linkName="assess-to-setMessage"
transitionCondition="bpws:getVariableData('riskAssessment', 'risk')='low'"/>
<source linkName="assess-to-approval"
transitionCondition="bpws:getVariableData('riskAssessment', 'risk')!='low'"/>
</invoke>
```

```
<reply name="reply" operation="approve" partnerLink="customer"
portType="apns:loanApprovalPT" variable="approvalInfo">
<target linkName="approval-to-reply"/>
<target linkName="setMessage-to-reply"/>
</reply>

<assign name="assign">
<target linkName="assess-to-setMessage"/>
<source linkName="setMessage-to-reply"/>
<copy>
<from expression="'approved'"/>
<to part="accept" variable="approvalInfo"/>
</copy>
</assign>

</flow>
</process>
```

A.2.2 Especificação das Interfaces dos *Web Services*

A.2.2.1 Interface do Processo de Negócio (loanapproval.wsdl)

```
<?xml version="1.0" encoding="UTF-8" ?>
<definitions
targetNamespace="http://loans.org/wsdl/loan-approval"
xmlns="http://schemas.xmlsoap.org/wsdl/"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:plnk="http://schemas.xmlsoap.org/ws/2003/05/partner-link/"
xmlns:tns="http://loans.org/wsdl/loan-approval"
xmlns:lns="http://loans.org/wsdl/loan-approval">

<message name="creditInformationMessage">
<part name="firstName" type="xsd:string"/>
<part name="name" type="xsd:string"/>
<part name="amount" type="xsd:integer"/>
</message>

<message name="approvalMessage">
<part name="accept" type="xsd:string"/>
</message>

<message name="riskAssessmentMessage">
<part name="level" type="xsd:string"/>
</message>

<message name="errorMessage">
<part name="errorCode" type="xsd:integer"/>
</message>
```

```
<portType name="loanServicePT">
<operation name="request">
<input message="lns:creditInformationMessage"/>
<output message="lns:approvalMessage"/>
<fault name="unableToHandleRequest" message="lns:errorMessage"/>
</operation>
</portType>

<portType name="riskAssessmentPT">
<operation name="check">
<input message="lns:creditInformationMessage"/>
<output message="lns:riskAssessmentMessage"/>
<fault name="loanProcessFault" message="lns:errorMessage"/>
</operation>
</portType>

<portType name="loanApprovalPT">
<operation name="approve">
<input message="lns:creditInformationMessage"/>
<output message="lns:approvalMessage"/>
<fault name="loanProcessFault" message="lns:errorMessage"/>
</operation>
</portType>

<binding name="SOAPBinding" type="tns:loanApprovalPT">
<soap:binding style="rpc"
transport="http://schemas.xmlsoap.org/soap/http"/>
<operation name="approve">
<soap:operation soapAction="" style="rpc"/>
<input>
<soap:body use="encoded" namespace="urn:loanapprover"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
</input>
<output>
<soap:body use="encoded" namespace="urn:loanapprover"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
</output>
</operation>
</binding>

<service name="LoanApprover">
<documentation>Loan Approver Service</documentation>
<port name="SOAPPort" binding="tns:SOAPBinding">
<soap:address location="http://172.17.105.62:8080/active-
bpel/services/ApproverWebService"/>
</port>
</service>

<binding name="SOAPBinding1" type="tns:riskAssessmentPT">
```

```
<soap:binding style="rpc"
transport="http://schemas.xmlsoap.org/soap/http"/>
<operation name="check">
<soap:operation soapAction="" style="rpc"/>
<input>
<soap:body use="encoded" namespace="urn:loanassessor"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
</input>
<output>
<soap:body use="encoded" namespace="urn:loanassessor"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
</output>
</operation>
</binding>

<service name="LoanAssessor">
<documentation>Loan Assessor Service</documentation>
<port name="SOAPPort1" binding="tns:SOAPBinding1">
<soap:address location="http://172.17.105.62:8080/active-
bpel/services/AssessorWebService"/>
</port>
</service>

<plnk:partnerLinkType name="loanPartnerLinkType">
<plnk:role name="loanService">
<plnk:portType name="lns:loanServicePT"/>
</plnk:role>
</plnk:partnerLinkType>

<plnk:partnerLinkType name="loanApprovalLinkType">
<plnk:role name="approver">
<plnk:portType name="lns:loanApprovalPT"/>
</plnk:role>
</plnk:partnerLinkType>

<plnk:partnerLinkType name="riskAssessmentLinkType">
<plnk:role name="assessor">
<plnk:portType name="lns:riskAssessmentPT"/>
</plnk:role>
</plnk:partnerLinkType>

</definitions>
```

A.2.2.2 Interface do Serviço Assessor (loanAssessor.wsdl)

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
<definitions targetNamespace="http://tempuri.org/services/loanassessor"
xmlns:tns="http://tempuri.org/services/loanassessor"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:loandef="http://tempuri.org/services/loandefinitions"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns="http://schemas.xmlsoap.org/wsdl/">

 <import namespace="http://tempuri.org/services/loandefinitions"
location="project:/wsdl/loandefinitions.wsdl"/>

<message name="riskAssessmentMessage">
<part name="risk" type="xsd:string"/>
</message>

<portType name="riskAssessmentPT">
<operation name="check">
<input message="loandef:creditInformationMessage"/>
<output message="tns:riskAssessmentMessage"/>
<fault name="loanProcessFault"
message="loandef:loanRequestErrorMessage"/>
</operation>
</portType>

<binding name="SOAPBinding" type="tns:riskAssessmentPT">
<soap:binding xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
<operation name="check">
<soap:operation xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
soapAction="" style="rpc"/>
<input>
<soap:body xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="urn:loanassessor" use="encoded"/>
</input>
<output>
<soap:body xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="urn:loanassessor" use="encoded"/>
</output>
</operation>
</binding>

<service name="LoanAssessor">
<documentation>Loan Assessor Service</documentation>
<port name="SOAPPort" binding="tns:SOAPBinding">
<soap:address xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
location="http://172.17.105.62:8080/active-
bpel/services/AssessorWebService"/>
</port>
</service>
```

```
</definitions>
```

A.2.2.3 Interface do *Web Service* Aprovador (loanApprover.wsdl)

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions targetNamespace="http://tempuri.org/services/loanapprover"
xmlns:tns="http://tempuri.org/services/loanapprover"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:loandef="http://tempuri.org/services/loandefinitions"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns="http://schemas.xmlsoap.org/wsdl/">

<import namespace="http://tempuri.org/services/loandefinitions"
location="project:/wsdl/loandefinitions.wsdl"/>

<message name="approvalMessage">
<part name="accept" type="xsd:string"/>
</message>

<portType name="loanApprovalPT">
<operation name="approve">
<input message="loandef:creditInformationMessage"/>
<output message="tns:approvalMessage"/>
<fault name="loanProcessFault"
message="loandef:loanRequestErrorMessage"/>
</operation>
</portType>

<binding name="SOAPBinding" type="tns:loanApprovalPT">
<soap:binding xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
<operation name="approve">
<soap:operation xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
soapAction="" style="rpc"/>
<input>
<soap:body xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="urn:loanapprover" use="encoded"/>
</input>
<output>
<soap:body xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="urn:loanapprover" use="encoded"/>
</output>
</operation>
</binding>

<service name="LoanApprover">
<documentation>Loan Approver Service</documentation>
<port name="SOAPPort" binding="tns:SOAPBinding">
```

```
<soap:address xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
location="http://172.17.105.62:8080/active-
bpel/services/ApproverWebService"/>
</port>
</service>
</definitions>
```

A.3 Instalação da Extensão A-ActiveBPEL

Inicialmente, esta extensão foi desenvolvida e testada baseando-se na versão 2.0 da *engine* ActiveBPEL, tendo sido utilizados também o servidor Web Apache-Tomcat [Apache 2006] a o ANT [Bailliez 2006].

Para os usuários comuns, que tenham a necessidade de apenas utilizar a extensão A-ActiveBPEL em seu formato original, um pequeno conjunto de passos pode ser seguido para a realização de tal intento:

- ✓ Instalar a *engine* ActiveBPEL em um servidor Web (sugere-se o Apache-Tomcat, onde diversos *scripts* disponibilizados facilitam essa tarefa);
- ✓ Inserir os arquivos da extensão A-ActiveBPEL (Aactivebpel.java, informws.java e substituir o arquivo AeInvokeHandler.java da própria *engine* pelo disponibilizado) dentro do código-fonte da *engine*, preferencialmente criando um pacote dentro do org.activebpel.rt.axis.bpel, por exemplo, org.activebpel.rt.axis.bpel.aactivebpel;
- ✓ Especificar no Aactivebpel.java onde encontrar o arquivo do *Web Service Repository* (no caso específico deste trabalho, o mesmo foi colocado no diretório /bin do servidor web);
- ✓ Adicionar a biblioteca do JDOM (jdom.jar) na biblioteca do ActiveBPEL (.../source/activebpel/lib) e na biblioteca do servidor WEB (por exemplo, Apache-Tomcat);
- ✓ Inserir a dependência desta biblioteca dentro do arquivo common.xml que vem no ActiveBPEL;
- ✓ É necessário também especificar que bibliotecas determinado pacote poderá precisar em sua execução. Esta especificação é feita em um arquivo com a extensão .properties (e que leva o mesmo nome do pacote correspondente) que fica na pasta projects do código-fonte da *engine*. No caso deste trabalho, o pacote aactivebpel precisa do jdom, o qual deve ser incluído no arquivo org.activebpel.axis.bpel.properties, seguindo o mesmo padrão das outras bibliotecas neste local especificadas;
- ✓ Recompilar a *engine*, utilizando o comando: *ant –f activebpel.xml.*
- ✓ Copiar os arquivos gerados para dentro do servidor Web através do *script* install.bat que vem no ActiveBPEL.

Após estes passos, a extensão já estará integrada com a *engine* do ActiveBPEL, estando apta a executar o mecanismo de adaptabilidade proposto neste trabalho.

Dissertação de Mestrado apresentada por **Fernando Antonio Aires Lins** à Pós-Graduação em Ciência da Computação do Centro de Informática da Universidade Federal de Pernambuco, sob o título **"Composição Adaptativa de *Web Services*"**, orientada pelo **Prof. Nelson Souto Rosa** e aprovada pela Banca Examinadora formada pelos professores:

Prof. Paulo Roberto Freire Cunha
Centro de Informática / UFPE

Profa. Claudia Maria Fernándes Araújo Ribeiro
Faculdade de Ciências Exatas e Naturais - UERN

Prof. Nelson Souto Rosa
Centro de Informática / UFPE

Visto e permitida a impressão.
Recife, 28 de fevereiro de 2007.

Prof. FRANCISCO DE ASSIS TENÓRIO DE CARVALHO
Coordenador da Pós-Graduação em Ciência da Computação do
Centro de Informática da Universidade Federal de Pernambuco.