Universidade Federal de Pernambuco

Centro de Informática

Graduação em Ciência da Computação

MONITORAMENTO E AÇÕES CORRETIVAS DE SERVIÇOS EM COMPOSIÇÃO DE *WEB SERVICES*

Carlos Frederico Medeiros de Souza

Recife – Dezembro de 2008.

Universidade Federal de Pernambuco
Centro de Informática
Graduação em Ciência da Computação

MONITORAMENTO E AÇÕES CORRETIVAS DE SERVIÇOS EM COMPOSIÇÃO DE *WEB SERVICES*

Trabalho de Graduação

Carlos Frederico Medeiros de Souza

Orientador: Nelson Souto Rosa

Este trabalho foi apresentado ao Centro de Informática da Universidade Federal de Pernambuco como requisito parcial para a obtenção do Grau de Bacharel em Ciências da Computação.

Recife – Dezembro de 2008

Agradecimentos

- A Deus;

- À minha família, a minha esposa Glênya Pessoa, a minha filha Louise Medeiros, minha irmã Roberta Medeiros e a minha mãe Carmem Lúcia;

- Ao professor Nelson Souto Rosa, pela sua orientação sempre objetiva e rica de boas idéias;

- Ao professor José Rodrigues Lemos, diretor do Núcleo de Tecnologia da Informação da Universidade Federal Rural de Pernambuco;

- Ao recente mestre, doutorando e professor Fernando Antonio Aires Lins, pela sua colaboração para conclusão deste trabalho;

Resumo

Nos dias atuais, é crescente a necessidade de integração de aplicações e de processos de negócio existentes em organizações diferentes. O uso de SOA (*Service Oriented Architecture*) e a sua instanciação através de *web services* constituem uma abordagem interessante para esta integração. Em particular, a integração ocorre normalmente através da composição de *web services* providos pelas diferentes organizações.

Apesar da grande proliferação dos *web services*, muitos problemas ainda estão presentes na sua utilização em composição de serviços, tais como: a falta de ferramentas de suporte à evolução e à adaptação nos processos de negócio e a inexistência de uma padronização dos requisitos de qualidade e funcionalidade na integração destas mesmas aplicações.

Em relação ao suporte à evolução e à adaptação da composição, destacamos a necessidade de adaptação da composição para atender condições definidas em documentos de SLA (*Service Level Agreement*). Neste caso, através do monitoramento da composição, pode-se verificar que o SLA não está sendo atendido (e.g., o *web service* está com um desempenho abaixo do que foi definido no SLA) e que há necessidade de troca de um *web service* particular. Estas ações corretivas devem ser realizadas na composição em tempo de execução (configuração dinâmica) sem a necessidade de reinicialização do sistema, o que pode gerar problemas de indisponibilidade dos serviços.

Neste cenário, este trabalho apresenta uma proposta de configuração dinâmica de composição de *web services*. A configuração dinâmica é realizada considerando requisitos desejáveis da composição (descritos no documento de SLA) e informações de monitoramento da execução dos *web services*. A solução apresentada mantém a composição original, permitindo que o projetista defina qual ou quais *web services* possuirão serviços de reserva para eventuais trocas definidas a partir do SLA. A solução se baseia na modificação de uma *engine* de composição de *web services*, a *engine* ActiveBPEL, que permita o monitoramento e a troca de *web services*.

Palavras chave: *web service*, composição de serviços, composição de *web services*, processos de negócios, *Service Level Agreement* (SLA).

Abstract

Nowadays, it is increasing the need of integration of applications and business processes belonging to different organisations. The adoption of SOA (Service Oriented Architecture) principles and its instantiation to web services have been widely adopted in this integration through the composition of services provided by the organizations.

Despite the proliferation of web services technologies, it is possible to observe that many important isssues still open such, such as: absence of support to evolution and adaptation of web service compositions and lack of a strategy to treat with non-functional properties of web service compositions. In terms of adaptation, an important point to be considered refers to the need of composition satisfying conditions imposed by SLA (Service Level Aggreement) at runtime. In this case, the composition has to be monitored at runtime in order to check if it is satisfying constraints defined in the SLA. As soon as the monitoring mechanism observes that the conditions are not being satisfied, it is necessary to act to fix the problem, i.e., change the composition in order to make it compatible with the SLA constraints. Furthermore, the aforementioned action has to be taken without completely stopping the application.

In this scenario, this work presents an approach to enable dynamic configuration of web service compositions based on SLA and monitoring mechanisms. The proposed solution preserves the initial composition (description) and allows to define backup to the web services initially executing in the composition and that may be changed at runtime. In practical terms, the proposed solution consists of altering an opern source BPEL engine, namely AactiveBPEL, in such way that it may incorporate monitoring mechanisms, SLA elements and dynamic configuration skills.

***Keywords**: Web service, Service Composition, Business process and Service Level Agreement (SLA).*

Índice

Índice de Figuras

Capítulo 1

Introdução

"Ao se caminhar para um objetivo, sobretudo um grande e distante objetivo, as menores tarefas se tornam fundamentais".
Amyr Klink (do livro *Cem dias entre céu e mar*)

Este capítulo apresenta inicialmente o contexo, a motivação, o problema considerado e uma breve descrição da solução proposta.

1.1. Contexto e Motivação

Atualmente muitas organizações mantêm uma parte significativa dos seus processos de negócio nas Tecnologias da Informação (TI), em particular com a utilização de sistemas distribuídos (CORREIA e CARDOSO, 2005) e a integração entre estes sistemas é o elemento principal para o desenvolvimento de aplicações neste ambiente de TI. A integração possibilita a combinação eficiente dos recursos para aperfeiçoar as operações entre estas organizações (PAPAZOGLOU, 2006). Para realizar esta integração, adota-se normalmente o conceito de serviços através da Arquitetura Orientada a Serviço – SOA (*Service Oriented Architecture*). Esta arquitetura de computação não é nova, mas tomou um grande impulso recentemente devido a proliferação de serviços distribuídos na *Web*. No entanto, uma característica fundamental da *Web*, a sua heterogeneidade, é um obstáculo aos requisitos de rápida integração de aplicações. Como solução para este problema surgiu a tecnologia dos *web services*. Esta tecnologia simplifica o desenvolvimento de aplicações distribuídas com base num conjunto de protocolos e tecnologias (CORREIA e CARDOSO, 2005).

Neste contexto, a composição de *web services* tem surgido como uma importante estratégia para possibilitar a colaboração entre as organizações. Mais especificamente, a

composição permite que aplicações mais complexas sejam construídas a partir da combinação de serviços mais básicos. Esta estratégia facilita o desenvolvimento de sistemas distribuídos, favorece a reusabilidade de serviços e acelera 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 (LINS, 2007). Os padrões e protocolos utilizados pelos *web services* não fornecem nenhum método para cobrir eventuais falhas que possam ocorrer no fornecimento dos serviços sob os quais os *web services* são desenvolvidos. E estas falhas são mais facilmente verificadas quando os *web services* fazem parte de uma composição de serviços, pois um único *web service* sem funcionar corretamente compromete todo o funcionamento da composição (CORREIA e CARDOSO, 2005).

Assim, é crescente a importância na composição de *web services* em definir e manter recursos ativos, onde a composição seja capaz de se adaptar em caso de falhas, em contraposição aos tradicionais recursos passivos que simplesmente geram alguma mensagem de erro. Logo, ao invés de meramente enviar um alerta quando um dado serviço está impossibilitado de manter os requisitos em um nível previamente definido, o próprio serviço de composição deveria ser capaz de tomar ações corretivas para finalizar as operações (PAPAZOGLOU, 2006).

A principal atividade em uma composição de serviços é definir claramente os papéis e as funcionalidades necessárias para a integração de múltiplos serviços em uma única composição, formando, no final, outro serviço (PAPAZOGLOU, 2006). Logo, o resultado desta composição de serviços pode ser utilizado por outras composições como um novo serviço básico ou como uma solução completa para uma aplicação a ser oferecida a um cliente. Com a utilização do modelo onde serviços são montados a partir de outros serviços, um dos maiores problemas para a indústria, adotar largamente a Arquitetura de Orientação a Serviço, é a dificuldade em criar uma composição de processos de negócios distribuída, ou melhor, desenvolver tecnologias, métodos e ferramentas que suportem a composição de processos de negócios distribuídos de forma efetiva, flexível, confiável, fácil de usar, com baixo custo, dinâmica e eficiente. Padrões como o *Business Process Execution Language* – BPEL (OASIS, 2007) e a *Web Service Description Language* – WSDL (W3C, 2001) são as bases para a construção destas composições.

1.2. Problema

Os padrões (BPEL e WSDL) citados na seção anterior não definem como as composições devem tratar as falhas dos serviços componentes. Alguns problemas na Composição de *web services* podem ser caracterizados neste contexto e hoje representam um grande desafio para os seus desenvolvedores, tal como: a análise da composição para a substituição, compatibilização e conformidade de processos de negócios de forma dinâmica (em tempo de execução). Este problema tem suas raízes em duas falhas presentes na composição de *web services*:

- A falta de ferramentas de suporte à evolução e à adaptação nos processos de negócios. É difícil definir composições de processos de negócios distribuídos que funcionem propriamente em todas as circunstâncias. Erros nos acordos entre diferentes organizações e falhas na especificação e implementação dos protocolos, facilmente ocorrem, especialmente em processos complexos.
- Inexistência de uma padronização dos requisitos de qualidade e funcionalidade na integração de processos de negócios e conseqüentemente na disponibilização dos serviços.

A conformidade na qualidade dos serviços dos *web services* garante a integridade da composição. Isto é obtido através de uma comparação dos requisitos das suas operações com os valores existentes em um documento (acordo entre os serviços). Este documento reforça as regras de utilização e também de oferta do serviço existente na composição. Este modelo de documento pode ser conseguido através de um *Service Level Agreement* – SLA (MYERSON, 2002). Desta forma o problema é identificar qual ou quais *web services* presentes na composição não estão atendendo, naquele instante, os requisitos existentes no acordo (documento SLA) definido na oferta do serviço e conseqüentemente promover a sua substituição por outro *web service* (se houver). Esta substituição pode ser feita diretamente através da própria composição onde o serviço reserva já estava definido (modelo estático) ou através de uma descoberta de serviços reservas, onde algum outro provedor informe quais serviços podem ser utilizados como *backup* do *web service* da composição com problemas (modelo dinâmico).

1.3. Soluções parciais para o problema

Algumas propostas para resolver o problema anteriormente descrito já foram apresentadas (LINS, 2007). Duas correntes têm predominado: utilização de orientação a aspectos e alteração sintática no padrão WS-BPEL para composições de *web services*. A primeira proposta apresenta a desvantagem de não possuir um exemplo real, pois possui uma inerente dificuldade de implementação. Já a segunda necessita de modificações no padrão WS-BPEL para funcionar, o que para um padrão já estabelecido não é conveniente (LINS, 2007).

1.4. Proposta: SLA-ActiveBPEL

A solução proposta para tratar a configuração dinâmica da composição de serviços consiste na troca de *web services* no momento em que eles são invocados na composição. Esta troca é definida a partir da constatação de que um determinado *web service* não está atendendo as condições estabelecidas pelo SLA (requisitos de qualidade) e que precisa ser substituído por outro (*backup*) que esteja funcionando e disponível no momento (PAPAZOGLOU, 2006).

O objetivo principal da proposta é apresentar uma *engine* de composição de *web services* de código aberto que realize o monitoramento e ações corretivas (substituições) em composições de *web services* em tempo de execução, mantendo a sintaxe do padrão WS-BPEL.

1.5. Estrutura do Trabalho

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

Capítulo 2: Neste capítulo serão introduzidos todos os conceitos básicos necessários ao entendimento do trabalho.

Capítulo 3: Este capítulo apresenta a *engine* SLA-ActiveBPEL para configuração dinâmica de composição de *web services*.

Capítulo 4: Este capítulo apresenta uma aplicação que é executada pela *engine* SLA-ActiveBPEL.

Capítulo 5: Os trabalhos relacionados ao tema proposto são apresentados neste capítulo.

Capítulo 6: Finalmente, este capítulo apresenta as conclusões, limitações e os trabalhos futuros.

Capítulo 2

Conceitos Básicos

"... Guerrear e conquistar, em qualquer batalha,
não é o que há de melhor, mas sim quebrar, sem luta,
a resistência do inimigo."
Sun Tzu (do livro, *A arte da guerra*)

Neste capítulo são introduzidos os conceitos básicos, tecnologias e protocolos adotados neste trabalho. Inicialmente, os principais conceitos de SOA (*Service Oriented Architecture*) são introduzidos. Em seguida, são apresentados os padrões relacionados à tecnologia de *web service*. Por fim, a composição de serviços também é analisada, em especial a composição de *web services*.

2.1. Arquitetura Orientada a Serviço (SOA)

SOA também conhecida como Computação Orientada a Serviço (*Service Oriented Computing* – SOC) é um novo paradigma computacional que utiliza serviços como as estruturas básicas para dar suporte ao desenvolvimento rápido, de baixo custo e de fácil composição às aplicações distribuídas, mesmo em ambientes heterogêneos. O objetivo principal da Arquitetura Orientada a Serviço é possibilitar a cooperação entre os serviços que podem ser facilmente acoplados para criar processos de negócios dinâmicos e flexíveis. A Arquitetura Orientada a Serviço é muito vasta e enormemente complexa, englobando muitos conceitos e tecnologias que encontram suas origens em diversas disciplinas. Além disto, é necessário integrar as tecnologias com os processos e problemas dos negócios e com a estrutura organizacional das instituições (PAPAZOGLOU, 2006).

Com o advento de programas independentes de plataforma, a idéia de se ter serviços sendo executados transparentemente em diversas plataformas possui um atrativo especial, em particular quando se trata de integração de sistemas. O serviço é a unidade atômica da Arquitetura Orientada a Serviço 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 aspectos desta tecnologia é a reutilização de serviços previamente implementados e testados, reduzindo o tempo total de projeto 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. A larga proliferação dos *web services* (principal exemplo atual da utilização do SOA) e a disseminação do uso da *Internet* (portais, serviços na *Internet*, etc.) possibilitou a idéia de usar serviços disponibilizados na grande rede (em especial, por parceiros e/ou 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 (LINS, 2007).

A construção de sistemas baseados no SOA difere significativamente dos métodos tradicionais de projeto, como, por exemplo, aqueles baseados no paradigma funcional e no orientado a objetos. A utilização de serviços exige 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) (LINS, 2007).

2.2. *Web Services*

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 as tecnologias que constituem os padrões de *web services*.

2.2.1. XML

XML é uma abreviação de *Extensible Markup Language* que significa Linguagem de Marcação Estendida. Assim como a Linguagem de Marcação de Hiper Textos – HTML (*Hypertext Markup Language*), ela tem como finalidade marcar um determinado texto que sofrerá algum tipo de processamento. Podemos dizer que a XML é uma técnica para criar dados estruturados baseados em um arquivo do tipo texto. A estruturação dos dados deve-se ao fato dela identificar de forma inequívoca uma peça individual de informação (RAMALHO, 2002).

A XML foi criada para descrever dados, e o seu foco está na descrição do dado. A HTML, por sua vez, foi criada para exibir dados e seu foco é na aparência do dado. A linguagem XML não substitui a linguagem HTML, que continua sendo ainda a principal linguagem usada pelos navegadores (*browsers*) para exibirem informação na *Internet*. A XML permite ordenar, filtrar, localizar, arranjar informações de diferentes, apresentar dados de forma estruturada e permite o armazenamento da informação e sua descrição em um mesmo local (RAMALHO, 2002).

A linguagem XML se tornou o padrão para a descrição de dados que objetiva descrever o conteúdo de um documento. Para marcar o documento, marcadores (*tags*) são utilizados. Um marcador (*tag*) na 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 (LINS, 2007). Abaixo segue um exemplo de um código em linguagem XML.

```
(1)  <?xml version="1.0" encoding="UTF-8" ?>
(2)  <Registro>
(3)     <Titulo>MATRIX</Titulo>
(4)     <Secao>FICCAO CIENTIFICA</Secao>
(5)  </Registro>
```

O código em XML acima descreve os dados de um filme em uma locadora. O marcador raiz é codificado com as tags: `<Registro>` e `</Registro>`. Os outros dois marcadores (`<Titulo>` e `<Secao>`) formatam (estruturam) os dados no arquivo. Mais detalhes sobre a linguagem XML ver W3C (W3C, 2008).

2.2.2. SOAP

O SOAP (*Simple Object Access Protocol*) foi criado inicialmente para possibilitar a invocação (requisição) 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 (*Distributed Component Object Model*), o CORBA (*Common Object Request Broker Architecture*) e o RMI (*Remote Method Invocation*). 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 (*Hypertext Transfer Protocol*) (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 (LINS, 2007).

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 (*Simple Mail Transfer Protocol*) sejam descritos na especificação do padrão (LINS, 2007).

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. O SOAP é baseado na linguagem XML e os marcadores 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 (LINS, 2007).

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, podem

compartilhar facilmente uns com os outros: o texto. O HTTP, assim como o SMTP, pode facilmente obter um arquivo de texto de um computador e transferí-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* (LINS 2007).

2.2.3. 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*). 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. O 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 para descrição de *web services*. Para um cliente invocar um serviço, ele deve antes acessar a interface WSDL do serviço (*web service*)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 (*Universal Description, Discovery and Integration*), o qual pode ter informações acerca da localização URL (*Uniform Resource Locator*) 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, 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*.

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*. A 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.
- 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. A WSDL seguiu este preceito, porém adotou a noção de extensibilidade. Caso seja desejado acrescentar mais alguma informação como por exemplo, aspectos de qualidade de serviço, o programador pode propor a sintaxe adequada e inseri-la no documento WSDL;
- **Interoperabilidade:** como a WSDL é baseado em XML, um documento WSDL pode ser acessado por um grande número de máquinas e sistemas operacionais;

- **Suporte a diversos protocolos:** A WSDL suporta diversas alternativas para a interação com *web services*. Embora o SOAP seja a opção mais utilizada, um cliente enviando mensagens RMI/IIOP (Sun, 2008) 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 demonstrar preocupação de como e onde o serviço é oferecido. A parte concreta é responsável por vincular fisicamente o cliente ao serviço, similar a IDL (*Interface Description Language*) de CORBA. Na totalidade, WSDL (Versão 1.1) é composta por seis elementos, a saber: *definitions*, *types*, *message*, *portType*, *binding* e *service*.

O elemento raiz em um documento WSDL é o *definitions* (`wsdl:definitions`). Ele contém todos os elementos descritos anteriormente, bem como outros para especificar o *namespace* 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. 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 (LINS, 2007).

2.2.4. 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 outro nível, o nível de registro de serviços. Este ambiente propiciou o surgimento do UDDI (*Universal Discovery, Description and Integration*) (OASIS, 2007). 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 maneira que ele seja constantemente atualizado). Adicionalmente, os dados UDDI são estruturados de maneira semelhante a 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;
- **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. Outro elemento de extrema importância no mecanismo UDDI é o seu modelo técnico (conhecido como *tModel*) (OASIS, 2007). O *tModel* representa uma especificação técnica em um registro UDDI. Este 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 (LINS, 2007).

2.3. Composição de serviços

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. 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 (LINS, 2007).

Os serviços podem ser compostos de maneira estática ou dinâmica. 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 a 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 requerem a interrupção da continuidade do processo. Composição dinâmica é uma alternativa importante neste caso, dado que existem processos críticos que não podem sofrer interrupções (LINS, 2007).

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, mais detalhes destas formas de composição podem ser vistas em Lins (LINS, 2007). Estes diferentes tipos de composição podem ser incorporados em linguagens de composição utilizando definições de operadores e combinadores (LINS, 2007).

Na próxima seção veremos um exemplo muito comum de integração de serviços, a composição de *web services*.

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 Serviço, *web services* devem ser compostos com o objetivo de se construir aplicações maiores e de significativa complexidade. 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) destes aspectos.

A composição de *web services* se preocupa com a implementação interna das operações dos serviços. O desenvolvimento de um serviço composto de uma instituição 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 (LINS, 2007).

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 para a especificação de sua interface, o SOAP para o envio e recebimento de informações e o UDDI 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) (LINS, 2007). Estes padrões serão detalhados nas próximas seções.

2.3.2. Processos de Negócio

Os processos de negócio (*business processes*) visam especificar um conjunto de atividades realizadas pelas organizações (OASIS, 2007). Estão associadas a este conceito as informações por ele manipuladas, a utilização de recursos (*resources*) e elementos da estrutura organizacional da instituição. Aqui, o foco principal é no negócio, e neles são baseadas a especificação e a implementação dos processos. 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 instituição ou diversas outras (LINS, 2007).

Os processos de negócio visam, em especial, a integração de sistemas, de parceiros comerciais e de 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. Devido às complexidades e custos envolvidos no desenvolvimento e gerenciamento de conexões em todos estes sistemas e pessoas, os processos de negócio continuam sendo um objetivo difícil para muitas empresas (LINS, 2007).

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 um *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* (LINS, 2007).

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. Este fato é decorrente da inexistência de um padrão específico para a composição de *web services*; comumente, 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 (LINS, 2007). Na próxima seção veremos mais detalhes deste padrão de composição de processos de negócio.

2.3.3. Linguagem 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 (*Web Service Flow Language*) da IBM e XLANG da Microsoft. 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 dos processos 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 (LINS, 2007).

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 novo *web service*. Adicionalmente, o padrão WS-BPEL assume que os serviços a serem compostos estão descritos usando WSDL (LINS, 2007).

Especificações baseadas em WS-BPEL usam XML para descrever aspectos do processo: parceiros nas trocas de mensagens, serviços disponíveis, a orquestração, o formato das mensagens, dentre outros aspectos (LINS, 2007). Na próxima seção veremos um exemplo de implementação (*engine*) que utiliza o padrão de composição de processos negócios WS-BPEL conhecida como ActiveBPEL (ActiveBPEL, 2008).

2.3.4. *Engine* ActiveBPEL

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, como um exemplo, será aqui apresentada e utilizada neste trabalho, a ActiveBPEL (ActiveBPEL, 2008). Por este

motivo, ela será alvo de um aprofundamento em alguns aspectos práticos. A *engine* ActiveBPEL 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 (LINS, 2007).

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 na *engine* 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;
- ***.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 arquivo descritor de implantação (.bpr). Este arquivo é utilizado no caso das interfaces WSDL dos *web services* participantes se encontrarem dentro do próprio projeto, e não na *Internet*.

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) podem ser encontrados na documentação oficial sobre a mesma (ActiveBPEL, 2008).

2.4. Documentos SLA

A qualidade dos serviços na tecnologia da informação, assim como a de vários outros produtos da área da computação, está fortemente ligada aos níveis de qualidade de serviços prestados pelos fornecedores. Esses níveis de serviços podem ser medidos e avaliados por meio de fatores e critérios objetivos, a partir de um nível de qualidade dos serviços acordado entre o provedor do serviço e o cliente, chamado de SLA (*Service Level Agreement*) (WESTPHALL, et al., 2000).

Um documento SLA é uma definição fomal do relacionamento que existe entre um fornecedor de serviços e seus clientes. Um SLA pode ser definido e usado no contexto de quaisquer tipos de negócios e especifica o que o cliente deve esperar do seu fornecedor de serviços e as obrigações dos clientes, tais como aspectos de qualidade, desempenho (*performance*), disponibilidade e segurança. Este documento também definine os procedimentos que devem ser seguidos para manter a conformidade, quando são usados por instituições que não possuem um acordo previamente firmado. Este aspecto é o mais importante no ambiente de composições de serviços, pois no contexto deste trabalho, os *web services* são utilizados sem a necessidade de adaptação para cada cliente, simplesmente o desenvolvedor da composição utiliza o *web service* de acordo com as funcionalidades (interface WSDL)que cada um apresenta. Logo o documento SLA é fundamental para que o cliente possua uma descrição de como o serviço será utilizado e quais níveis de qualidade o *web service* pode oferecer (VERMA, 2005). Um *Service Level Agreement* pode possuir tipicamente as seguintes informações: uma descrição da natureza dos serviços que serão fornecidos; o desempenho esperado do nível de serviço, especificamente sua confiabilidade e disponibilidade; os procedimentos necessários para os clientes informarem problemas com o serviço; o tempo de resposta que o fornecedor define para resolver os problemas relatados pelos clientes; qual o processo de gerência de monitoramento dos *web services* serão utilizados; quais as conseqüências para o fornecedor de serviços caso ele não atenda aos requisitos descritos no documento SLA (VERMA, 2005).

É possível que os documentos SLA não apresentem todos os componentes descritos anteriormente, pois as características dos serviços oferecidos não demandam todos os aspectos mencionados.

2.5. Composição dinâmica de serviço

Em projetos relacionados à adaptabilidade de composições de *web services*, devemos avaliar três aspectos: como, quando e onde o monitoramento e as ações corretivas devem ser executados. O primeiro aspecto, "como", é tratado no contexto de WS-BPEL da seguinte forma: quando a *engine* inicia o processo de execução de uma especificação BPEL, esta 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, com uma única exceção, 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>`. Contudo, a URL do *web service* associada à invocação pode ser modificada, aqui está a característica principal da composição BPEL que nos permitiu implementar a solução proposta neste trabalho. A idéia principal é procurar os serviços reservas associados ao *web service* original presente na especificação da composição imediatamente antes de se efetuar cada invocação. Se for observado que o *web service* analisado possui um reserva e também ele não está atendendo aos requisitos do documento SLA, o mecanismo de substituição do *web service* é ativado antes da chamada ser direcionada para o *web service* participante original. 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* quanto ao atendimento dos níveis definidos no documento SLA, verifica se existe pelo menos um *web service* reserva para o *backup* e escolhe um deles (caso exista mais de um serviço reserva definido) baseado em um determinado critério ou política de *backup*.

Como mencionado anteriormente, o segundo aspecto básico é "quando" executar o monitoramento e as ações corretivas na composição. A abordagem proposta adota uma estratégia específica que possibilita o acoplamento de serviços compatíveis (com possíveis implementações diferentes, mas com interfaces WSDL iguais) em tempo de execução. As mudanças no repositório (arquivo de configuração dos *web services* reservas) podem ocorrer a qualquer momento, contudo elas só serão percebidas pela *engine* SLA-ActiveBPEL (descrita no próximo capítulo) e só terão efeito nos instantes anteriores à chamada ao *web service* participante. Desta forma, o mecanismo de substituição é acionado, caracterizando a reconfiguração (adaptação) da *engine* devido a alteração da URL dos *web services* que não estão em conformidade com o documento SLA.

Finalmente, o último aspecto a ser considerado é "onde" o mecanismo de monitoramento e ações corretivas deve ser inserido. Na abordagem proposta para este trabalho e diante dos requisitos já apresentados, o código deste mecanismo é inserido dentro da própria *engine*. Desta forma, as sintaxes das primitivas do padrão WS-BPEL, em especial o `<invoke>`, não precisam ser alteradas.

2.6. Considerações Finais

Vimos neste capítulo os conceitos, tecnologias e protocolos necessários ao entendimento do trabalho. Introduzimos o conceito de SOA e mostramos a sua importância no contexto da abordagem proposta. Apresentamos as principais tecnologias e padrões utilizados no SOA, como os *web services*, a WSDL, SOAP e o WS-BPEL. Descrevemos também, o modelo de composição de processos de negócio, mais especificamente a composição de *web services*, o conceito de SLA e a composição dinâmica de *web services*.

Capítulo 3

Engine SLA-ActiveBPEL

"Todos nós vivemos sob o mesmo céu, mas ninguém tem o mesmo horizonte!"

Konrad Adenauer

Este capítulo apresenta detalhes do desenvolvimento da *Engine* SLA-ActiveBPEL. Inicialmente é apresentada uma visão geral do funcionamento de uma *engine* de composição de *web services*. Em seguida são definidos os requisitos, a arquitetura, o projeto, as alterações realizadas na *engine* ActiveBPEL e a sua implementação.

3.1. Visão Geral

A Figura 3-1 mostra os principais elementos presentes em uma composição de *web services* no padrão WS-BPEL.

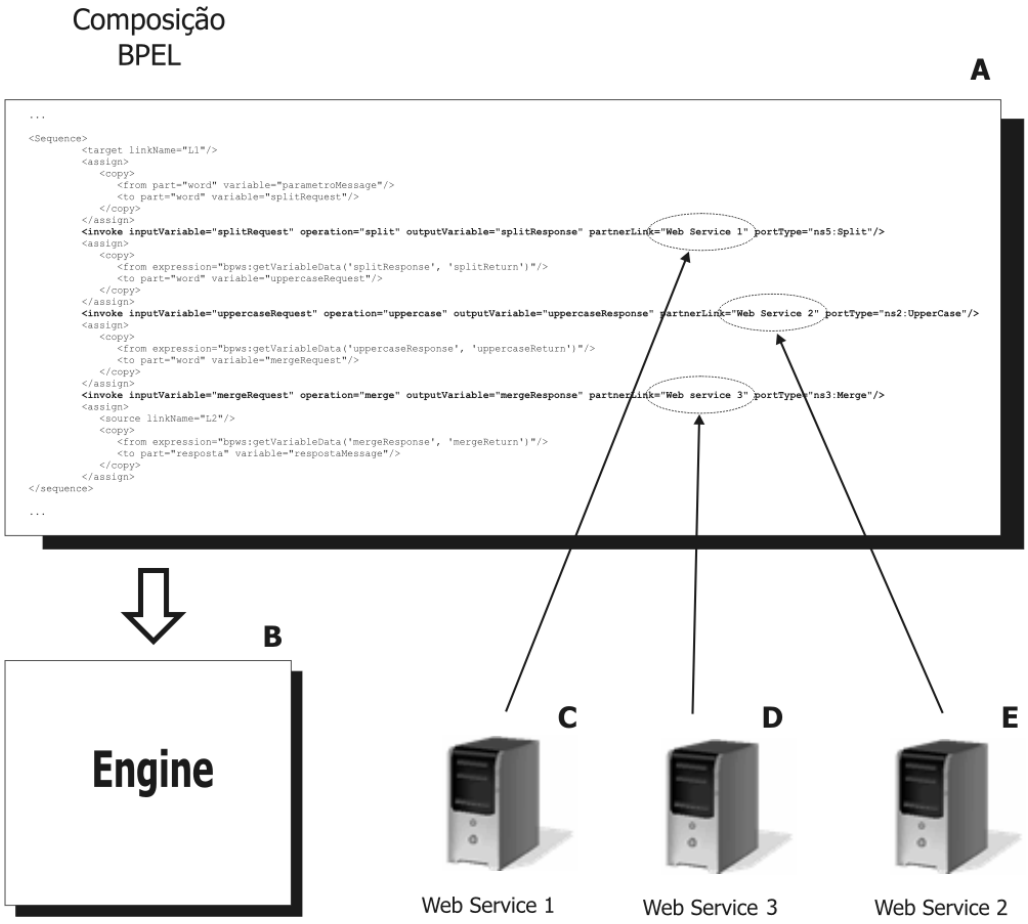

Figura 3-1. Componentes de uma composição de *web services*

Os componentes presentes na Figura 3-1 estão identificados por letras em negrito e são descritos como segue:

- (**A**) **Composição BPEL**. Este elemento contém a descrição WS-BPEL da composição de *web services*. No exemplo, é visto a presença dos *web services* 1, 2 e 3 na especificação do arquivo (.bpel).
- (**B**) ***Engine* de composição**. A ferramenta utilizada para executar a composição WS-BPEL. Esta ferramenta carrega o arquivo (BPEL) e executa as invocações aos respectivos *web services* definidos no processo de negócios.
- (**C, D e E**) ***Web service* 1, 2 e 3**. *Web services* que implementam funcionalidades do negócio.

Os elementos apresentados fazem parte de uma composição tradicional de composição de *web services* baseada em WS-BPEL. A modificação sugerida na extensão proposta da ActiveBPEL, a *engine* SLA-ActiveBPEL, requer a adição de novos elementos neste cenário, tais como: definição de requisitos de qualidade associados aos *web services*, um mecanismo de monitoramento dos *web services* envolvidos na composição e uma forma de definir os *web services* reservas (*backup*) da composição. Diante disto, os requisitos a seguir foram definidos:

- Implementação de um mecanismo de monitoramento dos *web services*;
- Definição de um modelo para expressar e obter os parâmetros de qualidade dos *web services*;
- Um repositório para armazenar os *web services* reservas.

Com estes requisitos principais da abordagem é possível então definir os requisitos específicos para a solução. Na próxima seção estes requisitos serão apresentados.

3.2. Requisitos específicos da solução

Como mencionado anteriormente, este trabalho visa prover a substituição de *web services* em tempo de execução de maneira transparente a partir de comparações efetuadas entre os requisitos de qualidade apresentados no SLA e a qualidade provida pelos *web services* em tempo de execução. Para isto, os seguintes requisitos específicos foram definidos:

- 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 organizações distintas, mas precisam ter a mesma interface WSDL para que sejam considerados serviços reservas (*backup*) dos *web services* originais. Veja que a semântica das operações não pode ser verificada na nossa abordagem, portanto mesmo que os *web services* reservas possuam a mesma interface WSDL, não podemos garantir que os resultados satisfaçam aos processos de negócio presentes na composição. Isto é devido ao fato das

operações (apesar de possuirem parâmetros e assinaturas iguais aos originais) poderem retornar valores totalmente diferentes.

- Os *web services* presentes na composição devem possuir na sua interface WSDL uma funcionalidade de controle onde se possa obter os dados em tempo de execução dos requisitos que serão comparados com o documento SLA. Esta interface de controle captura as informações internas dos *web services* para serem enviadas para a *engine*. Todos os *web services* devem possuir esta interface, inclusive os serviços reservas.
- A definição de quais dados devem estar presentes na interface de controle fica inteiramente a cargo dos projetistas da composição. Estas informações são totalmente parametrizáveis e não possuem um padrão definido, no entanto, uma vez estabelecidos devem ser os mesmos em toda a composição. Na nossa abordagem, utilizaremos como interface de controle um único método retornando um valor numérico que representa uma métrica presente no *web service*.
- A modificação deverá prever uma forma de monitorar os *web services* através da *engine*, não sendo permitido a utilização de ferramentas externas. Desta forma será necessário fazer modificações no código fonte da *engine* adotada para realizar a execução da composição.
- Utilização de uma *engine* de composição de *web services* de código aberto. Este requisito é fundamental, pois a modificação depende do acesso ao código fonte da ferramenta.
- A solução não pode fazer qualquer tipo de modificação nos padrões existentes na composição de *web services* (WS-BPEL) ou nos padrões de *web services* (WSDL e SOAP).
- A modificação proposta não pode afetar a execução normal da *engine*, apenas deve possibilitar a substituição dos *web services* sem comprometer as outras funcionalidades da *engine*. Além disto, o *overhead* adicionado deve ser mínimo.

Após estes requisitos terem sido definidos, a próxima seção descreve uma proposta de arquitetura para o desenvolvimento da *engine* SLA-ActiveBPEL.

3.3. Arquitetura da *engine* SLA-ActiveBPEL

Uma vez definidos os requisitos, apresentamos a arquitetura da *engine* SLA-ActiveBPEL. Já vimos que é necessária uma modificação no código da *engine*, em especial na *engine* A-ActiveBPEL (LINS, 2007). Também precisamos criar um repositório (neste trabalho será um arquivo texto no formato XML) onde seja possível definir os *web services* reservas.

A Figura 3-2 apresenta a arquitetura da solução proposta neste trabalho.

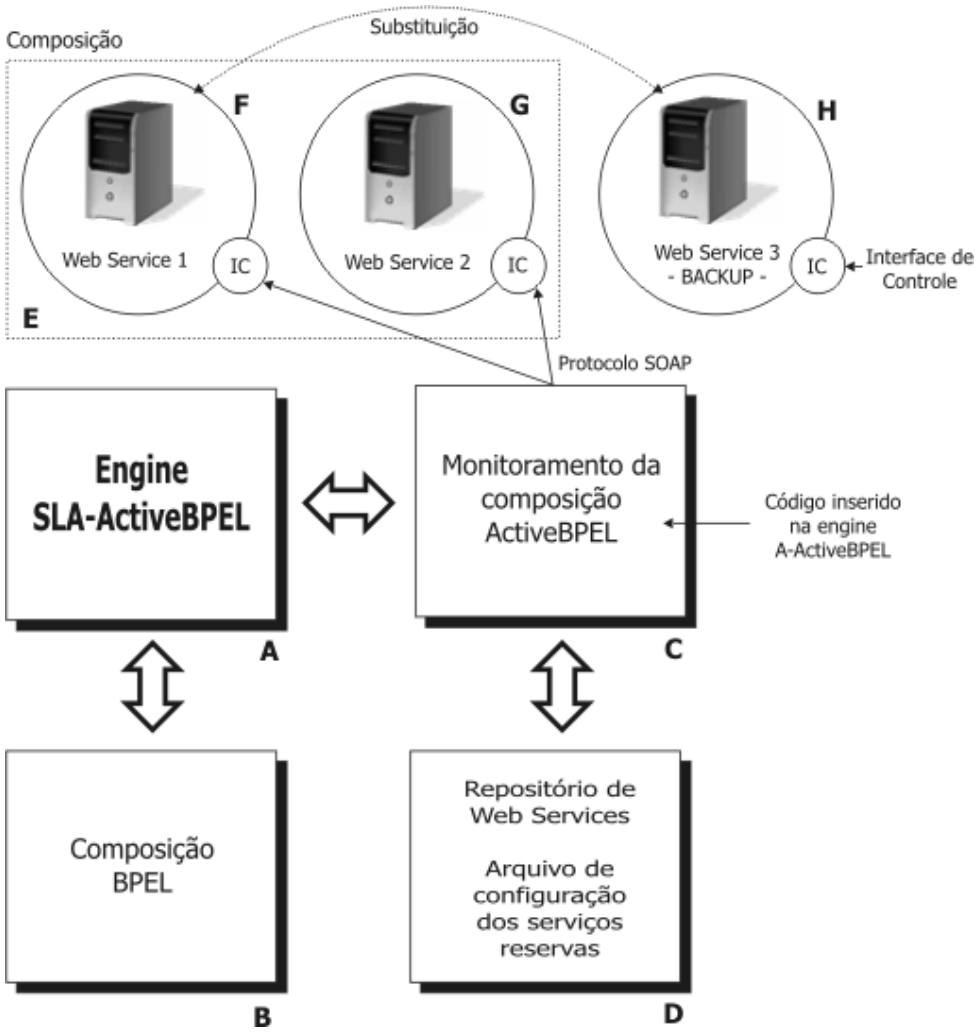

Figura 3-2. Arquitetura da solução proposta

Segue a descrição detalhada de cada um dos elementos presentes nesta arquitetura. Na Figura 3-2, cada componente está identificado por uma letra em negrito:

- (**A**) A *engine* modificada, já com a inserção do código necessário ao monitoramento e às ações corretivas;

- (**B**) A composição BPEL desenvolvida e já disponibilizada na *engine* para execução;
- (**C**) A ferramenta desenvolvida para ser inserida no código da *engine* e prover o mecanismo de monitoramento e substituição dos *web services*;
- (**D**) Repositório de configuração com a definição dos *web services* e seus serviços reservas.
- (**E**) Composição BPEL, constituída pelos *web service* 1 e *web service* 2;
- (**F**) *Web service* 1 e (**G**) *Web service* 2 são usados na composição;
- (**H**) *Web service* 3 (reserva). Neste caso, ele está definido como serviço *backup* do *web service* 1.

Na Figura 3-2 podemos observar ainda a presença da interface de controle (IC) nos *web services*. É importante observar que o documento SLA utilizado neste trabalho não foi representado nesta arquitetura. Isto se deve ao fato da utilização de documentos SLA não fazer parte da abordagem proposta, mas sim da solução que será implementada.

Na próxima seção descreveremos o projeto da *engine* SLA-ActiveBPEL, tomando como base a arquitetura definida.

3.4. Projeto

O próximo passo para a criação do Diagrama de Classes é a decomposição, em classes, dos elementos presentes na arquitetura mostrada na Figura 3-2 e da utilização dos requisitos já definidos. Inicialmente foi mostrado um elemento principal na arquitetura: o monitoramento dos *web services*. Este elemento também se comunica com a *engine* original, faz requisições aos *web services* da composição e faz consultas ao repositório dos *web services* reservas e ao documento SLA. A comunicação com a *engine* original será feita pela modificação da classe `AeInvokeHandler` pertencente ao pacote `org.activebpel.rt.axis.bpel` da implementação Java da *engine* ActiveBPEL. Esta modificação se baseia na inserção de um objeto (classe de monitoramento) antes das chamadas (invocação) dos *web services*. Maiores detalhes serão apresentados na próxima seção (implementação). A Figura 3-3 representa a classe `MonitoringTool`, ela é a classe principal da arquitetura proposta.

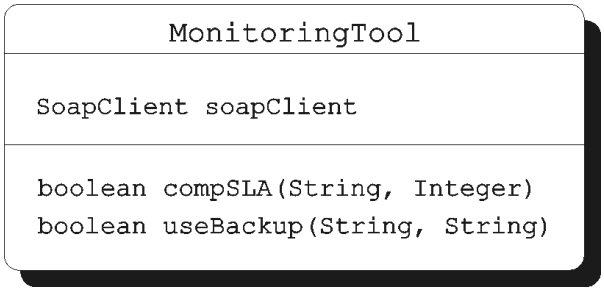

Figura 3-3. Classe MonitoringTool

A classe mostrada na Figura 3-3 apresenta um objeto denominado `soapClient`. Este objeto (do tipo `SoapClient`) é responsável por realizar a comunicação (requisições) com os *web services* da composição. Este objeto é a implementação de um cliente SOAP. Logo, a comunicação da classe de monitoramento com os *web services* será realizada utilizando o mesmo protocolo para a comunicação entre *web services*. O método `compSLA()` da classe `MonitoringTool` é o responsável por realizar a comparação dos dados obtidos da interface de controle, através do `soapClient()`, com os dados presentes no arquivo SLA. Já o método `useBackup()` será utilizado pelo objeto `MonitoringTool`, instanciado na classe `AeInvokeHandler`, para fazer a chamada do objeto `soapClient`. A classe `SoapClient` é mostrada na figura abaixo.

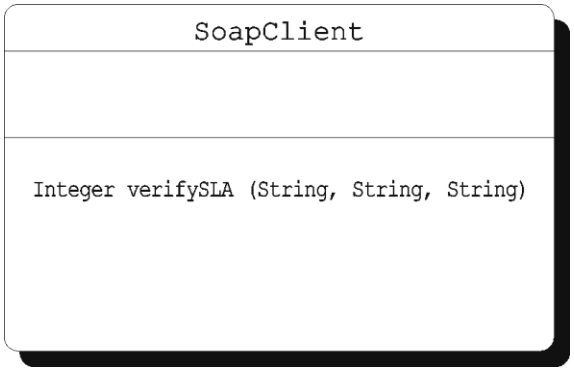

Figura 3-4. Classe SoapClient

A classe `SoapClient` possui um único método responsável por realizar a comunicação com os *web services* da composição e obter os dados através da interface de controle. As duas classes `MonitoringTool` e `SoapClient` foram inseridas em um novo pacote `org.activebpel.monitor` na implementação da *engine* original (A-ActiveBPEL). A Figura 3-5 apresenta o Diagrama de Classes da *engine* SLA-ActiveBPEL.

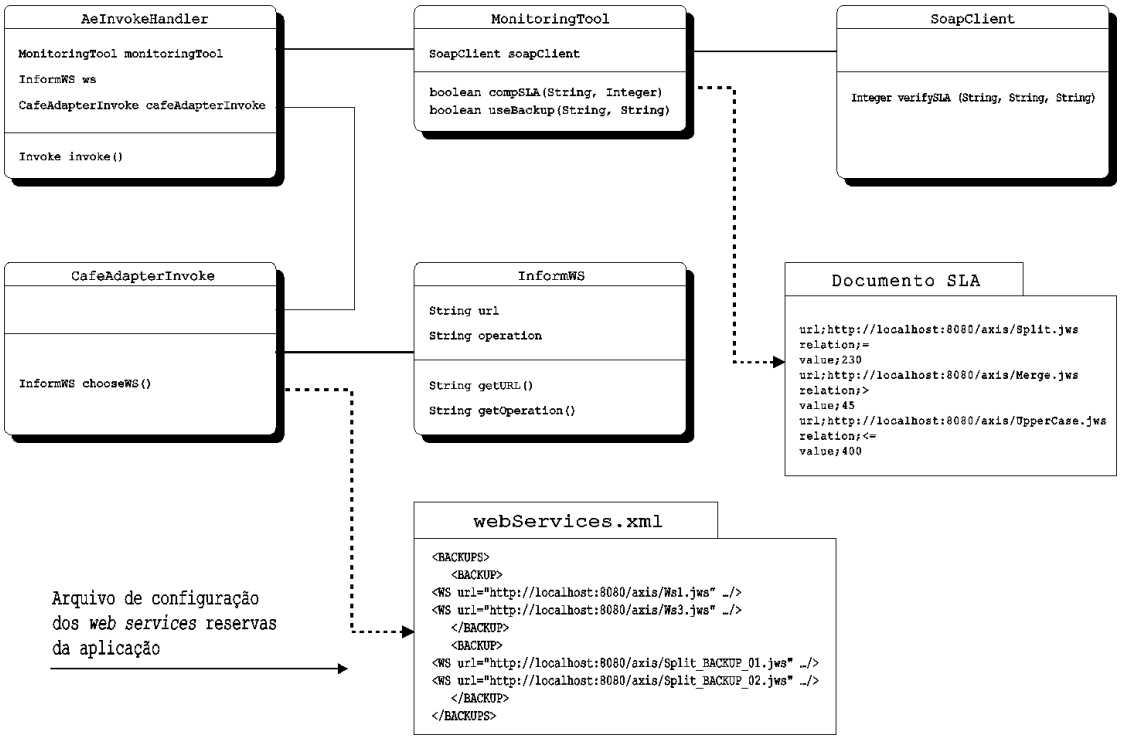

Figura 3-5. Diagrama de Classes da aplicação

A classe `AeInvokeHandler` é a classe modificada da *engine* ActiveBPEL. Nesta classe foram adicionados os objetos `MonitoringTool`, `InformWS` e `CafeAdapterInvoke`. As duas últimas fazem parte da *engine* A-ActiveBPEL (LINS, 2007). O relacionamento entre as classes do diagrama são todos do tipo associação um para um (1..1) no modelo UML (*Unified Modeling Language*). As linhas pontilhadas indicam ligações entre as classes e os arquivos de configuração dos *web services* reserva e do documento SLA. No contexto desta aplicação, todos os arquivos do projeto estão no mesmo servidor local da *engine* SLA-ActiveBPEL. Após a conclusão do projeto da aplicação, a próxima seção descreve a implementação das classes que foram definidas no modelo UML desta aplicação.

3.5. Implementação

Após a fase de projeto ter sido concluída, é possível então implementar os artefatos que serão necessários a modificação da *engine* proposta neste trabalho. 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 seqüenciamento (<sequence>), primitiva de concorrência (<flow>) e primitiva de iteração (<while>). Operações de entrada e saída são especificadas com o uso de duas primitivas, <receive> e <reply>, 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 BPEL, onde a primitiva de invocação é destacada.

```
(1)   <process name="UpperCase" ...>
(2)      <partnerLinks>
(3)   …
(4)      </partnerLinks>
(5)      <variables>
(6)   …
(7)      </variables>
(8)      <receive . . .>
(9)   …
(10)     </receive>
(11)     <reply . . .>
(12)  …
(13)     </reply>
(14)     <sequence>
(15)  …
(16)        <invoke operation="split" .../>
(17)        <invoke operation="uppercase" .../>
(18)     </sequence>
(19)  </process>
```

A sintaxe da primitiva de invocação <invoke> continua a mesma, contudo a extensão adicionada à *engine* verifica (monitora) os serviços oferecidos pelo *web service* presente no <invoke>. Antes de realizar a chamada para os *web services* participantes, a *engine* verifica se é necessário substituir o *web service*, caso seja necessário e o *web service* em questão possua um serviço reserva (definido no repositório de serviços reservas) a *engine* então substitui o *web service* do <invoke>.

A extensão SLA-ActiveBPEL é um conjunto de arquivos Java colocados juntos com o código-fonte da *engine* A-ActiveBPEL. Modificações foram feitas na classe

`AeInvokeHandler` da A-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 novas classes `MonitoringTool` e `SoapClient` foram introduzidas, além das classes `CafeAdapterInvoke` e `InformWS` já introduzidas pela A-ActiveBPEL. O método `useBackup()` (da classe `MonitoringTool`) é chamado imediatamente antes da invocação do *web service* para verificar através do `SoapClient` se o *web service* está atendendo aos requisitos definidos. Caso o resultado da requisição anterior seja positivo, isto é, seja necessária a substituição, então o método `adaptiveInvoke()` é chamado. Após esta etapa, o método `adaptiveInvoke()` realiza uma busca no Repositório (arquivo de configuração dos *web services* reservas, neste caso `WebServices.xml`) com o intuito de verificar se existem serviços reservas definidos para o *web service* analisado no momento. Finalmente, o método `adaptiveInvoke()` configura todas as variáveis de ambiente necessárias para redirecionar a chamada para o endereço do *web service* reserva que foi definido no arquivo do repositório já mencionado.

O fragmento de código a seguir apresenta parte da implementação da extensão SLA-ActiveBPEL, presente na classe `AeInvokeHandler`.

```
(1)   package org.activebpel.rt.axis.bpel;
(2)   import org.activebpel.cafe.CafeAdapterInvoke;
(3)   import org.activebpel.cafe.InformWS;
(4)   import org.activebpel.monitor.MonitoringTool;
(5)   public class AeInvokeHandler implements IAeInvokeHandler {
(6)   …
(7)   public IAeWebServiceResponse handleInvoke … {
(8)   …
(9)   boolean useBackup = false;
(10)  …
(11)  Call call = createCall( wsdlService, operation,
      endpointReference, addressHandling);
(12)  String originalURL = call.getTargetEndpointAddress();
(13)  String op = call.getOperation().getName();
(14)  useBackup = false;
(15)  MonitoringTool monitoringTool = new MonitoringTool();
```

```
(16)  useBackup =
      monitoringTool.useBackup(originalURL,TAG_OPERATION_CONTROL);
(17)  InformWS ws;
(18)  String tempURL = originalURL;
(19)  while (useBackup) {
(20)  ws = CafeAdapterInvoke.getInstance().adapterInvoke(tempURL,op);
(21)  call.setTargetEndpointAddress(ws.getURL());
(22)  tempURL = call.getTargetEndpointAddress();
(23)  useBackup =
      monitoringTool.useBackup(tempURL,TAG_OPERATION_CONTROL);
(24)  }
(25)  }
(26)  }
```

Inicialmente, a chamada é construída através do construtor `Call` (linha 11), 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 `CafeAdapterInvoke`. O retorno deste método especifica o *web service* definido pelo mecanismo adaptativo para funcionar como o *web service* reserva, que é armazenado em um objeto do tipo `InformWS`, criado pela extensão especialmente para armazenar os *web services* reservas (LINS, 2007). O código da classe `InformWS` é mostrado a seguir.

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

Esta classe armazena os dois atributos relevantes para a chamada de um *web service*: o seu endereço (url) na linha 3 (três) e a sua operação (operation) na linha 4 (quatro).

A classe `CafeAdapterInvoke` será agora detalhada. Esta classe executa uma das tarefas fundamentais no contexto da extensão SLA-ActiveBPEL: executa a seleção dos *web services* reservas. Inicialmente, é desejado que exista apenas uma instância desta classe em um determinado instante da 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. Outra é sempre utilizar um método `getInstance()`, que faz o controle da instanciação, criando um novo objeto apenas quando necessário (LINS, 2007). Abaixo segue o código fonte desta classe.

```
(1)   package org.activebpel.cafe;
(2)   import java.util.Iterator;
(3)   import java.io.File;
(4)   import org.jdom.Document;
(5)   import org.jdom.Element;
(6)   import org.jdom.JDOMException;
(7)   import org.jdom.input.SAXBuilder;
(8)   public class CafeAdapterInvoke {
(9)   private final String ATRIBURL = "url";
(10)  private final String ATRIBOPERATION = "name_operation";
(11)  private final String ATRIBOrdemDeploy = "number";
(12)  private static String NOME_ARQUIVO =
      "C:\\UFPE\\Graduacao\\2008.2\\Trabalho_Graduacao\\activebpel-
      2.0\\lib\\WebServices.xml";
(13)  private static SAXBuilder parser = null;
(14)  private static Document document = null;
(15)  private static CafeAdapterInvoke singleton = null;
(16)  private static File xmlFile ;
(17)  private static long lastModified = 0;
(18)  protected CafeAdapterInvoke(){}
(19)   // Verifica se o sistema está com a versão atualizada do arquivo
(20)  public static CafeAdapterInvoke getInstance(){
(21)  if (lastModified < xmlFile.lastModified()) {
```

```
(22)   try {
(23)   document = parser.build(NOME_ARQUIVO);
(24)   lastModified = xmlFile.lastModified();
(25)   } catch (JDOMException e) {
(26)   // TODO Auto-generated catch block
(27)   e.printStackTrace();
(28)   }
(29)   }
(30)   return singleton;
(31)   }
(32)   static {
(33)   singleton = new CafeAdapterInvoke();
(34)   parser = new SAXBuilder();
(35)   parser.setValidation(false);
(36)   xmlFile = new File(NOME_ARQUIVO);
(37)   try {
(38)   document = parser.build(NOME_ARQUIVO);
(39)   lastModified = xmlFile.lastModified();
(40)   } catch (JDOMException e) {
(41)   // TODO Auto-generated catch block
(42)   e.printStackTrace();
(43)   }
(44)   }
(45)   public InformWS adapterInvoke(String url, String operation){
(46)   Element grupos = document.getRootElement();
(47)   Iterator itGrupo = grupos.getChildren().iterator();
(48)   while (itGrupo.hasNext()) {
(49)   Element grupo = (Element) itGrupo.next();
(50)   Iterator itWS = grupo.getChildren().iterator();
(51)   while(itWS.hasNext()) {
(52)   Element WS = (Element) itWS.next();
(53)   if(WS.getAttributeValue(ATRIBURL).equalsIgnoreCase(url))
(54)   {
(55)if(WS.getAttributeValue(ATRIBOPERATION).equalsIgnoreCase(operation))
return chooseWS(grupo);
(56)   }
(57)   }
(58)   }
(59)   return new InformWS(url,operation);
(60)   }
(61)   private InformWS chooseWS(Element grupo){
(62)   Iterator itWS = grupo.getChildren().iterator();
(63)   Element WSmaior = (Element) itWS.next();
(64)   while(itWS.hasNext()){
(65)   Element WS = (Element) itWS.next();
```

```
(66)if(Integer.parseInt(WSmaior.getAttributeValue(ATRIBOrdemDeploy))<
Integer.parseInt(WS.getAttributeValue(ATRIBOrdemDeploy)))
(67)   WSmaior = WS;
(68)   }
(69)   }
(70)   return new InformWS(WSmaior.getAttributeValue(ATRIBURL),
WSmaior.getAttributeValue(ATRIBOPERATION));
(71)   }
(72)   public static void main(String[] args) {}
(73)   }
```

Ao se analisar a estrutura geral da classe, pode-se perceber que o método `getInstance()` (linha 20) 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 Repositório. Como será descrito posteriormente, 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 (novos *web services* reservas podem ter sido adicionados); se tiver, ele carrega as novas informações para análise de qual *web service* reserva será utilizado; caso contrário, ele simplesmente mantém a instância como está atualmente.

A classe `MonitoringTool` é responsável pelo monitoramento dos *web services* participantes da composição. Esta classe faz as requisições através das interfaces de controle dos *web services* e retorna para a *engine* se é necessário ou não fazer a substituição do *web service* que está sendo analisado. O seu código fonte é apresentado a seguir.

```
(1)   package org.activebpel.monitor;
(2)   public class MonitoringTool {
(3)         private SoapClient soapClient;
(4)         public MonitoringTool() {
(5)               this.soapClient = new SoapClient();
(6)         }
(7)         public boolean compSLA(String newURL, Integer slaValue) {
(8)               boolean ret = false;
(9)               if (slaValue >= 250) ret = true; else ret = false;
(10)              return ret;
(11)        }
(12)        public   boolean   useBackup(String   newURL   ,   String
            newOperationName) throws Exception {
(13)              Integer sla = 0;
(14)              boolean ret = false;
```

```
(15)              // Retorna o valor da métrica DEFAULT
(16)              sla     =     this.soapClient.verifySLA(newURL     ,
      newOperationName , "DEFAULT");
(17)  if (this.compSLA(newURL,sla)) ret = true; else ret = false;
(18)  return ret;
(19)  }
(20)  }
```

A classe `MonitoringTool` utiliza um cliente do protocolo SOAP para acessar as interfaces de controle dos *web services*. Este cliente SOAP é implementado na classe `SoapClient` apresentada abaixo.

```
(1)   package org.activebpel.monitor;
(2)   import org.apache.axis.client.Call;
(3)   import org.apache.axis.client.Service;
(4)   public class SoapClient {
(5)   public SoapClient() {}
(6)   public    Integer    verifySLA(String    newURL    ,    String
newOperationName , String newParam) throws Exception {
(7)   // Endereço, local onde encontra-se o Web Service
(8)   String url = newURL;
(9)   String operationName = newOperationName;
(10)  // Criando e configurando o serviço
(11)  Call callService = (Call) new Service().createCall();
(12)  // Configurando o endereço.
(13)  callService.setTargetEndpointAddress(url);
(14)  // Marcando o método a ser chamado.
(15)  callService.setOperationName(operationName);
(16)  // Parâmetros da função que obtém o SLA
(17)  Object[] param = new Object[]{new String(newParam)};
(18)  // Retorno da Função
(19)  Integer ret = (Integer)callService.invoke(param);
(20)  return ret;
(21)  }
(22)  }
```

A implementação de um cliente SOAP é necessário para realizar a comunicação da ferramenta de monitoramento com os *web services* da composição. Ela foi desenvolvida de uma forma que os nomes dos métodos (assinaturas) da interface de controle podem

ser modificados e as métricas de controle de qualidade (definida no parâmetro destes métodos) parametrizáveis.

3.6. Repositório de *web services* reservas

Como mencionado anteriormente, a extensão SLA-ActiveBPEL tem um repositório que provê informações sobre os *web services* reservas participantes (como endereço e operações). 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 (LINS, 2007).

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 possuírem serviços reservas são colocados em um grupo denominado `<backup>` (*tag* do XML). Quando a *engine* analisa a especificação WS-BPEL, ela identifica qual *backup* pertence ao *web service* analisado. Nesta estratégia, existe a necessidade de especificar a localização do *web service* reserva. Todo *web service* inserido no repositório deve ter pelo menos um serviço reserva.

O código abaixo apresenta um exemplo de especificação de *web services backup*.

```
(1)<BACKUPS>
(2)    <BACKUP>
(3)    <WS url="http://localhost:8080/axis/Ws1.jws" …/>
(4)    <WS url="http://localhost:8080/axis/Ws3.jws" …/>
(5)    </BACKUP>
(6)
(7)    <BACKUP>
(8)    <WS url="http://localhost:8080/axis/Split_BACKUP_01.jws" …/>
(9)    <WS url="http://localhost:8080/axis/Split_BACKUP_02.jws" …/>
(10)    </BACKUP>
(11)</BACKUPS>
```

Esta é uma maneira prática e intuitiva de se explicitar os *web services* e os seus reservas. Além de ser um arquivo texto, o mesmo segue um padrão largamente utilizado (XML) com muitas implementações para manipular esta linguagem. No exemplo acima, o *web service* localizado na URL `"http://localhost:8080/axis/Ws3.jws"` é o serviço

reserva do *web service* original "`http://localhost:8080/axis/Ws1.jws`". No contexto da proposta da solução implementada, o endereço completo é utilizado como referência para a substituição dos *web services*.

3.7. Considerações Finais

Este capítulo apresentou detalhes da *engine* SLA-ActiveBPEL. Inicialmente apresentamos os requisitos necessários ao projeto para possibilitar a definição da arquitetura. Em seguida, iniciamos o processo de desenvolvimento do projeto, mostrando os diagramas UML das classes e arquivos envolvidos na construção da aplicação. Por fim, foi mostrada as principais partes das implementações (códigos fontes) dos artefatos da solução.

Capítulo 4

Exemplo

"O que se aprende profundamente, jamais se esquece."

L. A. Sêneca – *Epístolas, 92*

Este capítulo a apresentar o exemplo utilizado para demonstrar a viabilidade da abordagem proposta. Uma aplicação (composição de *web services*) foi implementada neste contexto, o *UpperCase* Distribuído. O *UpperCase* Distribuído foi desenvolvido tendo em vista o aspecto didático, apresentando um caso bastante simplificado de composição de *web services*.

4.1. O *UpperCase* Distribuído

Implementado especificamente para a utilização nesse trabalho, o *UpperCase* Distribuído é conceitualmente bastante simples (Allen, 1997). O processo de negócio é composto por três *web services* e cada um possui uma função específica na composição. Esta aplicação recebe uma única entrada (*string*) e retorna este mesmo *string* com seus caracteres em maiúsculos. Embora seja um serviço conceitualmente muito simples, vale ressaltar que a complexidade de implementação do mesmo não é relevante no contexto deste trabalho, pois esta complexidade não é levada em consideração pela *engine* SLA-ActiveBPEL no instante da substituição do *web service* pelo serviço reserva.

Os *web services* participantes da composição são:

- ***Web service* 1 – *Split***. Este *web service* recebe como entrada uma cadeia de caracteres (*string*) de qualquer tipo, com exceção de caracteres especiais ou com acentuação. Um método denominado `split()` recebe essa entrada e retorna como resultado duas cadeias de *string* que são

simplesmente o particionamento da cadeia original. Como exemplo uma cadeia de entrada "aabbcc" tem como saída duas cadeias, a primeira "aab" e a segunda "bcc".

- ***Web service* 2 – UpperCase**. Este *web service* recebe como entrada duas cadeias de caracteres e como resultado retorna as mesmas duas cadeias com os caracteres maiúsculos (quando for o caso). Como exemplo uma entrada com as cadeias, "bbA34" e "AA12vv" retorna como saída as cadeias "BBA34" e "AA12VV", respectivamente.
- ***Web service* 3 – Merge**. Finalmente, o último participante da composição é responsável por receber duas cadeias de caracteres e retornar uma única cadeia resultante da concatenação das duas entradas. Como exemplo a entrada formada pelas cadeias "aaaXXX" e "Uiiqw7" retornará como resultado a cadeia "aaaXXXUiiqw7".

A Figura 4-1 representa a composição realizada com os *web services*.

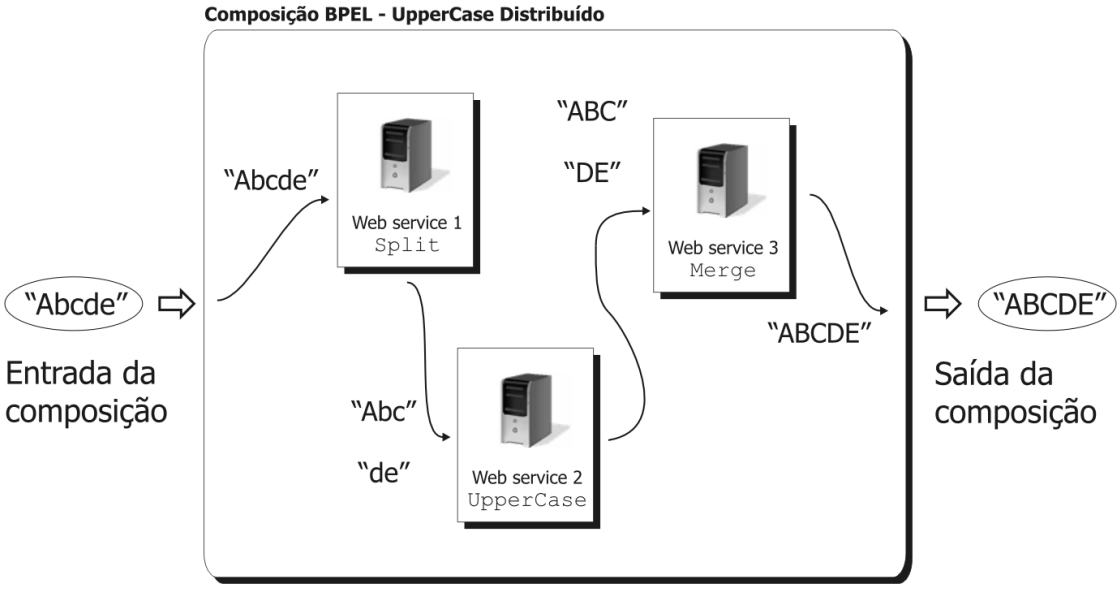

Figura 4-1. Funcionamento da composição *UpperCase* distribuído

Para que estes *web services* possam ser utilizados na composição, primeiramente devemos implementar e posteriormente disponibilizá-los na *Web*, este é o processo de desenvolvimento de um *web service*. Estas duas etapas serão detalhadas nas próximas seções.

4.2. Desenvolvimento dos *web services*

No contexto deste trabalho, os *web services* para o exemplo, foram desenvolvidos utilizando a linguagem Java. A escolha da linguagem Java é motivada apenas pela forma como o *web service* é disponibilizado na *Web*. O processo de publicação (*deployment*) do servidor (*web service*) na *Web* será descrito na próxima seção.

A seguir o código fonte de um dos *web services* utilizados na composição.

```
(1)public class Split {
(2)
(3)   public Split() {
(4)
(5)   }
(6)
(7)   public String split(String word) {
(8)         String ret = null;
(9)         Integer middle = word.length() / 2;
(10)        String part1 = word.substring(0, middle);
(11)        String part2 = word.substring(middle);
(12)        ret = part1 + ";" + part2;
(13)        return ret;
(14)  }
(15)
(16)  public Integer getSLA(String param) {
(17)        Integer ret = 0;
(18)        if (param.equals("DEFAULT")) {
(19)               ret =  1+(int)(500*Math.random());
(20)        }
(21)        return ret;
(22)  }
(23)
(24)}
```

Neste código pode ser visto a implementação do *web service* `Split`. Dois métodos foram mostrados, primeiramente o `split()`(linha 7), responsável pelo algoritmo de particionamento das cadeias de entrada, serviço principal deste *web service*. Depois é visto o método `getSLA()` (linha 16), esta é a função responsável pela implementação da interface de controle do *web service*. A função de controle (`getSLA`), neste exemplo, é simplesmente um gerador aleatório de números inteiros. O objetivo é mostrar que esta

função pode ser qualquer métrica definida pelos documentos SLA. No caso proposto deste exemplo, os valores obtidos com este método de controle não representam nenhum valor com semântica definida. Devido ao caráter didático desta demonstração tal conteúdo da interface de controle são números inteiros entre 1 (um) e 500 (quinhentos). No entanto, para trabalhos futuros e implementações mais elaboradas desta solução, o conteúdo desta função de controle deve atender àqueles requisitos definidos nas especificações dos documentos SLA.

Os outros *web services* possuem, cada um, um método responsável pelo seu serviço principal e também implementam a mesma função (`getSLA`) de controle. Os códigos fontes de todos os *web services* utilizados neste exemplo estão disponíveis no Apêndice deste trabalho.

É conveniente também observar que os *web services* reservas definidos para os *web services* principais, foram desenvolvidos da mesma forma e são uma cópia dos códigos fontes dos seus *web services* principais (ver o apêndice). Logo, os serviços reservas não possuem nenhuma característica diferente ou foram publicados na *Web* de outra forma que seus *web services* principais. Na próxima seção será apresentado o modelo de publicação (*deployment*) dos *web services* na *Web*.

4.3. Publicação (*deployment*) de um *web service*

Existem métodos diferentes de se publicar (*deploy*) *web services* na *Web*, tais como através da utilização de servidores de aplicação e de *frameworks*. Como exemplo de servidor de aplicação existe o .NET (Microsoft). No presente trabalho será utilizado o *framework* Axis (Apache, 2008). Este *framework* tem a vantagem de tornar bastante simples a disponibilização dos *web services*. O principal objetivo do Axis é prover a comunicação entre os *web services* através do protocolo SOAP. O Axis permite a criação automática dos arquivos WSDL dos *web services* e possui uma forma simplificada de utilização, bastando colocar os arquivos fontes (Java) da aplicação (*web service*) em um diretório específico da ferramenta. Depois deste passo o *web service* já se encontra disponível para a utilização.

O Axis necessita apenas de um servidor *web* para aplicações desenvolvidas em Java. Neste caso foi utilizado o Apache Tomcat (Apache, 2008). Após a instalação do Axis no servidor Tomcat e da publicação (*deployment*) dos *web services* através do Axis, os *web services* deste exemplo já estão prontos para serem utilizados nas composições. Na

próxima seção será descrito como é desenvolvida a composição de *web services* e sua publicação na *engine* SLA-ActiveBPEL.

4.4. Construção da composição *UpperCase* Distribuído

O processo de criação de uma composição de *web services* (codificação do arquivo BPEL) realizada sem o auxílio de uma ferramenta automática, pode gerar muitos erros e dificultar o desenvolvimento de composições. Por este motivo, foi utilizada, neste trabalho, uma ferramenta para criar, testar e publicar as composições de forma automática, evitando assim inúmeros erros na fase do projeto da composição. O ActiveBPEL Designer (ActiveBPEL, 2008) é um software destinado a criar composições baseadas no padrão WS-BPEL. Ele também gera todos os arquivos necessários a publicação de uma composição na *web* (.bpr, .pdd, .wsdl e .bpel) facilitando o processo de composição dos *web services*. Além dessas facilidades, o ActiveBPEL Designer gera todos os arquivos com o objetivo de serem utilizados pela *engine* ActiveBPEL, desta forma evita-se falhas motivadas por incompatibilidade de ferramentas. O ActiveBPEL Designer foi desenvolvido como um *plugin* da IDE Eclipse. A Figura 4-2 mostra a interface gráfica do software ActiveBPEL Designer e um exemplo de composição.

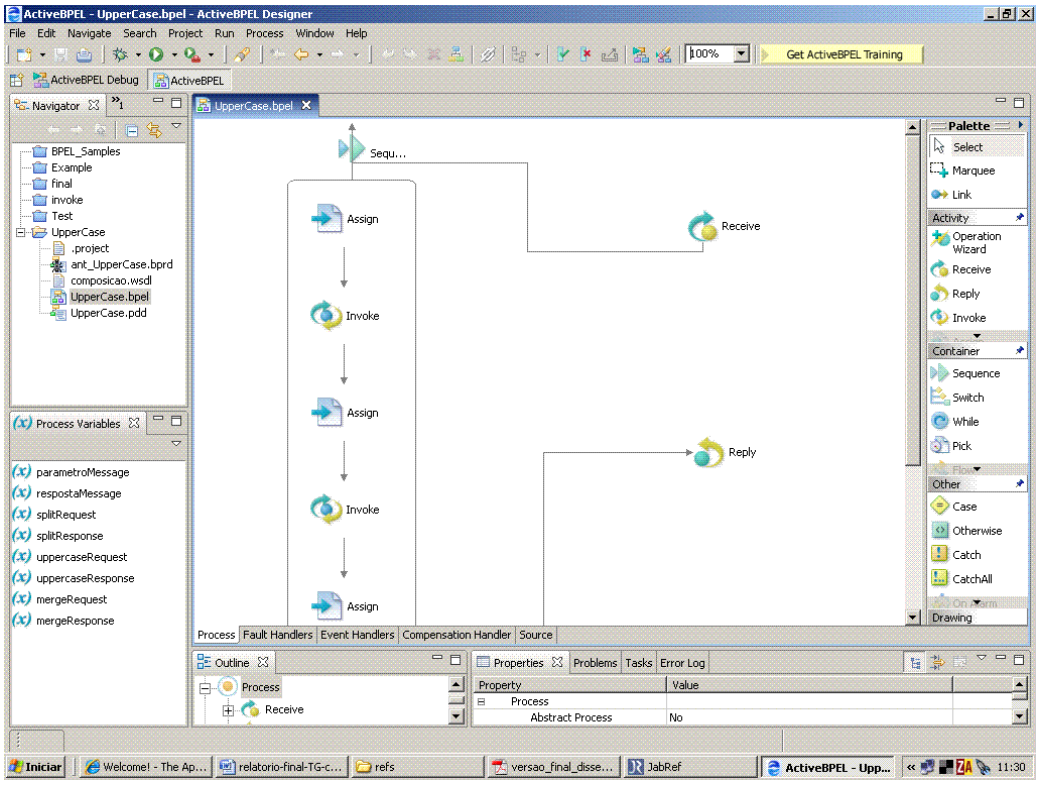

Figura 4-2. Ferramenta ActiveBPEL Designer

A ferramenta ActiveBPEL Designer cria as composições utilizando uma interface gráfica que auxilia processo. No final da composição, a ferramenta gera os arquivos .bpel e .wsdl e ainda cria automaticamente os arquivos para a publicação (*deployment*) automática da composição na *engine* SLA-ActiveBPEL. A composição do exemplo deste capítulo (*UpperCase* Distribuído) desenvolvido pela ferramenta é vista na Figura 4-3.

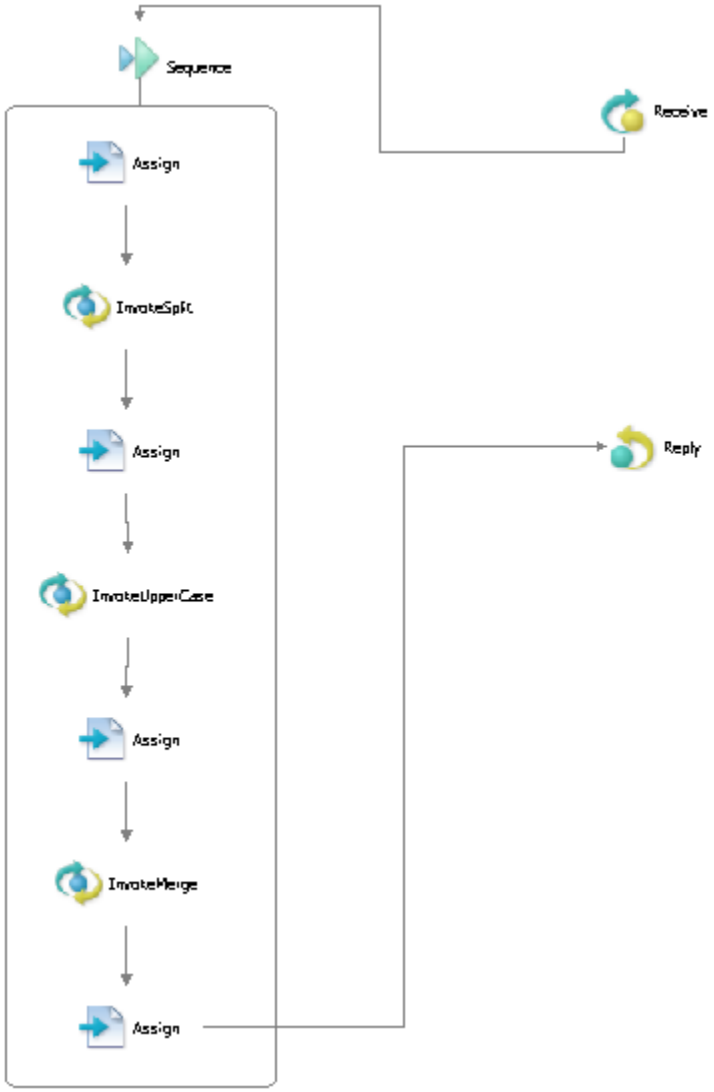

Figura 4-3. Composição do *UpperCase* Distribuído

NaFigura 4-3 é apresentado a composição *UpperCase* Distribuído. Os três elementos principais `invokeSplit`, `invokeUpperCase` e `invokeMerge` são mostrados. Eles são responsáveis pelas chamadas (invocação) dos *web services* da composição. Na próxima seção será apresentado o processo de utilização da *engine* SLA-ActiveBPEL e conseqüentemente do *UpperCase* Distribuído.

4.5. Utilização da *engine* SLA-ActiveBPEL

Após a publicação dos *web services* (inclusive os reservas) na *Web* através do *framework* Axis e do servidor *web* Tomcat, mais a definição da composição *UpperCase* com o auxílio da ferramenta ActiveBPEL Designer e conseqüentemente da sua publicação (*deployment*) na *engine* SLA-ActiveBPEL, será apresentado o processo de utilização do exemplo proposto. O ambiente, em particular a *engine* SLA-ActiveBPEL, estão prontos para receber as requisições dos clientes e executar o processo de negócios definido na composição. Como já mostrado anteriormente, a composição funciona exatamente como um *web service* e portanto, se comunica com os clientes ou outros *web services* através do protocolo SOAP.

É possível utilizar duas estratégias para fazer requisições à composição *UpperCase* Distribuído. A primeira utiliza o próprio navegador (*browser*), através de uma sintaxe específica, para enviar mensagens à composição. A outra forma utiliza a implementação de um cliente SOAP. O trabalho apresentará as duas formas.

4.5.1. Requisições via navegador (*browser*)

Ao se publicar uma composição na *engine* SLA-ActiveBPEL, uma interface WSDL é automaticamente gerada e possibilita então que os clientes conheçam os parâmetros de entrada e as assinaturas das requisições da composição. No exemplo deste trabalho, o *UpperCase* Distribuído possui uma sintaxe bastante simples. A mensagem de requisição e o seu parâmetro são descritas a seguir:

- Mensagem de requisição: `request`
- Parâmetro de entrada: `word`

O exemplo a seguir mostra um exemplo da sintaxe de uma requisição à composição através do navegador:

- `http://localhost:8080/active-bpel/services/uppercaseService?method=`**`request`**`&`**`word`**`=jhjhkj`

A Figura 4-4 mostra a requisição realizada no navegador. No exemplo, está sendo utilizado para a demonstração o navegador Internet Explorer (Microsoft). No entanto, qualquer navegador pode ser utilizado para realizar as requisições aqui descritas.

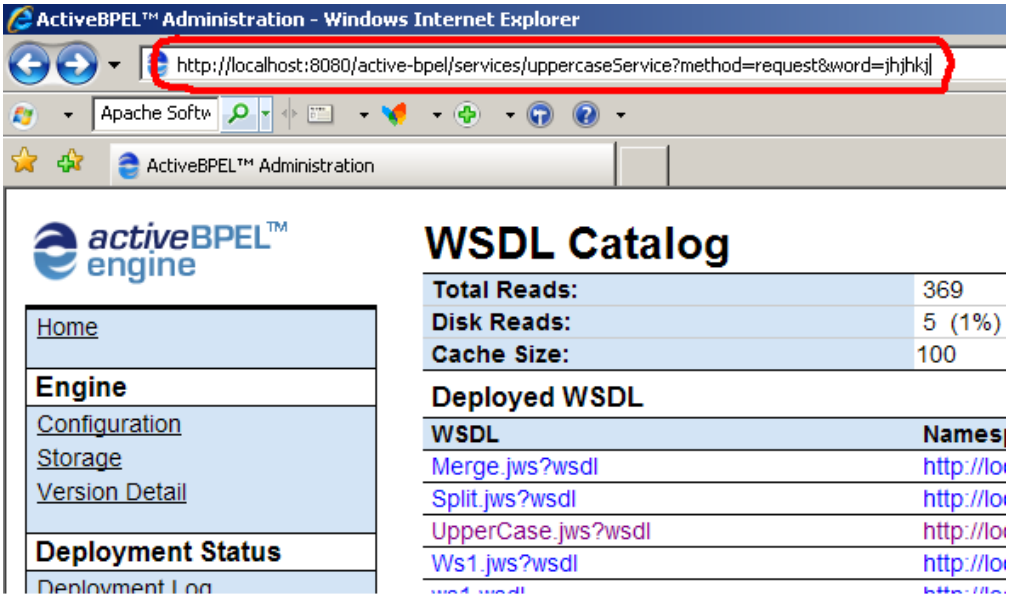

Figura 4-4. Requisição realizada através do navegador

Após a requisição feita pelo navegador à composição, a *engine* SLA-ActiveBPEL executa o processo de negócios definido pelo arquivo .bpel e retorna o resultado da requisição (o texto original de entrada em letras maiúsculas) para o mesmo navegador, utilizando o protocolo SOAP. Veja a Figura 4-5.

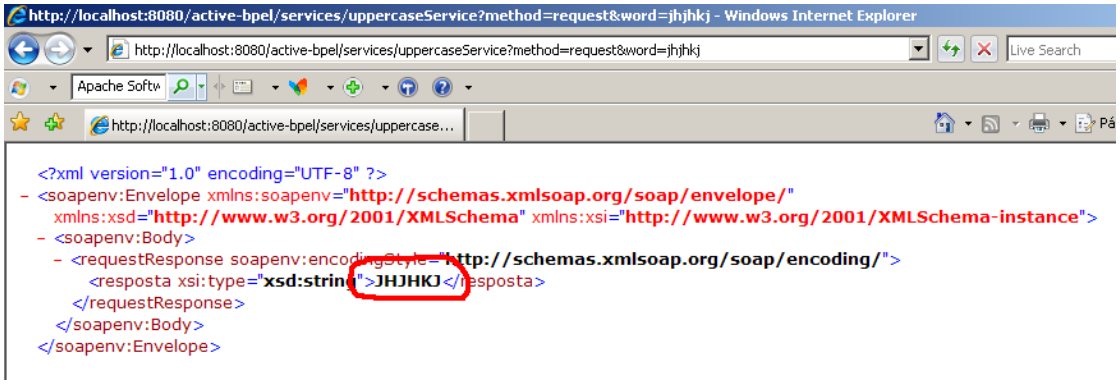

Figura 4-5. Resposta da composição à requisição

A Figura 4-5 mostra a resposta da composição. Um texto no formato XML é enviado para o navegador. Nele é mostrado o envelope SOAP.

O processo descrito anteriormente mostra o funcionamento de uma requisição feita à *engine* SLA-ActiveBPEL. O resultado foi obtido e não se verificou qualquer alteração no comportamento da *engine* ou na sintaxe dos protocolos. No entanto, antes das chamadas aos *web services* da composição a *engine* SLA-ActiveBPEL realizou o monitoramento da execução do processo de negócios e registrou as ações em um *log* de

atividades. Uma funcionalidade foi acrescentada a *engine* para prover este mecanismo de *log*. Ele pode ser visto a seguir.

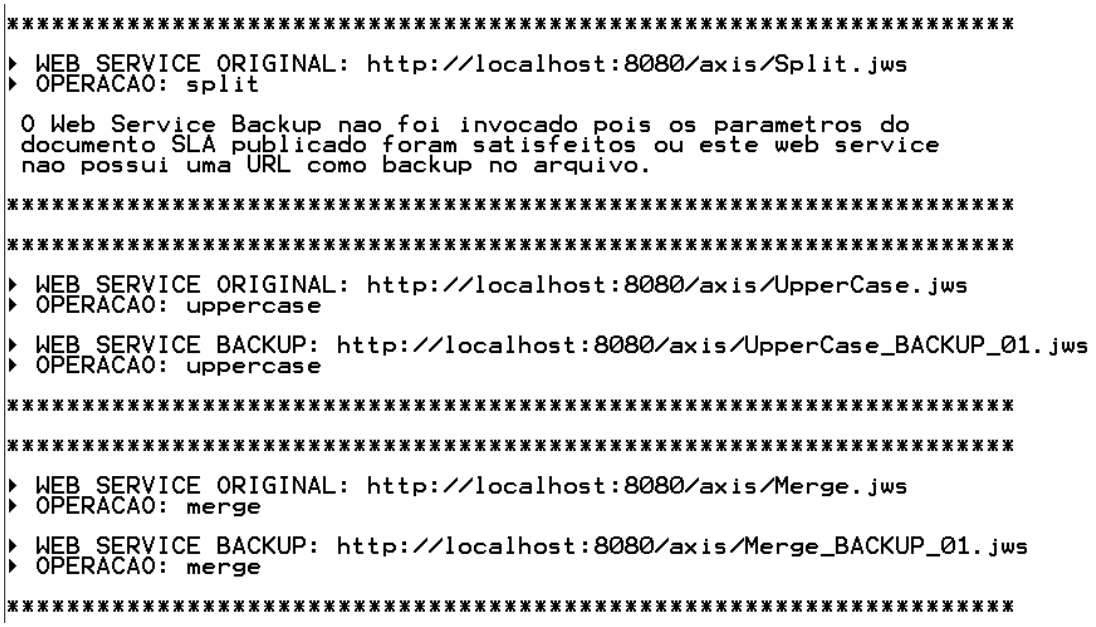

Figura 4-6. *Log* de atividades da *engine* SLA-ActiveBPEL

Na Figura 4-6 é possível verificar que, neste exemplo, após as comparações dos dados obtidos através da interface de controle, os *web services* `UpperCase` e `Merge` foram substituídos pelos seus reservas, `UpperCase_BACKUP_01` e `Merge_BACKUP_01` respectivamente. Já o *web service* `Split` não foi substituído devido aos parâmetros do documento SLA terem sido satisfeitos ou ele (o *web service*) não possuir um serviço reserva no arquivo de configuração.

4.5.2. Requisições via cliente SOAP

Para o propósito da demonstração da *engine* SLA-ActiveBPEL, também foi desenvolvido um cliente SOAP que realiza várias requisições à composição. A idéia da implementação deste cliente é mostrar que a *engine* proposta também se mostra adequada em um ambiente de múltiplas requisições. Para tal objetivo foi codificado um programa em Java que realiza várias requisições SOAP à composição *UpperCase* Distribuído. Neste caso as requisições e respostas não utilizam o navegador, todas as mensagens são trocadas entre o cliente SOAP e a composição *UpperCase* Distribuído. A seguir o código fonte deste cliente.

```
(1) import org.apache.axis.client.Call;
```

```
(2) import org.apache.axis.client.Service;
(3) import org.apache.axis.AxisFault;
(4) import java.util.HashMap;
(5) import java.math.BigInteger;
(6) import javax.xml.rpc.ParameterMode;
(7)
(8) public class BPELTestClient {
(9)
(10) public static void main(String[] args) {
(11)    try {
(12)         BPELTestClient client = new BPELTestClient();
(13)         HashMap map = new HashMap();
(14)
(15)         for(int i = 0 ; i < 10 ; i = i + 1) {
(16)
(17)         map.put("url","http://localhost:8080/active-
             bpel/services/uppercaseService");
(18)         map.put("operation","request");
(19)         map.put("word","texto "+i);
(20)
(21)         String result = client.call(map);
(22)         System.out.println(result);
(23)
(24)         }
(25)
(26)    } catch (Exception e) {
(27)         e.printStackTrace();
(28)    }
(29)}
(30)
(31) public BPELTestClient() throws Exception {}
(32)
(33) public String call(HashMap map)throws Exception {
(34)   Service service = new Service();
(35)   Call call = (Call) service.createCall();
(36)   String url = (String) map.get("url");
(37)   String operation = (String) map.get("operation");
(38)   call.setTargetEndpointAddress(new java.net.URL(url));
(39)   call.setOperationName(operation);
(40)    call.addParameter("word",
org.apache.axis.Constants.XSD_STRING,ParameterMode.IN);
(41)   call.setReturnType(org.apache.axis.Constants.XSD_STRING);
(42)
(43)    String result = null;
(44)
```

```
(45)   try {
(46)   result = (String) call.invoke(new Object[] {
(47)                        ((String) map.get("word")),});
(48)
(49)     } catch (AxisFault af) {
(50)          System.out.println("Catch 1");
(51)          result = af.toString();
(52)     } catch (Exception e) {
(53)          System.out.println("Catch 2");
(54)          result = "Erro Inesperado: " + e.toString();
(55)     }
(56)
(57)   return result;
(58)
(59)  }
(60) }
```

As linhas principais deste programa apresentam a definição da localização da URL do serviço (linha 17), o nome da mensagem de requisição (linha 18) e o nome do parâmetro de entrada (linha 19). A chamada da composição *UpperCase* Distribuído (*web service*) utilizando o método `invoke` se encontra na linha 46. A Figura 4-7 apresenta a resposta (saída) após a execução do programa cliente.

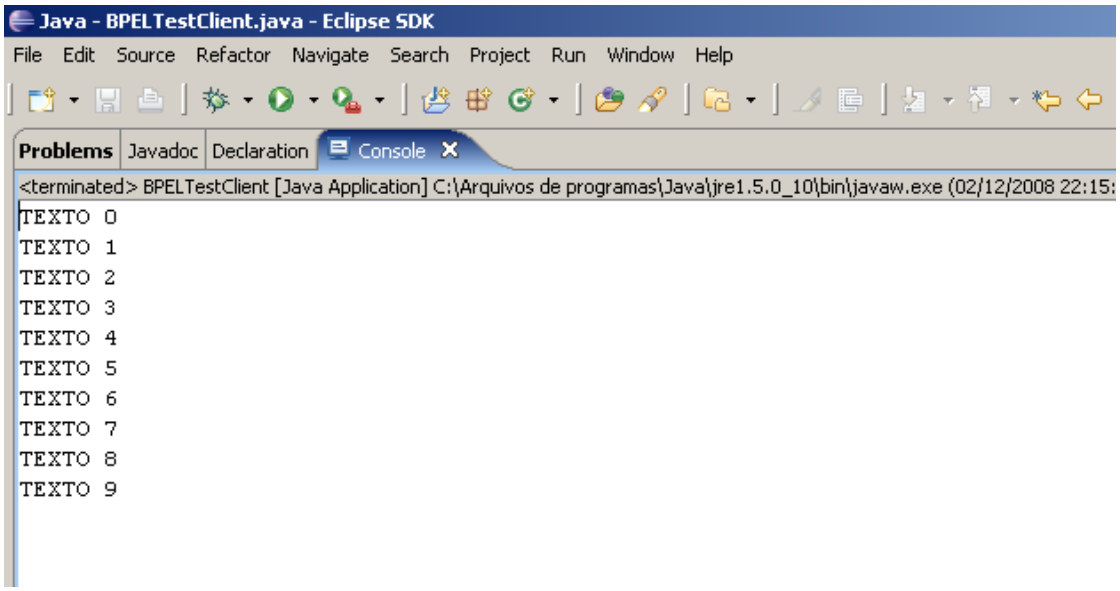

Figura 4-7. Saída da execução do cliente SOAP.

Para o desenvolvimento e execução do programa foi utilizado a IDE Eclipse. Cada linha da resposta verificada na Figura 4-7 é a resposta de uma requisição à composição *UpperCase* Distribuído. Veja no arquivo fonte do cliente SOAP (linha 19) que as cadeias de entrada nas requisições são formadas pela palavra "`texto`" seguida de um número inteiro entre 0 (zero) e 9 (nove).

Como na seção anterior (Requisições via navegador), o processo descrito anteriormente mostra o funcionamento de algumas requisições direcionadas à *engine* SLA-ActiveBPEL. Os resultados foram mostrados e não se verificou qualquer alteração no comportamento da *engine* ou na sintaxe dos protocolos. No entanto, antes das chamadas aos *web services* da composição a *engine* SLA-ActiveBPEL realizou o monitoramento da execução do processo de negócios e registrou as ações em um *log* de atividades. Neste caso não será mostrado todo o *log* devido a quantidade de requisições. Um fragmento dele pode ser visto a seguir.

```
**********************************************************************

▸ WEB SERVICE ORIGINAL: http://localhost:8080/axis/Split.jws
▸ OPERACAO: split

▸ WEB SERVICE BACKUP: http://localhost:8080/axis/Split_BACKUP_01.jws
▸ OPERACAO: split

**********************************************************************

**********************************************************************

▸ WEB SERVICE ORIGINAL: http://localhost:8080/axis/UpperCase.jws
▸ OPERACAO: uppercase

 O Web Service Backup nao foi invocado pois os parametros do
 documento SLA publicado foram satisfeitos ou este web service
 nao possui uma URL como backup no arquivo.

**********************************************************************

**********************************************************************

▸ WEB SERVICE ORIGINAL: http://localhost:8080/axis/Merge.jws
▸ OPERACAO: merge

▸ WEB SERVICE BACKUP: http://localhost:8080/axis/Merge_BACKUP_01.jws
▸ OPERACAO: merge

**********************************************************************
```

Figura 4-8. *Log* da *engine* SLA-ActiveBPEL com requisições via cliente SOAP

Na Figura 4-8 é mostrado o último bloco do *log* da *engine* SLA-ActiveBPEL para as requisições realizadas pelo cliente SOAP. Como pode ser observado não existe diferença no comportamento da *engine* para requisições múltiplas ou únicas, como também não importa qual método de requisição esteja sendo utilizado.

4.6. Considerações Finais

Neste capítulo foi apresentado o modelo de utilização da aplicação desenvolvida para avaliar a abordagem proposta. Para tal, foi construído o exemplo *UpperCase* Distribuído. Foram realizadas requisições à composição e foi possível verificar a viabilidade da ferramenta ao se visualizar as trocas dos *web services* em tempo de execução da *engine* SLA-ActiveBPEL. Para a realização das requisições foram utilizados dois métodos: requisições diretamente via navegador e através de uma implementação de um cliente SOAP.

Capítulo 5

Trabalhos Relacionados

"O momento é sempre adequado para fazer o certo."
Martin Luther King

Neste capítulo são apresentados os trabalhos relacionados ao tema deste trabalho de graduação. Em especial, na área de composição adaptativa de *web services*, duas correntes vem tendo 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. Concluindo o capítulo, também é mostrada outra abordagem através da Composição Adaptativa de *web services*.

5.1. Alteração Sintática do Padrão WS-BPEL

A proposta de alteração sintática do padrão WS-BPEL (LINS, 2007) é baseada na proposição de uma extensão 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*. 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 (LINS, 2007).

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 (LINS, 2007). 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. 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 ou 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 ou desativada em tempo de execução, a depender das condições do momento. (LINS, 2007).

5.3. Composição Adaptativa de *web services*

Esta abordagem apresenta uma alternativa para composição adaptativa de *web services* em relação às outras introduzidas nas seções anteriores, 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 nesta abordagem, 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*, esta abordagem propõe uma extensão para uma das *engines* mais difundidas e de código aberto que dão suporte ao WS-BPEL (mais especificamente, ActiveBPEL). 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 desenvolvedores.

A abordagem do presente trabalho foi baseada nesta última seção de trabalhos relacionados. A utilização de composições dinâmicas de *web services* já foi descrita na implementação de uma *engine* adaptativa (A-ActiveBPEL) em Lins (LINS, 2007), que por sua vez foi baseada na *engine* ActiveBPEL. Esta adaptabilidade e os seus conceitos foram utilizados neste trabalho. As principais contribuições adicionadas ao trabalho de Lins (LINS, 2007) foram a utilização dos conceitos de monitoramento da qualidade (requisitos) dos *web services* através de documentos SLA, a implementação de uma interface de controle nos *web services* e por fim propor uma modificação na *engine* A-ActiveBPEL para substituição dos serviços (*web services*) sem conformidade com os documentos SLA por outro serviço reserva.

5.4. Considerações Finais

Este capítulo apresentou os principais trabalhos relacionados ao tema deste trabalho de graduação. Uma visão geral de cada um deles foi apresentada. De forma resumida, três correntes distintas foram identificadas na área de composição adaptativa de *web services*: uma liderada pela proposição do mecanismo "*find and bind*" (Alteração Sintática do Padrão WS-BPEL), outra com a utilização de orientação a aspectos e finalmente a utilização do conceito de adaptação dinâmica de composições de *web services*.

Capítulo 6

Conclusões e Trabalhos Futuros

"Nenhum problema está concluído, antes que o concluamos bem."

Ella Wilcox – Settle the Question Right

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

6.1. Conclusões

A 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 (LINS, 2007). Além disto, a enorme proliferação dos *web services* e dos chamados processos de negócio (que são baseados em composições desses mesmos *web services*) mostram a necessidade de se desenvolver uma metodologia ou um padrão (referência) de gerenciamento que possam ser utilizados para o controle de qualidade (requisitos) na oferta dos serviços distribuídos.

A utilização de documentos SLA como parâmetro para verificar a conformidade dos requisitos estabelecidos entre os serviços na área de composição de *web services* ainda é um campo em estudo, onde estão abertas diversas possibilidades de propostas.

Devido às constantes mudanças que ocorrem com a disponibilização, a modificação e a exclusão de serviços, é extremamente desejável que os sistemas tenham a capacidade de verificar se os serviços ofertados estão mantendo os requisitos previamente acordados entre fornecedor e cliente e, se possível, manter a continuidade dos serviços sem causar maiores danos.

Este trabalho teve como principal objetivo propor uma abordagem para a composição reconfigurável (adaptável) de *web services* em tempo de execução. A proposta se mostrou válida e 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 propõem alterações sintáticas em padrões já extensamente difundidos e aceitos (como o WS-BPEL), ou necessitam de alterações no código-fonte original da aplicação (diminuindo a transparência da inserção da substituição dos *web services* no projeto) e finalmente, requerem que os desenvolvedores envolvidos no projeto tenham que adquirir conhecimentos que podem não estar ligados diretamente ao sistema (como a orientação a aspectos).

Embora tenha sido utilizada a *engine* de código aberto ActiveBPEL, mais especificamente a *engine* A-ActiveBPEL, qualquer outra poderia ser escolhida utilizando as idéias contidas no capítulo três deste trabalho. Outro ponto a ser ressaltado é que todas as características desta *engine* foram preservadas, sendo apenas adicionadas mais duas características: o monitoramento dos *web services* participantes da composição através de comparação de documentos SLA e a substituição dos *web services* em tempo de execução. Não houve nenhuma perda de funcionalidade da *engine* em detrimento da adoção do monitoramento e reconfiguração dos *web services*.

Os resultados obtidos através do exemplo apresentado permitem afirmar que a extensão SLA-ActiveBPEL proposta implementou corretamente o monitoramento dos *web services* e realizou a troca dos mesmos quando necessário, possibilitando que a *engine* pudesse se reconfigurar (adaptar) ainda em tempo de execução, baseada em critérios pré-estabelecidos pelos documentos SLA e em uma política de substituição dos *web services*. Adicionalmente, a idéia de procurar mudanças imediatamente antes das chamadas dos *web services* participantes acaba permitindo que as substituições 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.

Tendo em vista o possível *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 dos arquivos de configuração dos serviços reservas (*backup*) se localizarem no mesmo servidor da *engine* (o que diminui consideravelmente custos de comunicação) e por este arquivo se tratar, no caso deste

trabalho, de um simples arquivo texto no padrão XML (sistemas gerenciadores de banco de dados mais complexos poderiam inferir um maior custo de acesso).

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 adição de um monitoramento por parte da *engine* na qualidade (requisitos) dos serviços oferecidos pelos *web services* participantes através da adoção de documentos SLA e também uma eventual substituição dos mesmos em tempo de execução. Isto foi possível através da inserção de modificações no instante da chamada da primitiva de invocação do padrão WS-BPEL (`<invoke>`) e do desenvolvimento de *web services* com uma interface de controle que permitisse a obtenção dos dados para as comparações através do documento SLA. Desta forma permitiu-se que a *engine* adquirisse um comportamento ativo diante de falhas nos componentes da composição, sem a necessidade de alterações sintáticas e de utilização de outras tecnologias que poderiam retardar o tempo de desenvolvimento do processo de negócio.

A substituição de *web services* participantes é 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 simples e intuitiva de especificar os serviços reservas. Sem requisitar a utilização de uma base de dados mais complexa, apenas utilizando uma sintaxe simples baseada em XML e definindo quais *web services* serão os reservas da composição, os desenvolvedores poderão atualizar este arquivo sem encontrar maiores dificuldades na realização desta tarefa.

Adicionalmente, foi implementada uma extensão da *engine* A-ActiveBPEL, denominada SLA-ActiveBPEL, a qual poderá ser utilizada para a disponibilização do monitoramento 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 elas terão indicações de aperfeiçoamento na seção de trabalhos futuros. Inicialmente, a interface dos *web services* que servirão de reserva (*backup*) precisam ser sintaticamente equivalentes aos presentes na composição. Esta limitação se baseia na idéia de que a *engine* utilizará a mesma sintaxe de mensagens SOAP para a invocação das operações requisitadas nos novos *web services* reservas. Se a interface usada pelo *web service* reserva for diferente da sintaxe do serviço atualmente utilizado, a mensagem SOAP não conseguirá invocar o serviço desejado. O termo equivalente quer dizer, mais especificamente, que essas interfaces deverão ter o mesmo nome (assinatura) para as suas operações e parâmetros.

A inserção de uma interface de controle nos *web services* participantes da composição, inclusive nos reservas, limita o uso da *engine* proposta. Esta interface de controle, apesar de fazer parte da descrição WSDL do serviço, está presente no *web service* apenas com o objetivo de prover os dados que serão utilizados pelo mecanismo de comparação dos documentos SLA. Esta limitação pode ser eliminada através de outra forma de verificação dos requisitos de qualidade do *web service*, esta outra forma será discutida na próxima seção.

6.4. Trabalhos Futuros

Diversos trabalhos futuros são vislumbrados no contexto deste trabalho, alguns inclusive já iniciados, como a Composição Adaptativa de *web services* (LINS, 2007). Inicialmente, o desenvolvimento de um padrão de gerenciamento da verificação de conformidade de serviços entre os sistemas parceiros deve ser proposto, no contexto deste trabalho utilizamos os documentos SLA junto com o monitoramento. Esta necessidade vem do fato da proliferação dos *web services* e de uma inexistência de controle de qualidade dos serviços. Se verificarmos os outros modelos de serviços, hoje largamente implementados, como o sistema de distribuição de energia elétrica, o sistema financeiro ou o sistema de telecomunicações, veremos que todos eles possuem um documento que descreve as características do fornecimento do serviço, estabelece os requisitos mínimos de qualidade que deve ser oferecido para os clientes e também define a forma de utilização que deve ser adotada pelos mesmos clientes. Logo, o mesmo conceito deve ser adotado pelos serviços na *Web*.

No contexto deste trabalho, definimos que seria necessário que os *web services* participantes da composição implementassem uma interface de controle. Na seção reservada às conclusões, afirmamos que esta interface poderia ser eliminada e mesmo assim o monitoramento e controle poderiam continuar. Esta outra forma de obter os dados de requisitos dos *web services* poderia ser então realizada por um mecanismo externo (independente) ao mesmo. Desta forma, os *web services* não necessariamente precisam possuir a interface de controle. Esta auditoria de serviços na *Web* não é um tema fácil de ser tratado e pode ser proposto como uma linha de pesquisa para a continuidade do presente trabalho. Muitos fatores dificultam o processo de obtenção das informações dos serviços na *Web*, tais como: problemas no nível de redes de computadores e do próprio acesso as informações internas dos *web service*s. Portanto, podemos considerar a definição e implementação de um modelo de gerência para o monitoramento e, conseqüentemente, a adaptação da composição em tempo de execução um grande desafio.

Referências bibliográficas

(ActiveBPEL, 2008) **ActiveBPEL L.L.C.** The Open Source BPEL Engine // The Open Source BPEL Engine. - 2008.

(Allen, 1997) **Allen, R. J.** (1997), A Formal Approach to Software Architecture, PhD thesis, School of Computer Science, Carnegie Mellon University

(Deitel, 1998) **Deitel H. M.** Java How to program [Livro]. - [s.l.] : Prentice Hall, 1998. - Second Edition : p. 1063.

(EndPoints, 2007) **EndPoints Active.** BPEL fundamentals part 1 - Partner Interaction [Relatório] / Active Endpoints. - 2007.

(APACHE, 2008) **Installing Apache SOAP** [Conferência]. - 2008.

(CORREIA, 2005) **Correia, J. M.** Aumento da resiliência em Web Services com uma Infra-Estrutura Peer-to-Peer [Periódico]. - 2005. - p. 10.

(LINS, 2007) **Lins, F. A. A.** COMPOSIÇÃO ADAPTATIVA DE WEB SERVICES [Relatório de Dissertação] / UNIVERSIDADE FEDERAL DE PERNAMBUCO. - 2007.

(Mahmoud, 2005) **Mahmoud, Q. H.** Service-Oriented Architecture (SOA) and Web Services: The Road to Enterprise Application Integration (EAI) [Periódico] // SUN. - 2005. - p. 7.

(Mani, 2002) **Mani, A.** Understanding quality of service for Web services [Periódico] // IBM. - 2002.

(MYERSON, 2002) **Myerson, J.** Use SLAs in a Web Services context, Part 1: Guarantee your Web service with SLA [Periódico] // IBM. - 2002. - p. 5.

(OASIS, 2007) **OASIS.** Web Services Business Process Execution Language Version 2.0 [Relatório] / OASIS. - 2007.

(PAPAZOGLOU, 2006) **Papazoglou, M. P.** SERVICE-ORIENTED COMPUTING RESEARCH ROADMAP [Periódico]. - 2006. - p. 29.

(RAMALHO, 2002) **Ramalho, J. A.** XML Teoria e Prática [Livro]. - [s.l.] : Berkeley, 2002. - p. 146.

(Sun, 2008) **Sun Microsystems**. Java RMI over IIOP. http://java.sun.com/products/rmi-iiop. Acessado em setembro de 2008.

(VERMA, 2005) **Verma, D. C.** Service Level Agreements on IP Networks. [Periódico]. - 2005. - p. 13.

(W3C, 2001) **W3C.** Web Services Description Language (WSDL) 1.1 - 2001.

(W3C, 2008) **W3C**. Extensible Markup Language (XML). http://www.w3c.org/XML. Acessado em setembro de 2008.

(WESTPHALL, et al., 2000) **Westphall, C. B., Ribas, J. C. da C.** Acordo de Nível de Serviço Service Level Agreement - SLA [Periódico]. - 2000. - p. 148.

Apêndices

Este apêndice disponibiliza os códigos fontes dos arquivos utilizados neste trabalho para facilitar o entendimento da abordagem e permitir referências futuras.

Códigos fontes dos *web services*

Nesta seção estão disponíveis o código fonte de todos os *web services* utilizados neste trabalho, inclusive o código fonte dos *web services* reservas.

Split.jws

```
public class Split {

	public Split() {

	}

	public String split(String word) {
		String ret = null;
		Integer middle = word.length() / 2;
		String part1 = word.substring(0, middle);
		String part2 = word.substring(middle);
		ret = part1 + ";" + part2;
		return ret;
	}

	public Integer getSLA(String param) {
	  Integer ret = 0;
	  if (param.equals("DEFAULT")) {
			ret =  1+(int)(500*Math.random());
	  }
	  return ret;
	}

}
```

UpperCase.jws

```
public class UpperCase {

	public UpperCase() {

	}

	public String uppercase(String word) {
		return word.toUpperCase();
	}
```

```
	public Integer getSLA(String param) {
	  Integer ret = 0;
	  if (param.equals("DEFAULT")) {
		ret =  1+(int)(500*Math.random());
	  }
	  return ret;
	}

}
```

Merge.jws

```
public class Merge {

	public Merge() {

	}

	public String merge(String word) {
		return ((word.split(";")[0]) + (word.split(";")[1]));
	}

	public Integer getSLA(String param) {
	  Integer ret = 0;
	  if (param.equals("DEFAULT")) {
		ret =  1+(int)(500*Math.random());
	  }
	  return ret;
	}

}
```

Split_BACKUP_01.jws

```
public class Split_BACKUP_01 {

	public Split_BACKUP_01() {

	}

	public String split(String word) {
		String ret = null;
		Integer middle = word.length() / 2;
		String part1 = word.substring(0, middle);
		String part2 = word.substring(middle);
		ret = part1 + ";" + part2;
		return ret;
	}

	public Integer getSLA(String param) {
	  Integer ret = 0;
	  if (param.equals("DEFAULT")) {
		ret =  1+(int)(500*Math.random());
	  }
	  return ret;
	}
```

```
}
```

UpperCase_BACKUP_01.jws

```
public class UpperCase_BACKUP_01 {

      public UpperCase_BACKUP_01() {

      }

      public String uppercase(String word) {
            return word.toUpperCase();
      }

      public Integer getSLA(String param) {
        Integer ret = 0;
        if (param.equals("DEFAULT")) {
              ret =  1+(int)(500*Math.random());
        }
        return ret;
      }

}
```

Merge_BACKUP_01.jws

```
public class Merge_BACKUP_01 {

      public Merge_BACKUP_01() {

      }

      public String merge(String word) {
            return ((word.split(";")[0]) + (word.split(";")[1]));
      }

      public Integer getSLA(String param) {
        Integer ret = 0;
        if (param.equals("DEFAULT")) {
              ret =  1+(int)(500*Math.random());
        }
        return ret;
      }

}
```

Códigos fontes das interfaces WSDL dos *web services*

Nesta seção estão disponíveis as interfaces WSDL de todos os *web services* utilizados neste trabalho, inclusive as interfaces WSDL dos *web services* reservas.

Split.wsdl

```
<?xml version="1.0" encoding="UTF-8" ?>
- <wsdl:definitions
    targetNamespace="http://localhost:8080/axis/Split.jws"
    xmlns:apachesoap="http://xml.apache.org/xml-soap"
    xmlns:impl="http://localhost:8080/axis/Split.jws"
    xmlns:intf="http://localhost:8080/axis/Split.jws"
    xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
    xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
    xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema">
      - <!--
      WSDL created by Apache Axis version: 1.4
      Built on Nov 29, 2005 (07:12:28 GMT+00:00)
    -->
- <wsdl:message name="splitResponse">
  <wsdl:part name="splitReturn" type="xsd:string" />
    </wsdl:message>
- <wsdl:message name="getSLARequest">
  <wsdl:part name="param" type="xsd:string" />
    </wsdl:message>
- <wsdl:message name="splitRequest">
  <wsdl:part name="word" type="xsd:string" />
    </wsdl:message>
- <wsdl:message name="getSLAResponse">
  <wsdl:part name="getSLAReturn" type="xsd:int" />
    </wsdl:message>
- <wsdl:portType name="Split">
- <wsdl:operation name="split" parameterOrder="word">
  <wsdl:input message="impl:splitRequest" name="splitRequest" />
  <wsdl:output message="impl:splitResponse" name="splitResponse" />
    </wsdl:operation>
- <wsdl:operation name="getSLA" parameterOrder="param">
  <wsdl:input message="impl:getSLARequest" name="getSLARequest" />
  <wsdl:output message="impl:getSLAResponse" name="getSLAResponse" />
    </wsdl:operation>
    </wsdl:portType>
- <wsdl:binding name="SplitSoapBinding" type="impl:Split">
  <wsdlsoap:binding style="rpc"
    transport="http://schemas.xmlsoap.org/soap/http" />
```

```
- <wsdl:operation name="split">
  <wsdlsoap:operation soapAction="" />
- <wsdl:input name="splitRequest">
  <wsdlsoap:body
    encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
    namespace="http://DefaultNamespace" use="encoded" />
    </wsdl:input>
- <wsdl:output name="splitResponse">
  <wsdlsoap:body
    encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
    namespace="http://localhost:8080/axis/Split.jws" use="encoded" />
    </wsdl:output>
    </wsdl:operation>
- <wsdl:operation name="getSLA">
  <wsdlsoap:operation soapAction="" />
- <wsdl:input name="getSLARequest">
  <wsdlsoap:body
    encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
    namespace="http://DefaultNamespace" use="encoded" />
    </wsdl:input>
- <wsdl:output name="getSLAResponse">
  <wsdlsoap:body
    encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
    namespace="http://localhost:8080/axis/Split.jws" use="encoded" />
    </wsdl:output>
    </wsdl:operation>
    </wsdl:binding>
- <wsdl:service name="SplitService">
- <wsdl:port binding="impl:SplitSoapBinding" name="Split">
  <wsdlsoap:address location="http://localhost:8080/axis/Split.jws" />
    </wsdl:port>
    </wsdl:service>
    </wsdl:definitions>
```

UpperCase.wsdl

```
  <?xml version="1.0" encoding="UTF-8" ?>
- <wsdl:definitions
    targetNamespace="http://localhost:8080/axis/UpperCase.jws"
    xmlns:apachesoap="http://xml.apache.org/xml-soap"
```

```
    xmlns:impl="http://localhost:8080/axis/UpperCase.jws"
    xmlns:intf="http://localhost:8080/axis/UpperCase.jws"
    xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
    xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
    xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema">
      - <!--
      WSDL created by Apache Axis version: 1.4
      Built on Nov 29, 2005 (07:12:28 GMT+00:00)
    -->
- <wsdl:message name="uppercaseRequest">
  <wsdl:part name="word" type="xsd:string" />
    </wsdl:message>
- <wsdl:message name="uppercaseResponse">
  <wsdl:part name="uppercaseReturn" type="xsd:string" />
    </wsdl:message>
- <wsdl:message name="getSLAResponse">
  <wsdl:part name="getSLAReturn" type="xsd:int" />
    </wsdl:message>
- <wsdl:message name="getSLARequest">
  <wsdl:part name="param" type="xsd:string" />
    </wsdl:message>
- <wsdl:portType name="UpperCase">
- <wsdl:operation name="getSLA" parameterOrder="param">
  <wsdl:input message="impl:getSLARequest" name="getSLARequest" />
  <wsdl:output message="impl:getSLAResponse" name="getSLAResponse" />
    </wsdl:operation>
- <wsdl:operation name="uppercase" parameterOrder="word">
  <wsdl:input message="impl:uppercaseRequest" name="uppercaseRequest" />
  <wsdl:output message="impl:uppercaseResponse" name="uppercaseResponse"
    />
    </wsdl:operation>
    </wsdl:portType>
- <wsdl:binding name="UpperCaseSoapBinding" type="impl:UpperCase">
  <wsdlsoap:binding style="rpc"
    transport="http://schemas.xmlsoap.org/soap/http" />
- <wsdl:operation name="getSLA">
  <wsdlsoap:operation soapAction="" />
- <wsdl:input name="getSLARequest">
```

```
  <wsdlsoap:body
    encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
    namespace="http://DefaultNamespace" use="encoded" />
    </wsdl:input>
- <wsdl:output name="getSLAResponse">
  <wsdlsoap:body
    encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
    namespace="http://localhost:8080/axis/UpperCase.jws" use="encoded" />
    </wsdl:output>
    </wsdl:operation>
- <wsdl:operation name="uppercase">
  <wsdlsoap:operation soapAction="" />
- <wsdl:input name="uppercaseRequest">
  <wsdlsoap:body
    encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
    namespace="http://DefaultNamespace" use="encoded" />
    </wsdl:input>
- <wsdl:output name="uppercaseResponse">
  <wsdlsoap:body
    encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
    namespace="http://localhost:8080/axis/UpperCase.jws" use="encoded" />
    </wsdl:output>
    </wsdl:operation>
    </wsdl:binding>
- <wsdl:service name="UpperCaseService">
- <wsdl:port binding="impl:UpperCaseSoapBinding" name="UpperCase">
  <wsdlsoap:address location="http://localhost:8080/axis/UpperCase.jws"
    />
    </wsdl:port>
    </wsdl:service>
    </wsdl:definitions>
```

Merge.wsdl

```
  <?xml version="1.0" encoding="UTF-8" ?>
- <wsdl:definitions
    targetNamespace="http://localhost:8080/axis/Merge.jws"
    xmlns:apachesoap="http://xml.apache.org/xml-soap"
    xmlns:impl="http://localhost:8080/axis/Merge.jws"
    xmlns:intf="http://localhost:8080/axis/Merge.jws"
```

```
    xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
    xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
    xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema">
      - <!--
      WSDL created by Apache Axis version: 1.4
      Built on Nov 29, 2005 (07:12:28 GMT+00:00)
    -->
- <wsdl:message name="getSLARequest">
  <wsdl:part name="param" type="xsd:string" />
    </wsdl:message>
- <wsdl:message name="mergeRequest">
  <wsdl:part name="word" type="xsd:string" />
    </wsdl:message>
- <wsdl:message name="getSLAResponse">
  <wsdl:part name="getSLAReturn" type="xsd:int" />
    </wsdl:message>
- <wsdl:message name="mergeResponse">
  <wsdl:part name="mergeReturn" type="xsd:string" />
    </wsdl:message>
- <wsdl:portType name="Merge">
- <wsdl:operation name="merge" parameterOrder="word">
  <wsdl:input message="impl:mergeRequest" name="mergeRequest" />
  <wsdl:output message="impl:mergeResponse" name="mergeResponse" />
    </wsdl:operation>
- <wsdl:operation name="getSLA" parameterOrder="param">
  <wsdl:input message="impl:getSLARequest" name="getSLARequest" />
  <wsdl:output message="impl:getSLAResponse" name="getSLAResponse" />
    </wsdl:operation>
    </wsdl:portType>
- <wsdl:binding name="MergeSoapBinding" type="impl:Merge">
  <wsdlsoap:binding style="rpc"
    transport="http://schemas.xmlsoap.org/soap/http" />
- <wsdl:operation name="merge">
  <wsdlsoap:operation soapAction="" />
- <wsdl:input name="mergeRequest">
  <wsdlsoap:body
    encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
    namespace="http://DefaultNamespace" use="encoded" />
    </wsdl:input>
- <wsdl:output name="mergeResponse">
```

```
  <wsdlsoap:body
    encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
    namespace="http://localhost:8080/axis/Merge.jws" use="encoded" />
    </wsdl:output>
    </wsdl:operation>
- <wsdl:operation name="getSLA">
  <wsdlsoap:operation soapAction="" />
- <wsdl:input name="getSLARequest">
  <wsdlsoap:body
    encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
    namespace="http://DefaultNamespace" use="encoded" />
    </wsdl:input>
- <wsdl:output name="getSLAResponse">
  <wsdlsoap:body
    encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
    namespace="http://localhost:8080/axis/Merge.jws" use="encoded" />
    </wsdl:output>
    </wsdl:operation>
    </wsdl:binding>
- <wsdl:service name="MergeService">
- <wsdl:port binding="impl:MergeSoapBinding" name="Merge">
  <wsdlsoap:address location="http://localhost:8080/axis/Merge.jws" />
    </wsdl:port>
    </wsdl:service>
    </wsdl:definitions>
```

Split_BACKUP_01.wsdl

```
  <?xml version="1.0" encoding="UTF-8" ?>
- <wsdl:definitions
    targetNamespace="http://localhost:8080/axis/Split_BACKUP_01.jws"
    xmlns:apachesoap="http://xml.apache.org/xml-soap"
    xmlns:impl="http://localhost:8080/axis/Split_BACKUP_01.jws"
    xmlns:intf="http://localhost:8080/axis/Split_BACKUP_01.jws"
    xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
    xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
    xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema">
      - <!--
      WSDL created by Apache Axis version: 1.4
```

```
      Built on Nov 29, 2005 (07:12:28 GMT+00:00)
    -->
- <wsdl:message name="getSLARequest">
  <wsdl:part name="param" type="xsd:string" />
    </wsdl:message>
- <wsdl:message name="splitRequest">
  <wsdl:part name="word" type="xsd:string" />
    </wsdl:message>
- <wsdl:message name="getSLAResponse">
  <wsdl:part name="getSLAReturn" type="xsd:int" />
    </wsdl:message>
- <wsdl:message name="splitResponse">
  <wsdl:part name="splitReturn" type="xsd:string" />
    </wsdl:message>
- <wsdl:portType name="Split_BACKUP_01">
- <wsdl:operation name="split" parameterOrder="word">
  <wsdl:input message="impl:splitRequest" name="splitRequest" />
  <wsdl:output message="impl:splitResponse" name="splitResponse" />
    </wsdl:operation>
- <wsdl:operation name="getSLA" parameterOrder="param">
  <wsdl:input message="impl:getSLARequest" name="getSLARequest" />
  <wsdl:output message="impl:getSLAResponse" name="getSLAResponse" />
    </wsdl:operation>
    </wsdl:portType>
- <wsdl:binding name="Split_BACKUP_01SoapBinding"
    type="impl:Split_BACKUP_01">
  <wsdlsoap:binding style="rpc"
    transport="http://schemas.xmlsoap.org/soap/http" />
- <wsdl:operation name="split">
  <wsdlsoap:operation soapAction="" />
- <wsdl:input name="splitRequest">
  <wsdlsoap:body
    encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
    namespace="http://DefaultNamespace" use="encoded" />
    </wsdl:input>
- <wsdl:output name="splitResponse">
  <wsdlsoap:body
    encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
    namespace="http://localhost:8080/axis/Split_BACKUP_01.jws"
    use="encoded" />
    </wsdl:output>
```

```
    </wsdl:operation>
- <wsdl:operation name="getSLA">
  <wsdlsoap:operation soapAction="" />
- <wsdl:input name="getSLARequest">
  <wsdlsoap:body
    encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
    namespace="http://DefaultNamespace" use="encoded" />
    </wsdl:input>
- <wsdl:output name="getSLAResponse">
  <wsdlsoap:body
    encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
    namespace="http://localhost:8080/axis/Split_BACKUP_01.jws"
    use="encoded" />
    </wsdl:output>
    </wsdl:operation>
    </wsdl:binding>
- <wsdl:service name="Split_BACKUP_01Service">
- <wsdl:port binding="impl:Split_BACKUP_01SoapBinding"
    name="Split_BACKUP_01">
  <wsdlsoap:address
    location="http://localhost:8080/axis/Split_BACKUP_01.jws" />
    </wsdl:port>
    </wsdl:service>
    </wsdl:definitions>
```

UpperCase_BACKUP_01.wsdl

```
   <?xml version="1.0" encoding="UTF-8" ?>
- <wsdl:definitions
    targetNamespace="http://localhost:8080/axis/UpperCase_BACKUP_01.jws"
    xmlns:apachesoap="http://xml.apache.org/xml-soap"
    xmlns:impl="http://localhost:8080/axis/UpperCase_BACKUP_01.jws"
    xmlns:intf="http://localhost:8080/axis/UpperCase_BACKUP_01.jws"
    xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
    xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
    xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema">
      - <!--
      WSDL created by Apache Axis version: 1.4
      Built on Nov 29, 2005 (07:12:28 GMT+00:00)
```

```
    -->
- <wsdl:message name="uppercaseResponse">
  <wsdl:part name="uppercaseReturn" type="xsd:string" />
    </wsdl:message>
- <wsdl:message name="getSLAResponse">
  <wsdl:part name="getSLAReturn" type="xsd:int" />
    </wsdl:message>
- <wsdl:message name="getSLARequest">
  <wsdl:part name="param" type="xsd:string" />
    </wsdl:message>
- <wsdl:message name="uppercaseRequest">
  <wsdl:part name="word" type="xsd:string" />
    </wsdl:message>
- <wsdl:portType name="UpperCase_BACKUP_01">
- <wsdl:operation name="getSLA" parameterOrder="param">
  <wsdl:input message="impl:getSLARequest" name="getSLARequest" />
  <wsdl:output message="impl:getSLAResponse" name="getSLAResponse" />
    </wsdl:operation>
- <wsdl:operation name="uppercase" parameterOrder="word">
  <wsdl:input message="impl:uppercaseRequest" name="uppercaseRequest" />
  <wsdl:output message="impl:uppercaseResponse" name="uppercaseResponse"
    />
    </wsdl:operation>
    </wsdl:portType>
- <wsdl:binding name="UpperCase_BACKUP_01SoapBinding"
    type="impl:UpperCase_BACKUP_01">
  <wsdlsoap:binding style="rpc"
    transport="http://schemas.xmlsoap.org/soap/http" />
- <wsdl:operation name="getSLA">
  <wsdlsoap:operation soapAction="" />
- <wsdl:input name="getSLARequest">
  <wsdlsoap:body
    encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
    namespace="http://DefaultNamespace" use="encoded" />
    </wsdl:input>
- <wsdl:output name="getSLAResponse">
  <wsdlsoap:body
    encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
    namespace="http://localhost:8080/axis/UpperCase_BACKUP_01.jws"
    use="encoded" />
    </wsdl:output>
```

```
    </wsdl:operation>
- <wsdl:operation name="uppercase">
  <wsdlsoap:operation soapAction="" />
- <wsdl:input name="uppercaseRequest">
  <wsdlsoap:body
    encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
    namespace="http://DefaultNamespace" use="encoded" />
    </wsdl:input>
- <wsdl:output name="uppercaseResponse">
  <wsdlsoap:body
    encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
    namespace="http://localhost:8080/axis/UpperCase_BACKUP_01.jws"
    use="encoded" />
    </wsdl:output>
    </wsdl:operation>
    </wsdl:binding>
- <wsdl:service name="UpperCase_BACKUP_01Service">
- <wsdl:port binding="impl:UpperCase_BACKUP_01SoapBinding"
    name="UpperCase_BACKUP_01">
  <wsdlsoap:address
    location="http://localhost:8080/axis/UpperCase_BACKUP_01.jws" />
    </wsdl:port>
    </wsdl:service>
    </wsdl:definitions>
```

Merge_BACKUP_01.wsdl

```
   <?xml version="1.0" encoding="UTF-8" ?>
- <wsdl:definitions
    targetNamespace="http://localhost:8080/axis/Merge_BACKUP_01.jws"
    xmlns:apachesoap="http://xml.apache.org/xml-soap"
    xmlns:impl="http://localhost:8080/axis/Merge_BACKUP_01.jws"
    xmlns:intf="http://localhost:8080/axis/Merge_BACKUP_01.jws"
    xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
    xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
    xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema">
      - <!--
      WSDL created by Apache Axis version: 1.4
      Built on Nov 29, 2005 (07:12:28 GMT+00:00)
```

```
    -->
- <wsdl:message name="mergeResponse">
  <wsdl:part name="mergeReturn" type="xsd:string" />
    </wsdl:message>
- <wsdl:message name="getSLAResponse">
  <wsdl:part name="getSLAReturn" type="xsd:int" />
    </wsdl:message>
- <wsdl:message name="mergeRequest">
  <wsdl:part name="word" type="xsd:string" />
    </wsdl:message>
- <wsdl:message name="getSLARequest">
  <wsdl:part name="param" type="xsd:string" />
    </wsdl:message>
- <wsdl:portType name="Merge_BACKUP_01">
- <wsdl:operation name="merge" parameterOrder="word">
  <wsdl:input message="impl:mergeRequest" name="mergeRequest" />
  <wsdl:output message="impl:mergeResponse" name="mergeResponse" />
    </wsdl:operation>
- <wsdl:operation name="getSLA" parameterOrder="param">
  <wsdl:input message="impl:getSLARequest" name="getSLARequest" />
  <wsdl:output message="impl:getSLAResponse" name="getSLAResponse" />
    </wsdl:operation>
    </wsdl:portType>
- <wsdl:binding name="Merge_BACKUP_01SoapBinding"
    type="impl:Merge_BACKUP_01">
  <wsdlsoap:binding style="rpc"
    transport="http://schemas.xmlsoap.org/soap/http" />
- <wsdl:operation name="merge">
  <wsdlsoap:operation soapAction="" />
- <wsdl:input name="mergeRequest">
  <wsdlsoap:body
    encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
    namespace="http://DefaultNamespace" use="encoded" />
    </wsdl:input>
- <wsdl:output name="mergeResponse">
  <wsdlsoap:body
    encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
    namespace="http://localhost:8080/axis/Merge_BACKUP_01.jws"
    use="encoded" />
    </wsdl:output>
    </wsdl:operation>
```

```
- <wsdl:operation name="getSLA">
  <wsdlsoap:operation soapAction="" />
- <wsdl:input name="getSLARequest">
  <wsdlsoap:body
    encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
    namespace="http://DefaultNamespace" use="encoded" />
    </wsdl:input>
- <wsdl:output name="getSLAResponse">
  <wsdlsoap:body
    encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
    namespace="http://localhost:8080/axis/Merge_BACKUP_01.jws"
    use="encoded" />
    </wsdl:output>
    </wsdl:operation>
    </wsdl:binding>
- <wsdl:service name="Merge_BACKUP_01Service">
- <wsdl:port binding="impl:Merge_BACKUP_01SoapBinding"
    name="Merge_BACKUP_01">
  <wsdlsoap:address
    location="http://localhost:8080/axis/Merge_BACKUP_01.jws" />
    </wsdl:port>
    </wsdl:service>
    </wsdl:definitions>
```

Códigos fontes da Composição *UpperCase* Distribuído

UpperCase.bpel

```
<?xml version="1.0" encoding="UTF-8"?>
<!--
BPEL Process Definition
Edited using ActiveBPEL(tm) Designer Version 2.0.0
(http://www.active-endpoints.com)
-->
<process name="UpperCase" suppressJoinFailure="yes"
targetNamespace="http://UpperCase"
xmlns="http://schemas.xmlsoap.org/ws/2003/03/business-process/"
xmlns:bpws="http://schemas.xmlsoap.org/ws/2003/03/business-process/"
xmlns:ns1="http://localhost:8080/web_services/composicao"
xmlns:ns2="http://localhost:8080/axis/UpperCase.jws"
xmlns:ns3="http://localhost:8080/axis/Merge.jws"
```

```
xmlns:ns4="http://172.16.15.129:8080/axis/Split.jws"
xmlns:ns5="http://localhost:8080/axis/Split.jws"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
   <partnerLinks>
      <partnerLink myRole="requestRole" name="requestPartnerLinkType"
partnerLinkType="ns1:requestPartnerLinkType"/>
      <partnerLink name="splitPartnerLinkType"
partnerLinkType="ns1:splitPartnerLinkType" partnerRole="splitRole"/>
      <partnerLink name="uppercasePartnerLinkType"
partnerLinkType="ns1:uppercasePartnerLinkType"
partnerRole="uppercaseRole"/>
      <partnerLink name="mergePartnerLinkType"
partnerLinkType="ns1:mergePartnerLinkType" partnerRole="mergeRole"/>
   </partnerLinks>
   <variables>
      <variable messageType="ns1:parametroMessage"
name="parametroMessage"/>
      <variable messageType="ns1:respostaMessage"
name="respostaMessage"/>
      <variable messageType="ns5:splitRequest" name="splitRequest"/>
      <variable messageType="ns5:splitResponse"
name="splitResponse"/>
      <variable messageType="ns2:uppercaseRequest"
name="uppercaseRequest"/>
      <variable messageType="ns2:uppercaseResponse"
name="uppercaseResponse"/>
      <variable messageType="ns3:mergeRequest" name="mergeRequest"/>
      <variable messageType="ns3:mergeResponse"
name="mergeResponse"/>
   </variables>
   <flow>
      <links>
         <link name="L1"/>
         <link name="L2"/>
      </links>
      <receive createInstance="yes" operation="request"
partnerLink="requestPartnerLinkType" portType="ns1:requestPT"
variable="parametroMessage">
         <source linkName="L1"/>
      </receive>
```

```
      <reply operation="request" partnerLink="requestPartnerLinkType"
portType="ns1:requestPT" variable="respostaMessage">
         <target linkName="L2"/>
      </reply>
      <sequence>
         <target linkName="L1"/>
         <assign>
            <copy>
               <from part="word" variable="parametroMessage"/>
               <to part="word" variable="splitRequest"/>
            </copy>
         </assign>
         <invoke inputVariable="splitRequest" operation="split"
outputVariable="splitResponse" partnerLink="splitPartnerLinkType"
portType="ns5:Split"/>
         <assign>
            <copy>
               <from
expression="bpws:getVariableData('splitResponse', 'splitReturn')"/>
               <to part="word" variable="uppercaseRequest"/>
            </copy>
         </assign>
         <invoke inputVariable="uppercaseRequest"
operation="uppercase" outputVariable="uppercaseResponse"
partnerLink="uppercasePartnerLinkType" portType="ns2:UpperCase"/>
         <assign>
            <copy>
               <from
expression="bpws:getVariableData('uppercaseResponse',
'uppercaseReturn')"/>
               <to part="word" variable="mergeRequest"/>
            </copy>
         </assign>
         <invoke inputVariable="mergeRequest" operation="merge"
outputVariable="mergeResponse" partnerLink="mergePartnerLinkType"
portType="ns3:Merge"/>
         <assign>
            <source linkName="L2"/>
            <copy>
               <from
expression="bpws:getVariableData('mergeResponse', 'mergeReturn')"/>
```

```
            <to part="resposta" variable="respostaMessage"/>
          </copy>
        </assign>
      </sequence>
   </flow>
</process>
```

composicao.wsdl

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions
targetNamespace="http://localhost:8080/web_services/composicao"
xmlns:bpws="http://schemas.xmlsoap.org/ws/2003/03/business-process/"
xmlns:uppercase="http://localhost:8080/axis/UpperCase.jws"
xmlns:plnk="http://schemas.xmlsoap.org/ws/2003/05/partner-link/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:split="http://localhost:8080/axis/Split.jws"
xmlns:comp="http://localhost:8080/web_services/composicao"
xmlns:merge="http://localhost:8080/axis/Merge.jws"
xmlns="http://schemas.xmlsoap.org/wsdl/">
  <import namespace="http://localhost:8080/axis/UpperCase.jws"
location="http://localhost:8080/axis/UpperCase.jws?wsdl"/>
  <import namespace="http://localhost:8080/axis/Merge.jws"
location="http://localhost:8080/axis/Merge.jws?wsdl"/>
  <import namespace="http://localhost:8080/axis/Split.jws"
location="http://localhost:8080/axis/Split.jws?wsdl"/>
  <message name="parametroMessage">
    <part name="word" type="xsd:string"/>
  </message>
  <message name="errorMessage">
    <part name="errorCode" type="xsd:integer"/>
  </message>
  <message name="respostaMessage">
    <part name="resposta" type="xsd:string"/>
  </message>
  <portType name="requestPT">
    <operation name="request">
      <input message="comp:parametroMessage"/>
      <output message="comp:respostaMessage"/>
```

```
      <fault name="unableToHandleRequest"
message="comp:errorMessage"/>
    </operation>
  </portType>
  <service name="uppercaseService">
  </service>
<plnk:partnerLinkType name="requestPartnerLinkType">
   <plnk:role name="requestRole">
      <plnk:portType name="comp:requestPT"/>
   </plnk:role>
</plnk:partnerLinkType>
<plnk:partnerLinkType name="splitPartnerLinkType">
   <plnk:role name="splitRole">
      <plnk:portType name="split:Split"/>
   </plnk:role>
</plnk:partnerLinkType>
<plnk:partnerLinkType name="uppercasePartnerLinkType">
   <plnk:role name="uppercaseRole">
      <plnk:portType name="uppercase:UpperCase"/>
   </plnk:role>
</plnk:partnerLinkType>
<plnk:partnerLinkType name="mergePartnerLinkType">
   <plnk:role name="mergeRole">
      <plnk:portType name="merge:Merge"/>
   </plnk:role>
</plnk:partnerLinkType>
</definitions>
```

UpperCase.pdd

```
<?xml version="1.0" encoding="UTF-8"?>
<process location="bpel/UpperCase/UpperCase.bpel"
name="bpelns:UpperCase" xmlns="http://schemas.active-
endpoints.com/pdd/2005/09/pd.xsd" xmlns:bpelns="http://UpperCase"
xmlns:wsa="http://schemas.xmlsoap.org/ws/2003/03/addressing">
   <partnerLinks>
      <partnerLink name="mergePartnerLinkType">
         <partnerRole endpointReference="static">
            <wsa:EndpointReference
xmlns:s="http://localhost:8080/axis/Merge.jws"
```

```
xmlns:wsa="http://schemas.xmlsoap.org/ws/2003/03/addressing">
<wsa:Address>http://localhost:8080/axis/Merge.jws</wsa:Address>
<wsa:ServiceName PortName="Merge">s:MergeService</wsa:ServiceName>
</wsa:EndpointReference>
         </partnerRole>
      </partnerLink>
      <partnerLink name="requestPartnerLinkType">
         <myRole allowedRoles="" binding="RPC"
service="uppercaseService"/>
      </partnerLink>
      <partnerLink name="splitPartnerLinkType">
         <partnerRole endpointReference="static">
            <wsa:EndpointReference
xmlns:s="http://localhost:8080/axis/Split.jws"
xmlns:wsa="http://schemas.xmlsoap.org/ws/2003/03/addressing">
<wsa:Address>http://localhost:8080/axis/Split.jws</wsa:Address>
<wsa:ServiceName PortName="Split">s:SplitService</wsa:ServiceName>
</wsa:EndpointReference>
         </partnerRole>
      </partnerLink>
      <partnerLink name="uppercasePartnerLinkType">
         <partnerRole endpointReference="static">
            <wsa:EndpointReference
xmlns:s="http://localhost:8080/axis/UpperCase.jws"
xmlns:wsa="http://schemas.xmlsoap.org/ws/2003/03/addressing">
<wsa:Address>http://localhost:8080/axis/UpperCase.jws</wsa:Address>
<wsa:ServiceName
PortName="UpperCase">s:UpperCaseService</wsa:ServiceName>
</wsa:EndpointReference>
         </partnerRole>
      </partnerLink>
   </partnerLinks>
   <wsdlReferences>
      <wsdl location="project:/UpperCase/composicao.wsdl"
namespace="http://localhost:8080/web_services/composicao"/>
      <wsdl location="http://localhost:8080/axis/UpperCase.jws?wsdl"
namespace="http://localhost:8080/axis/UpperCase.jws"/>
      <wsdl location="http://localhost:8080/axis/Merge.jws?wsdl"
namespace="http://localhost:8080/axis/Merge.jws"/>
      <wsdl location="http://localhost:8080/axis/Split.jws?wsdl"
namespace="http://localhost:8080/axis/Split.jws"/>
```

```
        </wsdlReferences>
    </process>
```

uppercaseService.wsdl

```
<?xml version="1.0" encoding="UTF-8" ?>
- <definitions
    targetNamespace="http://localhost:8080/web_services/composicao"
    xmlns="http://schemas.xmlsoap.org/wsdl/"
    xmlns:bpws="http://schemas.xmlsoap.org/ws/2003/03/business-process/"
    xmlns:comp="http://localhost:8080/web_services/composicao"
    xmlns:merge="http://localhost:8080/axis/Merge.jws"
    xmlns:plnk="http://schemas.xmlsoap.org/ws/2003/05/partner-link/"
    xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
    xmlns:split="http://localhost:8080/axis/Split.jws"
    xmlns:uppercase="http://localhost:8080/axis/UpperCase.jws"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <import location="http://localhost:8080/axis/UpperCase.jws?wsdl"
    namespace="http://localhost:8080/axis/UpperCase.jws" />
  <import location="http://localhost:8080/axis/Split.jws?wsdl"
    namespace="http://localhost:8080/axis/Split.jws" />
  <import location="http://localhost:8080/axis/Merge.jws?wsdl"
    namespace="http://localhost:8080/axis/Merge.jws" />
- <message name="parametroMessage">
  <part name="word" type="xsd:string" />
    </message>
- <message name="errorMessage">
  <part name="errorCode" type="xsd:integer" />
    </message>
- <message name="respostaMessage">
  <part name="resposta" type="xsd:string" />
    </message>
- <portType name="requestPT">
- <operation name="request">
  <input message="comp:parametroMessage" />
  <output message="comp:respostaMessage" />
  <fault message="comp:errorMessage" name="unableToHandleRequest" />
    </operation>
    </portType>
- <binding name="uppercaseServiceBinding" type="comp:requestPT">
```

```
  <soap:binding style="rpc"
    transport="http://schemas.xmlsoap.org/soap/http"
    xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/" />
- <operation name="request">
  <soap:operation soapAction="" style="rpc"
    xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/" />
- <input>
  <soap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
    use="encoded" xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/" />
    </input>
- <output>
  <soap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
    use="encoded" xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/" />
    </output>
- <fault name="unableToHandleRequest">
  <soap:fault encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
    name="unableToHandleRequest" use="encoded"
    xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/" />
    </fault>
    </operation>
    </binding>
- <service name="uppercaseService">
- <port binding="comp:uppercaseServiceBinding"
    name="uppercaseServicePort">
  <soap:address location="http://localhost:8080/active-
    bpel/services/uppercaseService"
    xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/" />
    </port>
    </service>
    </definitions>
```