UNIVERSIDADE METODISTA DE PIRACICABA

FACULDADE DE CIÊNCIAS EXATAS E DA NATUREZA

MESTRADO EM CIÊNCIA DA COMPUTAÇÃO

PLATAFORMA DE EXECUÇÃO DE SERVIÇOS WEB COREOGRAFADOS POR INTERPRETAÇÃO DE DOCUMENTOS WS-CDL

KLEBER EMANUELLE BAPTISTA

ORIENTADOR: PROF. DR. MÁRCIO MERINO FERNANDES

PIRACICABA, SP
2006

UNIVERSIDADE METODISTA DE PIRACICABA

FACULDADE DE CIÊNCIAS EXATAS E DA NATUREZA

MESTRADO EM CIÊNCIA DA COMPUTAÇÃO

PLATAFORMA DE EXECUÇÃO DE SERVIÇOS WEB COREOGRAFADOS POR INTERPRETAÇÃO DE DOCUMENTOS WS-CDL

KLEBER EMANUELLE BAPTISTA

ORIENTADOR: PROF. DR. MÁRCIO MERINO FERNANDES

Dissertação apresentada ao Mestrado em Ciência da Computação, da Faculdade de Ciências Exatas e da Natureza, da Universidade Metodista de Piracicaba – UNIMEP, como requisito para obtenção do Título de Mestre em Ciência da Computação.

PIRACICABA, SP
2006

Baptista, Kleber Emanuelle
Plataforma de execução de serviços Web coreografados por interpretação de documentos WS-CDL. Piracicaba, 2006.
137p.

Orientador: Prof. Dr. Márcio Merino Fernandes
Dissertação (mestrado) - Programa de Pós-Graduação em Ciência da Computação - Universidade Metodista de Piracicaba

1- Sistemas distribuídos. 2- Serviços Web. 3- Orquestração e coreografia de serviços. 4- WS-CDL.

Aos meus pais Emanuel e Fátima, que com amor deram-me consciência.

Ao meu filho Kelvin, que com espontaneidade, deu-me alegria.

À minha esposa Silvana, que com consciência deu-me amor.

AGRADECIMENTOS

Ao professor Manoel de Jesus Mendes que fez despertar toda minha esperança na conquista deste Mestrado.

A todos os professores do programa de Mestrado em Ciência da Computação da Universidade Metodista de Piracicaba que com dedicação e amizade, proporcionaram-me colher frutos jamais imaginados.

Em especial, ao professor Márcio Merino Fernandes que me acolheu como orientando e com serenidade, discernimento e conhecimento possibilitou a finalização deste trabalho.

RESUMO

Com o crescimento da Internet popularizou-se a facilidade e a agilidade da realização da troca de dados de forma geral, e conseqüentemente despertou interesse de entidades comerciais de dispor desses atributos em seus sistemas para alcançar resultados mais eficientes. Para desenvolver soluções capazes de fornecer essas qualidades novas tecnologias foram desenvolvidas, entre elas os Serviços Web. Junto com essa tecnologia, outros conceitos foram apresentados, em especial a Coreografia de serviços, que prevê a coordenação das atividades de troca de dados ou mensagens entre serviços distintos que compartilham objetivos comuns.

Esta dissertação apresenta inicialmente os conceitos de serviços web e as tecnologias necessárias para desenvolvê-los. Posteriormente aborda a utilização desses serviços em um ambiente determinado como colaborativo, explorando a definição da coreografia de serviços empregando especificamente a linguagem WS-CDL. Como objetivo principal há a proposta de uma plataforma de execução de serviços web coreografados para incitar a aplicação dos conceitos apresentados. Um protótipo da plataforma é implementado e mediante um estudo de caso realiza-se testes da plataforma e a descrição de notas referentes aos resultados obtidos.

PALAVRAS-CHAVE: Sistemas Distribuídos, Serviços Web, Orquestração e Coreografia de Serviços, WS-CDL.

ABSTRACT

Given the increasing use of the Internet, data exchange has generally become more popular and more easily conducted, consequently drawing the interest of commercial entities in the availability of such capability.. In this context, new technologies have been developed with the purpose of supplying quality solutions, including Web Services. In this work, Web Services and related concepts are presented, in particular Services Choreography, which is concerned with the coordination of messages exchange between distinct services that share common goals.

This dissertation initially presents Web Services concepts and the technology required to develop them. Next, it approaches the utilization of such services in a cooperative environment employing the WS-CDL language to implement choreography of services. The main objective of this work is to propose a platform for the execution of choreographed Web Services, aiming to stimulate the application of these concepts. A prototype of the platform is implemented and, through a case study, tests are performed and figures related to the obtained results are also presented.

KEYWORDS: Distributed Systems, Web Services, Services Orchestration and Choreography, WS-CDL.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE ABREVIATURAS E SIGLAS

BMPS	*Business Process Management System*
BPEL4WS	*Business Process Execution Language for Web Services*
BPML	*Business Process Modeling Language*
DTD	*Document Type Definition*
FTP	*File Transfer Protocol*
HTTP	*Hiper-Text Transfer Protocol*
IIS	*Internet Information Services*
J2EE	*Java 2 Enterprise Edition*
PNG	*Portable Network Graphics*
RPC	*Remote Procedure Call*
SMTP	*Simple Mail Transfer Protocol*
SOAP	*Simple Object Access Protocol*
TCP/IP	*Transmission Control Protocol/Internet Protocol*
UDDI	*Universal Description Discovery and Integration*
URI	*Uniform Resource Identifiers*
URL	*Uniform Resource Locator*
W3C	*World Wide Web Consortium's*
WS-CDL	*Web Services Choreography Description Language*
WSCI	*Web Service Choreography Interface*

WSDL	*Web Services Description Language*
WSFL	*Web Services Flow Language*
XLANG	*Web Services for business process design*
XML	*Extensible Markup Language*

1 INTRODUÇÃO

De forma geral percebe-se que as relações comerciais existentes entre organizações distintas estão sempre necessitando de tecnologias que possibilitem criar modelos sistêmicos que devam caracterizar-se por adjetivos como facilidade, agilidade, rapidez, alto desempenho e custos baixos. Provavelmente a falta de ferramentas que incorporem essas características determina modelos de trabalhos que ocasionam a elevação do custo operacional, ou até mesmo na inviabilização de operações de integração entre empresas.

A tecnologia de **Web services** (serviços web) possibilita criar uma integração entre sistemas distintos, independente da plataforma utilizada em seu desenvolvimento e também da plataforma onde os sistemas estarão sendo executados (AUSTIN, 2004). Isso viabiliza a construção de um canal de comunicação para ser utilizado na troca de informações entre empresas, tornando possível colaborar na solução de problemas relacionados com a integração de sistemas.

Existem várias tecnologias consolidadas para tratar do problema de integração, como por exemplo, a composição de RPCs (Remote Procedure Calls ou chamadas de procedimento remoto). Entretanto serviços web diferem das existentes por usarem padrões neutros de plataforma, como HTTP, SOAP e XML, permitindo assim ocultar os detalhes relativos à implementação (HENDRICKS, 2002).

As vantagens proporcionadas pelo uso de padrões neutros de plataforma têm impulsionado a utilização da tecnologia de serviços web, resultando em mudanças cada vez mais freqüentes na área de tecnologia de informação de empresas ou centros de pesquisas.

Considerando essas necessidades e relacionando com o cenário apresentado no inicio deste tópico (a integração de processos de negócios de empresas),

surgem questões que devem ser estudadas para prever e possibilitar a aplicação da tecnologia de serviços web dentro dos ambientes empresariais. Entre essas questões estão a colaboração e orquestração, que são fundamentais para que se tenha um controle completo da aplicação desta tecnologia para resolver questões relacionadas com a integração de sistemas heterogêneos.

A colaboração e a orquestração tratam especificamente da realização das atividades que representam as execuções dos processos de negócios existentes em transações colaborativas (GAO, 2005). Neste contexto, diversas especificações têm sido propostas para melhorar a estrutura e formalizar a definição e integração entre essas atividades colaborativas. Através destas especificações é possível o desenvolvimento de ferramentas para apoiar o projeto, implementação e execução de sistemas colaborativos utilizando-se a tecnologia de serviços web.

Uma plataforma de execução da coreografia de Serviços Web, como esta proposta neste trabalho, implementa as principais ações previstas nas especificações relacionadas à colaboração dos serviços disponíveis, facilitando e agilizando o ambiente operacional das aplicações, possibilitando a reserva de tempo maior ao desenvolvimento do software, possibilitando aumentar a qualidade do produto final.

1.1 OBJETIVOS E JUSTIFICATIVAS DO TRABALHO

O objetivo principal deste trabalho é propor uma plataforma de execução da coreografia de serviços web através da interpretação de documentos WS-CDL. Esta plataforma deverá possibilitar diversos fatores que contribuirão ao desenvolvimento de soluções de serviços web, destacando:

- Interpretar documentos coreográficos desenvolvidos na linguagem WS-CDL;

- Analisar e validar documentos coreográficos desenvolvidos na linguagem WS-CDL;
- Visualizar as interfaces entre os serviços web dispostos na coreografia;
- Encapsular as rotinas de chamada aos serviços web, facilitando a disponibilização dos serviços e consecutivamente reduzindo o tempo de desenvolvimento de soluções baseadas em serviços web.
- Executar os serviços web participantes de uma coreografia, gerando um ambiente de processamento de serviços dispostos frente a um objetivo comum, como exemplo uma aplicação comercial envolvendo diversas entidades, entre elas: cliente (consumidor pessoa física) e vendedor (entidade jurídica) representando diversos produtores;
- Acompanhar os resultados do processamento dos serviços web, criando um ambiente de depuração dos serviçosque facilite a análise e avaliação, por parte do desenvolvendor, dos resultados gerados na execução de serviços coreografados;
- Estar disponível em um ambiente associado ao contexto de serviços web, possibilitando acesso por parte de todos os envolvidos na colaboração. Este ambiente pode ser prioritariamente a Internet.

Face ao exposto acima, pretende-se verificar a adequação da linguagem WS-CDL como estrutura básica para funcionamento da plataforma em questão, visando executar a integração de serviços. Dessa forma, estende-se como objetivo deste trabalho testar a possibilidade de empregar-se a linguagem WS-CDL para construção de um protótipo da plataforma capaz de interpretar e executar serviços web participantes de um ambiente colaborativo, e na medida que for realizado a implementação da plataforma proposta, explicitar os

benefícios proporcionados e as diferenças das abordagens convencionais de desenvolvimento de serviços web.

A plataforma proposta será testada pela aplicação de um estudo de caso que representará a aplicação de serviços web coreografados, atendendo um objetivo comum. Mais especificamente, este estudo de caso representará um serviço web denominado Livraria (que recebe procura de determinados produtos, em especial livros) que necessita solicitar a um serviço denominado Central (um distribuidor de produtos livros) a informação referente a disponibilidade de produtos específicos. O serviço Central exerce o papel de um parceiro de negócio externo. Este estudo de caso, mediante ao relacionamento das entidades envolvidas, possibilitará a aplicação e análise da necessidade da coreografia de serviços web em um ambiente de colaboração.

1.2 JUSTIFICATIVAS

Para a aplicação comercial em massa da tecnologia de serviços web, certamente o atributo facilidade deverá ser melhorado, pois no desenvolvimento de soluções comerciais o fator tempo sempre é considerado um dos recursos de maior vitalidade.

Essa facilidade pode ser conquistada pela disponibilização de um ambiente gráfico que agregue recursos ao desenvolvimento de serviços web, como por exemplo a capacidade de executar as rotinas que chamam os serviços web participantes de uma coreografia. O fato da plataforma realizar as chamadas aos serviços evita que o desenvolvedor tenha que implementar essas mesmas rotinas. Quanto maior for a capacidade da plataforma de disponibilizar funcionalidades como a citada acima, maior será a atenção do desenvolvedor à construção das regras e operações dos softwares, diminuindo substancialmente o tempo de desenvolvimento e elevando a qualidade do produto final.

Este trabalho busca exatamente incitar esta plataforma de execução, mesmo que incialmente isto ocorra, na prática, de forma isolada e independente, mas

certamente após a implementação e validação deste recurso, haverá a possibilidade de migração para um ambiente composto de diversos outros recursos.

A principal razão de desenvolvimento da plataforma proposta neste trabalho é possibilitar a execução da coreografia de serviços web utilizando a linguagem WS-CDL, proposta pelo consórcio W3C, detentor de outros padrões já altamente difundidos na Internet, como exemplo o HTML, XML, SOAP, etc.

Com a disponibilidade da plataforma poderá diretamente ganhar tempo de desenvolvimento, qualidade no serviço desenvolvido, facilidade e agilidade na execução dos serviços web e indiretamente, análise e depuração dos serviços coreografados.

1.3 ESTRUTURA DO DOCUMENTO

Este documento está estruturado de modo a evoluir de um contexto genérico para aspectos mais específicos relacionados com a elaboração do trabalho de pesquisa proposto. Os conteúdos a serem apresentados são descritos resumidamente como se seguem:

No capítulo 2 serão expostos os conceitos básicos de serviços web incluindo fundamentos que citam a definição destes serviços, assim como a arquitetura e detalhes que apresentam todas as tecnologias bases utilizadas para possibilitar o funcionamento e utilização destes sistemas.

No capítulo 3 serão apresentadas as necessidades de realizar-se a composição de serviços web, assim como os objetivos e as funções da composição e colaboração. Também serão apresentados os padrões existentes para realizar a orquestração e coreografia dos serviços web. Serão abordados os primeiros trabalhos realizados, e descritos com maiores detalhes os três padrões atualmente mais completos (BPEL4WS, WS-CDL e BPML). No final deste capítulo será analisado um relacionamento entre estes padrões. O objetivo principal é conhecer as características dos padrões para orquestrar e

coreografar serviços web, fornecendo assim conhecimento para melhor direcionar nosso trabalho principal, que envolve exclusivamente WS-CDL.

Na seqüência, o capítulo 4 irá detalhar os recursos técnicos dos principais padrões necessários para a construção de um serviço web. A leitura dessa seção capacitará ao conhecimento básico do desenvolvimento de um serviço web.

Um detalhamento da linguagem WS-CDL será exposta no capítulo 5. Este detalhamento indicará as principais funcionalidades previstas em um documento WS-CDL. Através do conhecimento destas funcionalidades será possível compreender melhor a coreografia dos serviços web.

No capítulo 6 serão apresentados inicialmente os requisitos funcionais previstos no desenvolvimento da plataforma de execução de serviços web por intermédio da interpretação de documentos WS-CDL. Posteriormente será implementado um protótipo da plataforma proposta, destacando as tecnologias diretamente utilizadas e os principais recursos disponíveis na plataforma.

Um estudo de caso testará a plataforma proposta do capítulo seis. Todas as características envolvidas na aplicação desse estudo de caso, assim como os resultados percebidos serão apresentados neste sétimo capítulo.

Concluindo a proposta deste trabalho, no capítulo 8 serão apresentadas as considerações finais ressaltando as contribuições conquistadas, assim como a avaliação dos objetivos atendidos. Encerrando, são expostas as sugestões de continuidade do presente trabalho.

2 SERVIÇOS WEB

Neste capítulo serão expostos os conceitos básicos de serviços web incluindo fundamentos que citam a definição destes serviços, assim como a arquitetura e detalhes que apresentam todas as tecnologias bases utilizadas para possibilitar o funcionamento e utilização destes sistemas.

2.1 DEFINIÇÃO DE SERVIÇOS WEB

Serviços Web são sistemas que permitem integrar sistemas distintos, de forma transparente, independente da plataforma de desenvolvimento ou utilização, usando das tecnologias padrões da Internet (BOOTH, 2004).

Ao realizar uma busca pela Internet, diversas outras definições serão encontradas relacionadas a Serviços Web. Para complementar há outra definição que classifica serviços web em dois aspectos principais (MENDES, 2002):

a) específico: Serviços Web são definidos como aplicações que expõem serviços e possibilitam facilmente a sua utilização por outras aplicações, usando tecnologias padrões da Internet (XML, SOAP, WSDL, UDDI).

b) amplo: Um Serviço Web representa uma unidade de um negócio, aplicação ou uma funcionalidade de um sistema que pode ser acessado pela Internet.

Há também a definição extraída da proposta do W3C (AUSTIN, 2004): Serviços web são sistemas identificados por uma URI, possuindo interfaces e ligações que são definidas e descritas usando XML. Estas definições podem ser descobertas por outros aplicativos, onde estes aplicativos podem se interagir com os serviços web, usando mensagens baseadas em XML e transportadas por protocolos da Internet.

2.2 ARQUITETURA PARA SERVIÇOS WEB

As funções dos componentes existentes em um Serviço Web estão definidas em sua arquitetura, assim como o relacionamento entre estes componentes e o modelo de trabalho para a construção destes sistemas. Dentro do modelo de trabalho, ou modelo de serviços web, estão as entidades e operações que representam o funcionamento destes sistemas. Também na arquitetura, está a pilha básica de serviços web, que define a maneira comum para construção dos sistemas, visando a interoperabilidade entre eles.

2.2.1 MODELO DE SERVIÇOS WEB

O funcionamento dos serviços web ocorre através da existência de três entidades: consumidor de serviços, provedor de serviços e o registro do serviço, além de operações que são responsáveis em interagir estas entidades: as operações de vincular, publicar e pesquisar por serviços web (KREGER, 2001).

Exemplificando pode-se compreender o funcionamento destas operações e as entidades pelo seguinte cenário: (i) Aloca-se um serviço web já desenvolvido em um provedor de serviços visando sua disponibilização pela Internet ou Intranet. (ii) Este provedor de serviços cria um documento de descrição para permitir a compreensão deste serviço por outras empresas e logo em seguida publica este documento em um local, o registro do serviço, para que possa ser acessado publicamente pelo consumidor de serviços com uma operação de pesquisa. (iii) Depois de encontrado o documento de descrição, o consumidor de serviços interpreta este documento de descrição para poder estabelecer a operação de vínculo com o provedor de serviços e então, utilizar as funcionalidades do serviço web em questão.

Especificamente, entende-se cada uma das entidades da seguinte maneira:

- Provedor de serviços: é o criador do serviço web, entendendo-se também como a plataforma responsável pela disponibilidade da

acessibilidade do serviço pela rede. O provedor de serviços é responsável por realizar a descrição do serviço web, para que este possa ser compreendido, e também por publicar esta descrição para permitir a disponibilidade deste serviço.

- Consumidor de serviços: é quem se utiliza do serviço criado pelo provedor de serviços. Para estabelecer a operação de vínculo com o serviço e então utilizar-se da aplicação é necessário realizar uma pesquisa pelo documento de descrição do serviço, interpretá-lo e finalmente inicializar a interação com o serviço web. O consumidor de serviços pode representar uma pessoa que se utiliza de um navegador para acessar o serviço ou um programa sem uma interface de uso que automaticamente chama pelo serviço, como por exemplo um outro serviço web.

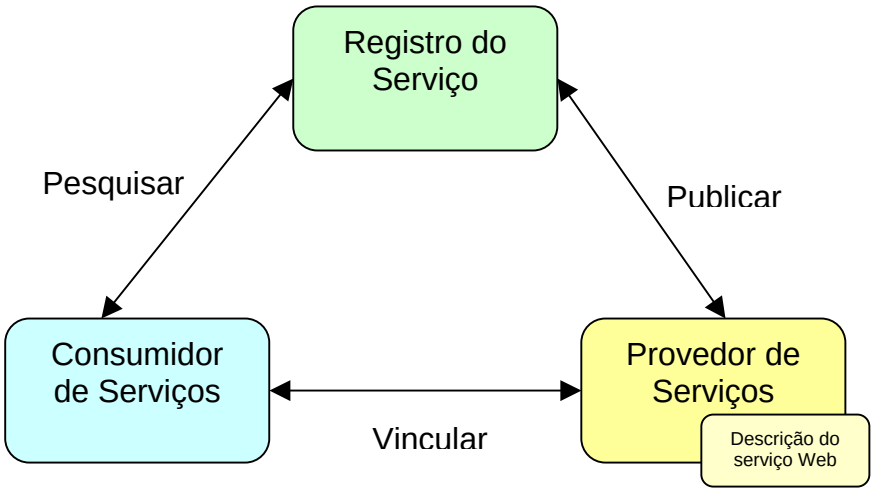

FIGURA 1 – ENTIDADES E OPERAÇÕES DE SERVIÇOS WEB (HENDRICKS, 2002).

- Registro do serviço: é um local centralizado onde são disponibilizadas, pelo provedor de serviços informações sobre um serviço web, através do documento de descrição do serviço. Após a publicação deste documento, torna-se possível a localização do serviço por parte do consumidor de serviços, por uma operação de pesquisa.

A Figura 1 mostra a disponibilidade destas entidades e as operações existentes.

2.2.1.1 OPERAÇÕES PARA O FUNCIONAMENTO DOS SERVIÇOS WEB

Para permitir o funcionamento dos serviços web descrito no modelo básico, há três operações fundamentais: a publicação, a pesquisa e a vinculação. Detalhadamente estas operações devem realizar as seguintes ações:

- Publicação: disponibilização do documento de descrição do serviço web por parte do provedor de serviços para permitir a acessibilidade do serviço pelo consumidor de serviços.

- Pesquisa: operação de busca pelo documento de descrição do serviço web em um registro dos serviços. Esta procura é realizada pelo consumidor de serviços para possibilitar a interpretação e o entendimento das funcionalidades do serviço.

- Vinculação: Após a análise do documento de descrição do serviço realiza-se o vínculo entre o consumidor de serviços e o provedor de serviços. Nesta operação inicia-se a integração entre estas duas entidades permitindo a utilização do serviço web.

2.2.2 PILHA BÁSICA DE SERVIÇOS WEB

Para permitir a interoperabilidade nas aplicações de serviços web é necessário utilizar-se padrões que garantam a execução das operações de publicar, pesquisar e vincular. Estas operações deverão ser realizadas independentemente da plataforma de desenvolvimento e utilização dos serviços web e portanto, precisam se basear em uma pilha básica que será comum entre todos que pertençam ao ambiente de serviços web.

Esta pilha básica representa a arquitetura necessária para possibilitar a execução das operações que permitirão a integração entre as entidades destacadas no modelo conceitual dos serviços web, lembrando que o fator

fundamental é permitir a transparência na execução destas operações, não importando a linguagem utilizada no seu desenvolvimento.

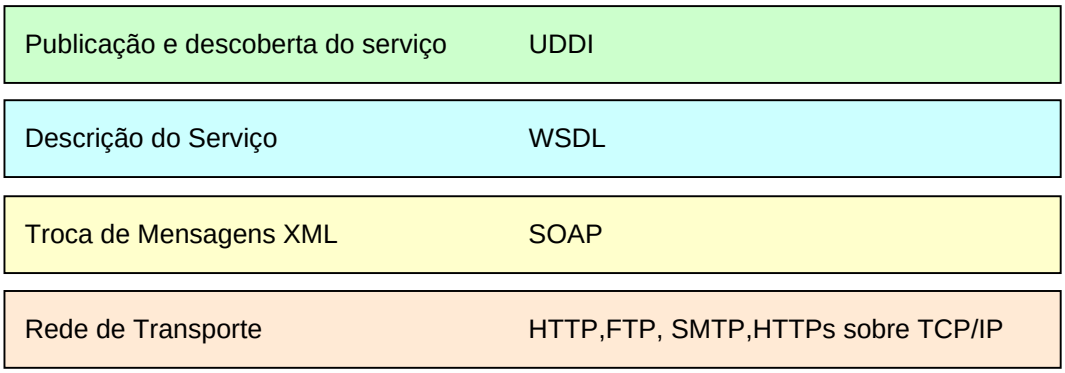

FIGURA 2 – PILHA BÁSICA DE SERVIÇOS WEB (HENDRICKS, 2002).

A Figura 2 mostra a pilha básica de Serviços Web. Na figura, a esquerda de cada bloco encontra-se a definição conceitual das camadas, enquanto a direita estão os rótulos que representam as tecnologias reais que estão presentes em cada camada. Observe também que cada um dos blocos está construído com base nos blocos abaixo dele.

A seguir serão analisados os conceitos de cada uma das camadas, não entrando nos detalhes técnicos dos padrões utilizados.

2.2.2.1 REDE DE TRANSPORTE

A primeira camada da pilha básica de serviços web é constituída pela Rede de Transporte, sendo responsável pela distribuição dos mesmos, deixando-os acessíveis por algum dos protocolos de transportes disponíveis. Há diversos protocolos de transporte que poderão ser utilizados, como o HTTP, SMTP, FTP, entre outros. A escolha do protocolo pode envolver diversos fatores como a segurança, performance, facilidade de uso e disponibilidade, no entanto a definição de como transportar o serviço será transparente para o desenvolvimento do aplicativo. Utiliza-se, atualmente, com maior freqüência o HTTP por ser suportado pela maioria dos navegadores e o de ser vastamente utilizado na Internet. Pode-se também definir se o serviço será disponibilizado em uma Intranet ou publicamente na Internet.

2.2.2.2 TROCA DE MENSAGENS XML

Nesta camada, a segunda da pilha básica de Serviços web, define-se o formato para a realização da troca de mensagens para comunicação entre os aplicativos. O protocolo padrão utilizado com freqüência nos serviços web é o SOAP, um protocolo simples, leve e que permite estabelecer a troca estruturada de informações entre aplicações sobre uma rede independente do sistema operacional e do ambiente de programação utilizado. Outras características importantes do SOAP estão na facilidade da expansibilidade através de XML e na independência do protocolo de transporte dos dados.

O SOAP consiste de três partes: o elemento envelope que define um *framework* para a descrição do conteúdo da mensagem e como ela deve ser processada, o elemento cabeçalho, opcional, que define dados adicionais necessários e o elemento corpo que contém as chamadas e as respostas dos procedimentos (GUDGIN, 2003).

Analisando o funcionamento destas duas camadas apresentadas até o momento, visualiza-se como é realizada a interação entre a aplicação cliente e uma aplicação serviço web, seguindo a ilustração da Figura 3.

As interações apresentadas na figura ocorrem da seguinte maneira: A aplicação cria uma mensagem SOAP que reúne a codificação das operações de serviços web com a finalidade de solicitar a chamada por um serviço específico. A seguir esta mensagem será transportada pelo protocolo de rede, por exemplo o HTTP, e recebido por um servidor SOAP, que será responsável em interpretar a mensagem e realizar a chamada da funcionalidade ao serviço web. Todas as respostas geradas serão novamente codificadas e transportadas de volta, pelo HTTP, para o cliente SOAP.

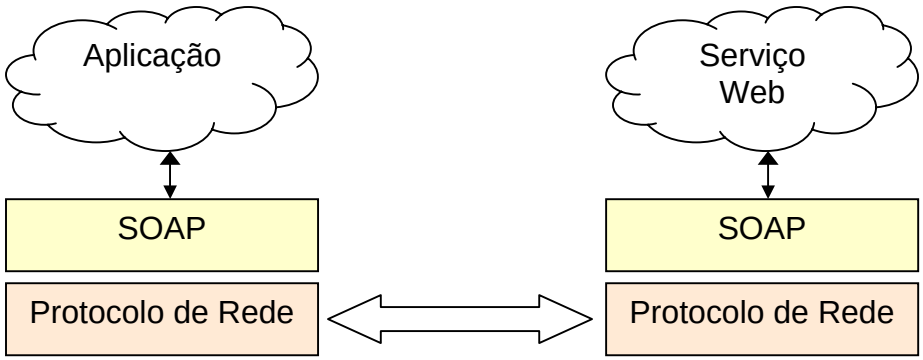

FIGURA 3 – TROCA DE MENSAGENS XML USANDO SOAP (KREGER, 2001)

Uma questão que precisa ser esclarecida a partir de agora é o formato que deverá ser utilizado nas mensagens do consumidor de serviços para garantir a comunicação entre as aplicações. Isto será discutido na próxima camada, a descrição do serviço.

2.2.2.3 DESCRIÇÃO DO SERVIÇO - WSDL

A camada de descrição do serviço fornece, ao provedor de serviços, um mecanismo para comunicar todas as especificações para as chamadas do Web service ao consumidor de serviços, descrevendo assim as funcionalidades existentes. A partir desta descrição, o consumidor entenderá o funcionamento da aplicação a ser chamada e o que será necessário realizar para conseguir interagir-se com esta aplicação, facilitando assim o estudo das solicitações e a programação para integrar o provedor de serviços e o consumidor.

A WSDL - *Web Service Description Language*, ou Linguagem para Descrição de Serviços Web - define o modelo e o formato XML para descrever o serviço web (CHRISTENSEN, 2001). Um documento WSDL descreve um serviço web como uma coleção portas operando independentemente, tanto nas mensagens orientadas ao documento, quanto nas orientadas aos procedimentos. Esta descrição é dividida em duas partes: a definição da interface do serviço e a definição da implementação do serviço.

A definição da interface do serviço é a parte reutilizável da definição do serviço e que representa apenas uma definição abstrata. Nela são conhecidos os elementos que descrevem a interface oferecida por um serviço e como chamá-

la. A interface do serviço contém os elementos: vínculos, tipos de porta, mensagens e tipos, conforme representados na Figura 4.

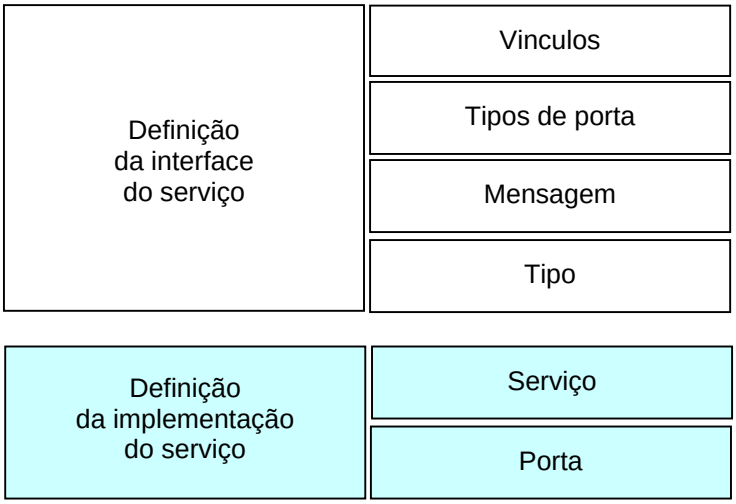

FIGURA 4 – ESTRUTURA DA DESCRIÇÃO DO SERVIÇO (HENDRICKS, 2002).

- Tipos de porta: definição das operações de um serviço web. Uma operação pode ter uma mensagem de entrada e uma mensagem de saída.

- Mensagens: contém a definição dos dados a serem transmitidos.

- Tipos: contém os tipos de dados que estão presentes na mensagem.

- Vínculos: captura o protocolo particular e os elementos tipos de porta, mensagem e tipo mencionados acima. Fornece informações sobre os detalhes da chamada do serviço e como os tipos de portas podem ser chamados.

A definição da implementação do serviço representa como uma interface de serviço específica está implementada por um provedor de serviços. Há os elementos serviço e porta:

- Serviço: O elemento serviço contém uma coleção de elementos porta.

- Porta: contém as informações necessárias para identificar o local para onde deve ser enviada a solicitação real, como exemplo um endereço de

rede ou uma URL. Diversas portas podem ser especificadas, sendo que cada porta é específica para o tipo de vínculo que foi descrito no elemento vínculo.

Com a camada de descrição do serviço, troca de mensagens e rede de transporte, forma-se a pilha de serviços web interoperáveis (KREGER 2001), e se estabelecem as ações necessárias para a realização da operação e comunicação dos serviços web, ou seja, através dessas três camadas iniciais consegue-se integrar e executar as funcionalidades do serviço web.

O consumidor de serviços poderia ter acesso a um documento de descrição, com o envio do arquivo por e-mail, por exemplo, mas certamente este tipo de procedimento restringiria e limitaria a utilização dos serviços web, por isso realiza-se a publicação para possibilitar a descoberta do serviço. Detalhes deste procedimento serão apresentados no próximo tópico.

2.2.2.4 PUBLICAÇÃO E DESCOBERTA DO SERVIÇO - UDDI

A camada de publicação e descoberta de serviços promove a disponibilidade dos serviços web. Pelo documento de descrição do serviço interpretam-se os procedimentos para estabelecer o vinculo com o serviço web, mas não se conhece o acesso a este documento WSDL.

O provedor de serviços realiza a publicação de informações referentes ao seu serviço web em um registro central, que se tornarão disponíveis publicamente aos consumidores de serviços interessados. Além da publicação do documento de descrição, para possibilitar a descoberta do serviço é necessário disponibilizar informações relacionadas ao negócio, como o nome do negócio, o contato comercial, os produtos que estão associados ao serviço, a categoria comercial deste serviço, entre outras.

Estas informações, além de possibilitar a análise e comparação, por parte do consumidor de serviços, ao escolher por um serviço web, oferecem a oportunidade de buscar por serviços usando uma pesquisa mais ampla, com

base em categorias comerciais, por exemplo, e posteriormente detalhar até encontrar os serviços específicos necessários.

A especificação para a publicação e localização de informações comerciais é a *Universal Discovery, Description, and Integration* (UDDI). A especificação UDDI fornece uma estrutura comercial, usada para descrever um determinado negócio, na qual constam as seguintes informações (CLEMENTE, 2004):

- Negócio: Contém as informações comerciais, como a empresa ou o fornecedor do serviço. Contém uma ou mais instâncias da estrutura do serviço. Incluem-se informações como o nome do negócio e uma descrição em várias línguas, informações para contatos e classificações comerciais.

- Serviço: Representa os diferentes serviços web fornecidos por este negócio. Cada serviço contém uma ou mais estruturas de especificação técnica.

- Especificação técnica: contém detalhes técnicos sobre um serviço web. Uma das especificações técnicas é a especificação WSDL.

De modo geral um registro UDDI contém informações sobre os negócios e os serviços oferecidos por ele. O UDDI foi projetado para manter informações arbitrárias sobre um negócio. Ele serve não apenas como um ponto de acesso para informações sobre serviços, como também sobre os próprios negócios.

3 COMPOSIÇÃO E COLABORAÇÃO DE SERVIÇOS WEB

Serão apresentadas neste capítulo as necessidades de se realizar a composição de serviços web, assim como os objetivos e as funções da composição e colaboração. Também serão apresentados os padrões existentes para realizar a orquestração e coreografia dos serviços web. Serão abordados os primeiros trabalhos realizados e descritos com maiores detalhes os três padrões atualmente mais completos. No final deste capítulo será analisado um relacionamento entre estes padrões. O objetivo principal é conhecer as características de todos os padrões para orquestrar e coreografar serviços web, fornecendo assim conhecimento para melhor direcionar nosso trabalho principal, que envolve exclusivamente a coreografia de serviços web.

3.1 SERVIÇOS WEB NO CONTEXTO DE NEGÓCIOS

A arquitetura dos serviços web, por meio das tecnologias HTTP (rede de transporte), SOAP (troca de mensagens) e WSDL (descrição dos serviços), dispõe a interoperabilidade destes serviços, garantindo a realização eficiente da comunicação para a troca de mensagens entre o consumidor de serviços e o provedor do serviço. Este procedimento constitui o processo operacional da utilização de um serviço web, responsabilizando-se então pelo seu funcionamento.

O cenário acima é perfeitamente suficiente se restringir à aplicação de um serviço web único, sem possibilidade de interação com outros tipos de serviços ou mesmo outros tipos de aplicações, ou seja, conectar-se ao serviço e usufruir das funcionalidades exclusivas deste único serviço web.

Porém, se a usabilidade dos serviços web ficassem restritas a este modelo único, certamente inviabilizaria a aplicação desta tecnologia.

A realidade da utilização dos serviços web está direcionada a soluções dinâmicas que podem classificar-se em uma faixa de complexidade de um nível

de baixo até um de alta complexidade. Certamente, em qualquer classificação que um serviço web se posicione, uma regra aplicável é a interação que este serviço necessitará para criar suas funcionalidades, e a quantidade das interações pode estar relativamente vinculada ao índice de complexidade do serviço.

A necessidade das interações também pode ser evidenciada quando se trata a utilização de um serviço web em um contexto de negócios. Um mercado empresarial, potencial consumidor de um serviço web, incorpora estas interações pela própria dinâmica natural existente.

Pode-se relacionar a aplicação de um serviço web em um contexto de negócios com a teoria de Turner (2003), para adicionar características que justifiquem as previsões dessas interações existentes.

Ainda segundo Turner (2003) o software deve se transformar em um serviço que possibilita sua aquisição sob demanda (somente as funções necessárias). O mais interessante e aproveitável neste trabalho até este momento é a camada de integração proposta para possibilitar a transformação do software em um serviço. A Figura 5(a) apresenta uma comparação com um modelo que representa a forma de disponibilização de um software, provendo se funções determinadas e limitadas (só há acesso às funcionalidades que se encontram desenvolvidas no software), e o modelo proposto. Na Figura 5(b), o software ou o serviço se adaptaria a uma camada de integração que seria responsável por interpretar necessidades do sistema, inclusive as funcionalidades de negócios, como por exemplo, a necessidade de invocar um outro serviço externo.

Como os serviços web também são serviços, este modelo torna-se similiar no que tange algumas características de um serviço web desta camada de Serviço de Integração.

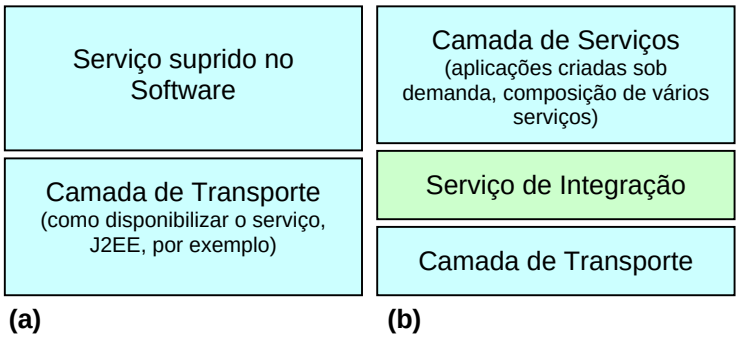

FIGURA 5 – MODELOS DE SERVIÇOS/SOFTWARES (TURNER, 2003)

Esta camada de integração incorpora quatro funções principais: (i) a descrição do serviço, (ii) a descoberta do serviço, (iii) a entrega do serviço; e (iv) a composição do serviço. As três primeiras funções já estão representadas na arquitetura básica do serviço web (as tecnologias HTTP, SOAP, WSDL e UDDI, por exemplo). A quarta função, a de composição, apresenta funcionalidades que são importantes em nosso contexto atual de trabalho, ou seja, a realidade da utilização dos serviços web para negócios. Os detalhes da composição do serviço web será discutido no próximo titulo, no entanto é importante ressaltar a necessidade desta integração quando se tratar de serviços comerciais.

3.2 PILHA DE SERVIÇOS WEB ESTENDIDA

Segundo Aalst (2003) no comércio eletrônico destacam-se duas tendências que trarão oportunidades e pressões para automatizar processos de negócios. Uma é a necessidade de tecnologias que suportem XML para explorar a Internet e a outra é a melhoria da eficiência de processos na perspectiva empresarial.

Para integrar os processos de negócios o suporte às interações oferecidas pelos padrões de troca de mensagens e protocolos de transporte são insuficientes (AALST, 2003). As interações destes processos requerem uma maneira de evidenciar os processos de negócios de forma que possam ser gerenciados.

A necessidade de envolver processos de negócios nos serviços web condiciona a necessidade de linguagens de composição de serviços. Esta composição é tratada na extensão da pilha básica de serviços web, criando suporte a outros padrões que supram as questões provenientes de serviços relacionados no contexto de negócios.

Assim, uma característica importante dos serviços web é a composição (KREGER, 2001). A extensão da pilha básica dos serviços web envolvendo a característica da composição pode ser visualizada na Figura 6.

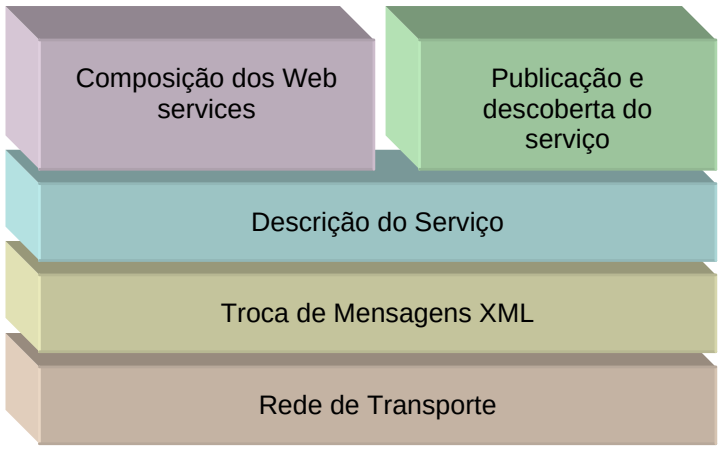

FIGURA 6 – PILHA DE SERVIÇOS WEB EXTENDIDA – COMPOSIÇÃO (AALST, 2003)

Há diversas linguagens que realizam a composição dos serviços web, como a BPEL4WS (*Business Process Execution Language for Web Services*) (ANDREWS, 2003), a WSFL (*Web Services Flow Language*) (LEYMANN, 2001), a XLANG (*Web services for business process design*), WSCI (*Web Services Choreography Language*) e a BPML (*Business Process Modeling Language*). A seguir serão discutidas as características destas linguagens, sendo que agora serão apresentados mais detalhes da funcionalidade e ligação da camada de composição com as demais camadas da pilha básica de serviços web.

Conforme a representação da Figura 6, a camada de composição localiza-se acima da camada de descrição do serviço (WSDL), sendo que todas as operações das camadas abaixo continuam a funcionar sem modificações. A

linguagem de composição interpretará um ou mais documentos de descrição (WSDL) os quais adquirirão as operações do serviço web. Após a relação de todas as operações disponíveis no serviço, realizará o confronto com o processo de negócio do serviço (BPEL4WS, por exemplo) e produzirá como resultado a ordem correta de realização das operações. Este esquema garantirá que o serviço funcione por completo, mesmo contendo diferentes procedimentos e interações com outros serviços. Uma diferença importante entre WSDL e uma linguagem de composição está na verificação da situação de uma operação. Com WSDL se detém o controle da situação de uma operação com relação ao envio e recebimento de mensagens, ou seja, controla-se a operação isolada. Já com uma linguagem de composição, uma operação corresponde ao registro da situação de um processo, sendo este mais complexo que uma simples solicitação de resposta.

3.2.1 ORQUESTRAÇÃO E COREOGRAFIA

Usa-se diversos termos para descrever componentes que são conectados para possibilitar a construção de modelos complexos de processos de negócios. Com a introdução dos serviços web, os termos composição e colaboração foram empregados para descrever a composição dos serviços em um fluxo de processos. Mais recentemente, os termos Orquestração e Coreografia foram propostos e usados (PELTZ, 2003a).

A Orquestração refere-se à descrição da execução dos processos de negócios e das interações estabelecidas internamente e externamente ao serviço web, descrevendo como cada serviço interage com outro serviço web, detalhando todas as trocas de mensagens que serão realizadas neste processo (KOKASH, 2005).

A partir desta descrição, permite-se perceber a lógica do negócio e a ordem de execução das tarefas, podendo também vincular as relações de aplicações e organizações, definindo-se assim modelos de longa duração, transacionais e de múltiplas etapas.

Orquestração sempre representa o controle proveniente da perspectiva de uma parte, significando que serão descritos os processos internos de uma determinada operação, tornando desta forma o documento de orquestração algo privado e não público. Um documento de orquestração irá detalhar como e quando será executada cada tarefa de uma determinada operação, como por exemplo, a descrição das tarefas realizadas para concluir a venda de uma passagem aérea, incluindo taxas de comissão, despesas administrativas da organização e outras. Desta forma, não se deve publicar um documento deste tipo a parceiros externos que não participam diretamente na execução desta operação. Ilustrando, um determinado serviço web que realiza a reserva da passagem aérea e retorna a situação ao seu usuário não deve conhecer os procedimentos realizados na venda da passagem, para que não torne público o modelo de negócios da empresa envolvida.

Isto se diferencia da Coreografia, que é mais colaborativa e permite que cada parte envolvida descreva a sua função na interação, ou seja, não trata das operações internas de uma determinada tarefa e sim das interações em um âmbito global, envolvendo todos os parceiros relacionados com o objetivo determinado. Aproveitando o exemplo acima, a Coreografia estaria presente na descrição da interação entre o serviço web que realiza a venda da passagem aérea e o serviço que faz a solicitação da reserva da passagem aérea.

Coreografia trata da seqüência de mensagens provenientes de múltiplas partes e origens (tipicamente, a troca de mensagens públicas que ocorre entre serviços web) que especificarão um processo de negócio que uma parte deverá executar (DAVIES, 2004). No documento que definirá a coreografia permite-se que cada participante envolvido descreva a sua parte na interação.

Relacionando com a definição da pilha de serviços web estendida, com a Coreografia e a Orquestração realiza-se a composição e colaboração dos serviços web (YUSHI, 2004).

A Figura 7 ilustra um nível mais alto que representa a possível relação entre Orquestração e Coreografia. Orquestração refere-se a um processo em

execução, enquanto que a Coreografia trilha a seqüência de mensagens entre as partes envolvidas.

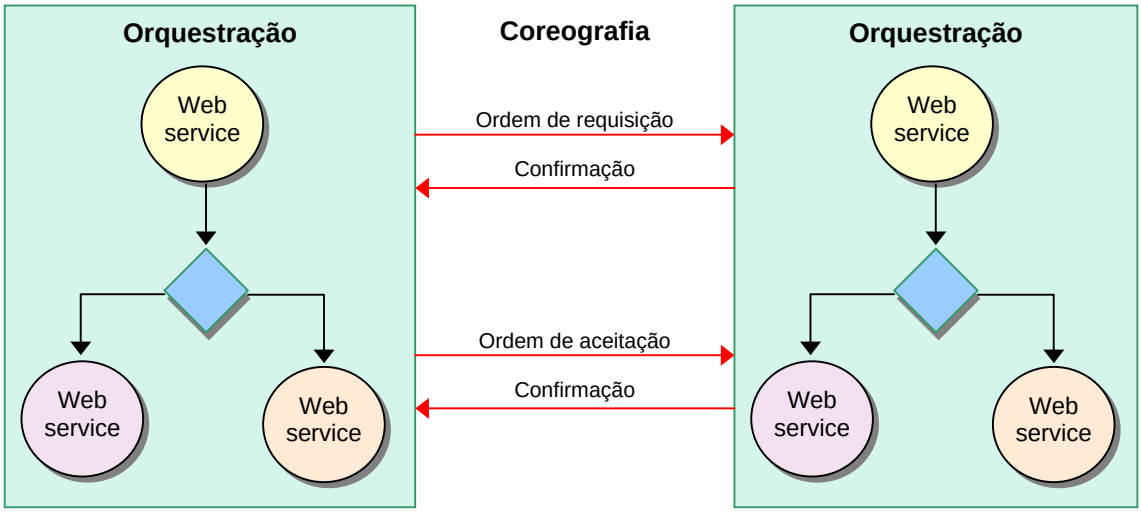

FIGURA 7 – ORQUESTRAÇÃO E COREOGRAFIA (PELTZ, 2003A)

3.3 PADRÕES EXISTENTES PARA ORQUESTRAR E COREOGRAFAR SERVIÇOS WEB.

Existem diversos padrões que podem ser utilizados para especificar a colaboração e composição de serviços web. A razão da existência desses diversos padrões está relacionada com a evolução em pesquisas e descobertas de necessidades especiais para a realização da colaboração. Cada especificação obtém características especiais e diferentes entre si, como exemplo padrões que realizam somente a orquestração e outros a coreografia.

Para compreender melhor a especificação da colaboração de serviços web, serão expostos a seguir os principais padrões utilizados.

3.3.1 PRIMEIROS TRABALHOS (XLANG, WSFL)

Os primeiros trabalhos na orquestração de serviços web incluem o XLANG (THATTE, 2001), desenvolvido pela Microsoft para o Microsoft BizTalk Server. XLANG tinha o foco na criação de processos de negócios e interações entre os provedores de Serviços web. A especificação proporcionava suporte ao controle do fluxo de processos de forma seqüencial, paralela e condicional.

Também incluía uma robusta facilidade no tratamento de exceções, com suporte as transações de longa duração e compensações. XLANG usava WSDL para descrever a interface do processo.

O *Web Services Flow Language* (WSFL) é uma proposta da IBM para descrever fluxos de processos públicos e privados. WSFL define a ordem especifica das operações (modelo global) e a troca de dados de um processo especifico (modelo processual). O modelo processual representa a série de atividades no processo, enquanto que o modelo global vincula cada atividade com a instância do serviço web específico. Um documento WSFL também pode ser exposto com uma interface WSDL permitindo uma decomposição recursiva. WSFL suporta o tratamento de exceções, mas não realiza diretamente o controle de transações.

Como já citado uma variedade de especificações e padrões foram introduzidos para realizar a orquestração e coreografia, mas neste trabalho serão priorizadas as três iniciativas mais recentes.

3.3.2 BPEL4WS (BUSINESS PROCESS EXECUTION LANGUAGE FOR WEB SERVICES)

BPEL4WS ou resumidamente representado por BPEL, é uma especificação proposta pela IBM, Microsoft, Siebel Systems, BEA e SAP, que especifica o comportamento dos serviços web na interação de processos de negócios (ANDREWS, 2003). Esta especificação fornece uma gramática baseada em XML para descrever o controle lógico requerido para coordenar um serviço web participante de um processo colaborativo. Esta gramática pode ser interpretada e executada por uma máquina (*engine*) de orquestração, a qual é controlada por uma das partes participantes sendo que este mecanismo também realiza uma compensação em todo o processo quando ocorrem erros.

BPEL é uma camada acima da WSDL, sendo que o WSDL define as operações específicas permitidas e o BPEL define como estas operações podem ser seqüenciadas. WSDL descreve as portas para cada processo BPEL, e os tipos de dados WSDL descrevem as informações que passam entre as requisições.

BPEL provê suporte para os processos executáveis, assim como para os processos de negócios abstratos. Processos executáveis representam o comportamento entre os participantes da interação de um negócio específico, modelando essencialmente a colaboração privada das operações. Um processo abstrato, ou protocolo de negócios especifica a troca de mensagens públicas entre os participantes. Os protocolos de negócios não são executáveis e não transportam detalhes internos de um processo colaborativo. Os processos executáveis representam o suporte à orquestração, enquanto o protocolo de negócios enfoca mais a coreografia dos serviços.

Esta especificação inclui suporte as atividades básicas e estruturadas (PELTZ, 2003b). Atividade básica é uma instrução que interage com algo externo do próprio processo. Por exemplo, atividades básicas tratariam de receber e responder pedidos de mensagens, assim como invocar serviços externos. Um cenário típico está no recebimento de uma mensagem por um processo BPEL, onde este processo precisaria invocar uma séria de serviços externos para colher informações e depois responder no mesmo modelo. A Figura 8 representa o processo de mensagens das atividades básicas de receber, responder e invocar serviços web.

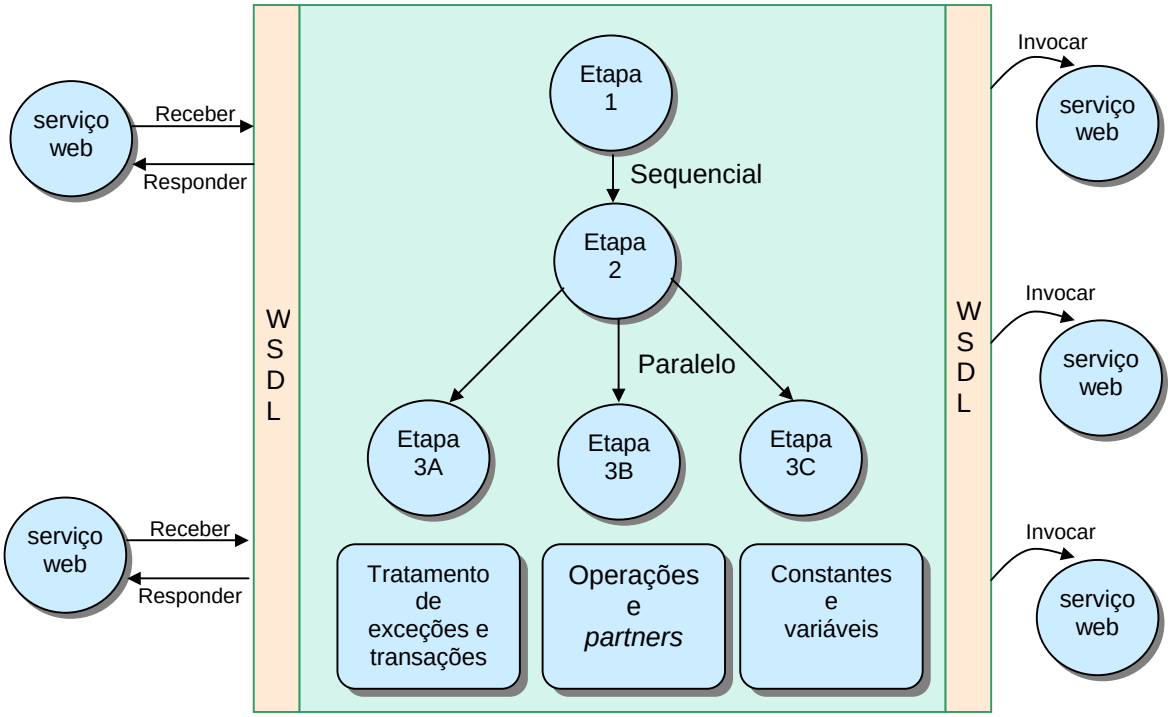

FIGURA 8 – BPEL4WS (PELTZ, 2003A)

Atividades estruturadas prescrevem a ordem na qual uma coleção de atividades deve ocorrer. Estas atividades descrevem a criação de um processo de negócio pela composição das atividades básicas, permitindo executar tarefas de recursividade, fluxo dos dados, tratamento de erros e a coordenação da troca de mensagens entre os processos instanciados.

Outros dois elementos importantes da especificação BPEL são o elemento *variable* e o elemento *partnerLink*. *Variable* fornece a especificação para realizar a troca dos dados no fluxo de mensagens, enquanto que o *partnerLink* define a interface de interação entre os serviços, especificando as funções de cada serviço no contexto da colaboração e também especifica onde cada serviço receberá as mensagens. BPEL também fornece um mecanismo robusto para o tratamento de exceções e transações.

3.3.3 WSCI (WEB SERVICES CHOREOGRAPHY INTERFACE)

A especificação WSCI, desenvolvida pela Sun, SAP, BEA e Intalio, define uma linguagem baseada em XML para a colaboração de serviços web. Esta especificação descreve a base para a troca coletiva de mensagens, seqüenciamento de regras, tratamento de exceções e a colaboração dinâmica existente em um serviço.

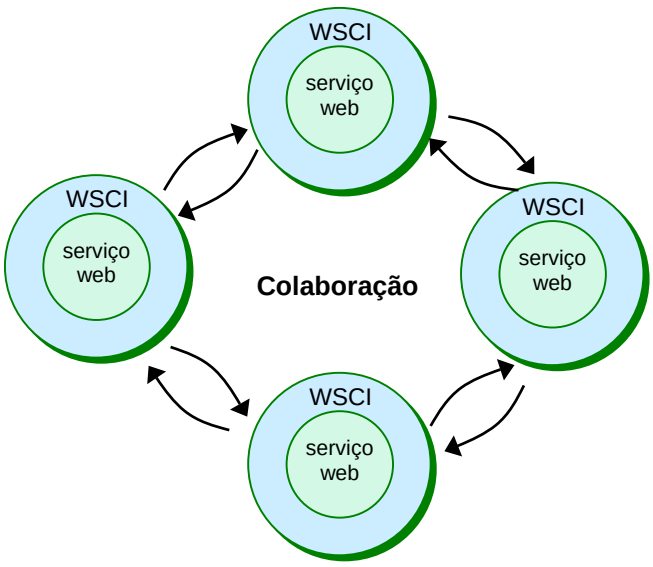

FIGURA 9 – WSCI (PELTZ, 2003)

Observa-se através da Figura 9 que o WSCI descreve o comportamento entre serviços web, não realizando a definição da execução dos processos de negócios, como o BPEL4WS. Além disso, um simples documento WSCI descreve a participação na troca de mensagens de somente um serviço envolvido no ambiente colaborativo.

O WSCI descreve o fluxo da troca de mensagens de um serviço web participante em interações de coreografias com outros serviços (ARKIN, 2002). Esta descrição permite desenvolvedores e ferramentas descreverem e comporem uma visão global da dinâmica da troca de mensagens, assim como entenderem as interações com o serviço web.

WSCI estende WSDL descrevendo como realizar a coreografia das operações WSDL disponíveis, ou seja, um documento WSDL descreve as portas disponíveis de cada serviço e WSCI descreve as interações entre operações WSDL. A seguir estão as principais características e definições desta especificação.

3.3.3.1 CARACTERÍSTICAS DA INTERFACE WSCI

Para possibilitar o gerenciamento da colaboração dos serviços web, operações individuais precisam permitir o transporte de informações suficientes para conhecer como os serviços serão utilizados em um determinado cenário dentro de uma ordem que os permitirá participar de processos mais complexos.

O WSCI realiza este gerenciamento pela definição de uma camada acima da pilha básica do serviço web, que descreve seu comportamento necessário relativo a um ambiente de troca de mensagens.

O WSCI suporta as seguintes funções existentes no contexto da troca de mensagens:

- Coreografia das mensagens: a interface WSCI descreve a ordem na qual as mensagens podem ser enviadas ou recebidas em uma determinada troca de mensagens, as regras para administrar cada

ordem e os limites da troca de mensagem (quando se inicia e quando termina). O WSCI não determinada a troca de mensagens que ocorrem no interior do serviço web.

- Transações e compensações: A interface WSCI descreve quais operações são executadas de forma transacional e, desta forma, notifica os outros participantes da capacidade daquele serviço web executar as tarefas de uma forma “todas ou nenhuma”. Esta capacidade também permite aos serviços web relacionarem-se em transações distribuídas com outros serviços.

- Tratamento de exceções: A interface WSCI descreve como um serviço web reagirá quando acontecerem condições excepcionais, fornecendo uma descrição de um comportamento alternativo, porém não se define o mecanismo para qual o serviço web irá controlar a situação ou erro.

- Gerenciamentos de *Thread*: A interface WSCI descreve se um serviço web é capaz de gerenciar múltiplas conversas (na mesma troca de mensagem) com o mesmo parceiro ou outros diferentes parceiros.WSCI não define como o serviço web é capaz de controlar as conversações múltiplas de forma concorrente.

- Propriedades: A interface WSCI descreve elementos que influenciam o comportamento de observar o serviço web, como as trocas a partir de valores de tempo de execução de alguma das partes das mensagens.

- Conectores: A interface WSCI descreve como as operações executadas por diferentes serviços web representarão uma mesma troca de mensagem que na prática estão juntas. WSCI permite o mapeamento das operações “consumidas” de um serviço web até as operações “produzidas” de outro serviço dentro de uma ordem para construir um modelo global de interação. Este é o significado de WSCI *Global Model* que também possibilita descrever como interfaces de diferentes serviços participantes de uma mesma troca de mensagens podem ser unidas para construir um modelo final de interações.

- Ambiente operacional: A interface WSCI descreve como o mesmo serviço web se comporta em um ambiente de troca de mensagens diferentes. Diferentes interfaces WSCI (considerando o comportamento) podem ser associadas com diferentes ambientes operacionais na qual o serviço web participa.

Resumindo, a interface WSCI descreve todos os objetos necessários para fornecer:

- Externamente, a visão de como os serviços web atuam com outros serviços no ambiente de uma determinada troca de mensagem, ou seja, como o serviço é visível do lado de fora no ambiente de uma determinada troca de mensagens;

- A visão da troca de mensagens visto pelo serviço, ou seja, como o serviço percebe o comportamento do lado externo no ambiente de uma determinada troca de mensagens.

3.3.4 BPML (BUSINESS PROCESS MANAGEMENT LANGUAGE)

BPML é uma linguagem baseada em XML para descrever processos de negócios (ARKIN, 2002). No principio, BPML foi desenvolvido para descrever processos de negócios executados por sistemas BPMS (*Business Process Management System*), sendo que a partir da versão apresentada em novembro de 2002, incorporou-se suporte ao WSCI (LARA, 2003).

BPML e WSCI compartilham o mesmo modelo básico de execução dos processos, entretanto WSCI descreve as interações das mensagens públicas (coreografia) enquanto o BPML realiza as implementações privadas.

BPML fornece a construção de um fluxo de processos e atividades similar ao BPEL. Há as atividades básicas para enviar, receber e invocar serviços, assim como atividades estruturadas como escolhas condicionais, atividades paralelas e seqüenciais, relacionamentos e recursividade. BPML também suporta o agendamento de tarefas para momentos específicos.

A linguagem foi designada para gerenciar processos de longa duração e também inclui características para suportar processos permanentes. BPML suporta também a composição para construir processos de negócios agregados a sub-processos.

BPML suporta a execução de transações curtas e longas, usando técnicas semelhante ao BPEL para controlar as regras de compensações. Finalizando, a linguagem inclui um robusto mecanismo para tratamento de exceções.

3.3.5 RELACIONAMENTO ENTRE OS PADRÕES APRESENTADOS

Relacionando as características dos padrões existentes para a definição da orquestração e coreografia dos serviços web, encontram-se similaridades que podem levantar questões sobre a importância da utilização de um determinado padrão no contexto da colaboração dos processos de negócios.

BPEL e BPML fornecem capacidades para definir a execução de um processo de negócio, enquanto WSCI concentra-se na troca de mensagens públicas entre os serviços web. WSCI tende-se mais para a coreografia e colaboração, requerendo que cada participante, na troca de mensagens, defina uma interface WSCI.

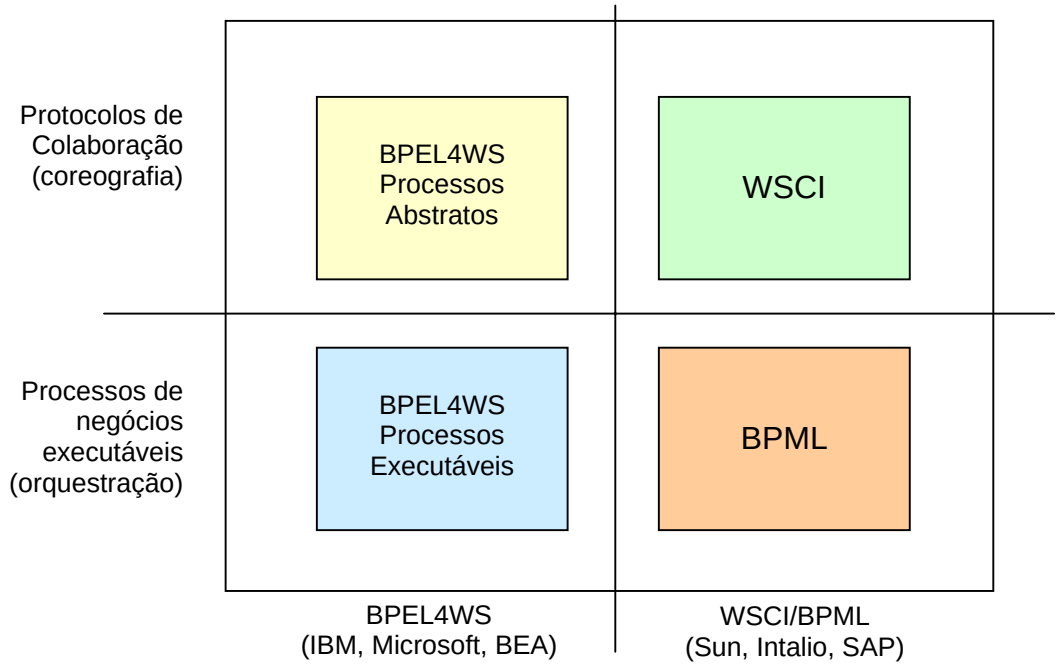

FIGURA 10 – RELAÇÃO ENTRE BPEL4WS, WSCI E BPML (PELTZ, 2003A)

Um cenário proposto por Peltz (2003a) possibilita melhorar a compreensão da aplicação de um determinado padrão, tendo como base as diferenças existentes nos padrões, na qual realiza-se uma categorização destas especificações. Este cenário está representado na Figura 10. O eixo Y distingue os protocolos de colaboração e os processos de negócios executáveis. Um protocolo de colaboração referencia-se a coreografia das trocas de mensagens entre as múltiplas partes do negócio, enquanto que os processos executáveis representam o fluxo de trabalho privado. O eixo X apresenta duas iniciativas do contexto global: BPEL e WSCI/BPML juntos. BPEL suporta tanto os processos executáveis quanto os colaborativos, enquanto que o BPML suporta os processos executáveis e o WSCI a coreografia entre os Serviços web.

Peltz (2003a) relaciona também vantagens encontradas em cada especificação procurando apontar uma escolha para realizar a colaboração dos serviços web. BPEL4WS apresenta um número maior de empresas no desenvolvimento da especificação, dificultando a aceitação do BPML/WSCI. Há mais documentação e ferramentas de desenvolvimento disponíveis para o BPEL4WS. WSCI apresenta-se mais robusto para a colaboração/coreografia, comparando-se com o BPEL4WS e a submissão ao W3C oferece a condição de uso sem pagamento de direitos autorais. Há a afirmação de um potencial cenário incluindo BPEL4WS para a linguagem de processos executáveis com WSCI para abstração/processos colaborativos.

Smith (2003) compartilha dos mesmos princípios, reforçando a atenção aos princípios iniciais de cada especificação. BPEL para controlar o fluxo de trabalho privado e WSCI para definir a orquestração dos serviços web.

3.3.6 GRUPO DE COREOGRAFIA DO W3C (WSCI E WS-CDL)

A especificação WSCI corresponde aos primeiros trabalhos elaborados pelo grupo de coreografia do W3C. Em meados de 2004, após mudanças na equipe de desenvolvimento dos trabalhos de coreografia, disponibilizou-se uma nova

publicação defendendo a forma de realizar a coreografia dos serviços web, a WS-CDL (comentada no capítulo 5).

Manteve-se todos os princípios anteriormente estabelecidos, sendo que a especificação WSCI é considerada a base da atual WS-CDL. As publicações referente a especificação WSCI continuam disponíveis, no entanto não há qualquer modificação desde a concentração dos trabalhos sobre a WS-CDL. Dessa forma, a WS-CDL passa ser referência para estabelecer a coreografia dos serviços web.

4 APLICAÇÃO DE SERVIÇOS WEB

Compreender o funcionamento dos serviços web corresponde a um requisito mínimo para entender a necessidade da colaboração entres eles. Neste aspecto, serão abordadas neste capítulo as etapas básicas para construção e funcionamento de uma aplicação de serviços web, detalhando as características das tecnologias empregadas diretamente no desenvolvimento desses serviços. Serão apresentados os principais elementos das tecnologias que possibilitam desenvolver um serviço web, incluindo descrições das sintaxes dos padrões.

Ao conhecer os detalhes das tecnologias empregadas na construção dos serviços web será possível compreender todas as atividades que necessitam ser implementadas para possibilitar a aplicação prática dos serviços web. Este conhecimento torna possível mensurar e elucidar as principais funcionalidades necessárias para desenvolver ferramentas que irão interagir com serviços web.

4.1 XML

A linguagem XML descreve uma classe de objetos de dados chamada documento XML e também descreve de certa forma o comportamento dos programas que processam esses documentos (BRAY, 2004). Da mesma forma que HTML, o XML utiliza etiquetas (*tags*) para representar os dados, porém existem várias diferenças entre estes dois padrões.

Os documentos XML são compostos de unidades de armazenamento denominados elementos, sendo dispostos através de etiquetas, que por sua vez definem o modelo do documento e sua estrutura lógica.

A Figura 11 apresenta um documento XML que descreve algumas características de um cliente (`<CLIENTE>`): o nome (`<NOME>`) e a renda

(<RENDA>). Nota-se que <CLIENTE> é um elemento do documento XML, tendo <NOME> e <RENDA> como sub-elementos.

```
<?xml version="1.1"?>
<EMPRESA>
  <CLIENTE>
    <NOME>
      Ronaldo Silva
    </NOME>
    <RENDA>
      3000
    </RENDA>
  </CLIENTE>
  <CLIENTE>
    <NOME>
      Luciana Oliveira
    </NOME>
    <RENDA>
      900
    </RENDA>
  </CLIENTE>
</EMPRESA>
```

FIGURA 11 – EXEMPLO DE DOCUMENTO XML – EMPRESA.XML

O conteúdo de um documento XML sempre será definido pelo usuário, assim qualquer elemento pode ser constituído, independente da sua integridade. Neste aspecto é importante validar um documento XML, definindo previamente um esquema que deverá ser seguido. As linguagens utilizadas para definição desses esquemas são a *Document Type Definition* (DTD) e XML *Schema.*

4.2 DTD E XML SCHEMA

Validar um documento XML significa especificar os requisitos estruturais, indicando regras que apontam exatamente quais os tipos de conteúdos dos elementos e quais sub-elementos podem ocorrer dentro destes elementos (ordem, cardinalidade, etc). A especificação XML 1.1 recomendada pelo W3C determina que essas regras sejam expressas por DTDs (BRAY, 2004).

A Figura 12 mostra um exemplo da definição DTD do documento apresentando na Figura 11 (clientes.xml). Nota-se que cada instrução `<!ELEMENT>` indica a definição de um novo elemento. Este DTD indica que `empresa` será o elemento de mais alto nível e que deve conter pelo menos um cliente, e que cada elemento `cliente` deve conter os elementos `nome` e `renda` (segunda linha). A terceira e quarta linha indicam que os elementos `nome` e `renda` armazenarão dados.

```
<!ELEMENT empresa (cliente)+>
<!ELEMENT cliente (nome, renda)>
<!ELEMENT nome (#PCDATA)>
<!ELEMENT renda (#PCDATA)>
```

FIGURA 12 – EXEMPLO DE ARQUIVO DTD – EMPRESA.DTD

No entanto os DTDs apresentam algumas limitações, como a simplicidade na expressão de tipos de dados (é possível especificar que um elemento deve conter PCDATA [*string*] – mas não que ele deve conter, por exemplo, um inteiro positivo. Para suprir essas deficiências surgiram os XML *Schemas*, linguagem baseada em XML para definição de representação de dados (FALLSIDE, 2004). O propósito de um esquema XML é definir e descrever uma classe de documentos XML através da utilização dos componentes da linguagem para restringir e documentar o significado, a utilização e as relações entre as partes constituintes do documentos: tipos de dados, elementos e seu conteúdo e atributos.

O suporte a diversos tipos de dados, entre eles tipos simples pré-definidos (*string, int, float, decimal, timeInstant, nonNegativeInteger*) e tipos complexos que podem ser definidos através da derivação de expressões regulares, potencializam o uso dos esquemas XML.

A Figura 13 apresenta um documento esquema XML substituto ao arquivo empresa.dtd (Figura 12). Neste arquivo, o XML *Schema* define os mesmos elementos, mas de uma forma hierárquica: `xs:element` marca a definição de

um novo elemento (`empresa`), composto de outro elemento complexo (`cliente`), que é composto de uma seqüência de elementos simples: `nome` tipo `xs:string` e renda tipo `xs:decimal`.

```
<?xml version="1.0"?>
<xs:schema>
  <xs:element name="empresa" type=”tipoEmpresa”/>
  <xs:complexType name=”tipoEmpresa”>
    <xs:sequence>
      <xs:element name="cliente" type="tipoCliente"/>
    </xs:sequence>
  </xs:complexType>
  <xs:complexType name=”tipoCliente”>
    <xs:sequence>
      <xs:element name="nome" type="xs:string"/>
      <xs:element name="renda" type="xs:decimal"/>
    </xs:sequence>
  </xs:complexType>
</xs:schema>
```

FIGURA 13 – EXEMPLO XML SCHEMA – EMPRESA.XSD

4.3 SOAP (SIMPLE OBJECT ACCESS PROTOCOL)

Embora XML seja feito para a troca de dados, ele sozinho não é suficiente para o intercâmbio de informações através da Internet. O protocolo SOAP provê um mecanismo simples e leve para troca de informações estruturadas entre pares em um ambiente distribuído e descentralizado usando XML (MITRA, 2003).

A comunicação se dará através do transporte de um envelope SOAP, contendo a mensagem, entre o servidor (aplicação serviço web) e o cliente. Envelopes SOAP podem trafegar sobre outros protocolos, como o HTTP, o SMTP ou algum outro desejado. Esta característica facilita o transporte das mensagens sobre as redes, principalmente quanto a transposição de *firewalls,* já que não necessita de uma porta especial habilitada.

Uma mensagem SOAP (Figura 14) é um documento XML comum, contendo os seguintes elementos:

- Envelope: elemento obrigatório que identifica o documento XML como sendo uma mensagem SOAP.
- Cabeçalho (*Header*): elemento opcional que permite adicionar recursos como autenticação e gerenciamento de transação.
- Corpo (*Body*): elemento obrigatório que contém a carga útil (informações) que deve ser recebida pelo receptor da mensagem.
- Falha (*Fault*): elemento opcional usado para o tratamento de erros em uma aplicação SOAP.

```
<?xml version="1.0"?>
<soap:Envelope xmlns:soap="http://www.w3.org/2001/12/soap-
envelope" soap:encodingStyle="http://www.w3.org/2001/12/soap-
encoding">
   <soap:Header>
      ...
      ...
   </soap:Header>
   <soap:Body>
      ...
      ...
      <soap:Fault>
            ...
            ...
      </soap:Fault>
   </soap:Body>
</soap:Envelope>
```

FIGURA 14 – EXEMPLO DA ESTRUTURA DE UMA MENSAGEM SOAP

4.4 WSDL (WEB SERVICE DESCRIPTION LANGUAGE)

WSDL é uma linguagem baseada em XML usada para descrever serviços web e a forma como acessá-los (CHRISTENSEN, 2001). WSDL responde três perguntas sobre um serviço web: a) o que é o serviço; b) onde encontrá-lo e c) como chamá-lo. A descrição do serviço web pelo WSDL é realizada em dois

estágios: abstrato e concreto. Sumariamente conceituando, no nível abstrato se define o envio e o recebimento das mensagens do serviço web, e no nível concreto especifica o transporte e os detalhes das interfaces do serviço. Os principais elementos contidos em um documento WSDL estão listados a seguir:

- <*type*>: define os tipos de dados que o serviço web utiliza;
- <*message*>: definem as mensagens usadas (enviadas/recebidas) pelo serviço web;
- <*portType*>: Operações realizadas pelo serviço web;
- <*binding*>: descreve como um tipo de porta é mapeado para um protocolo de chamada de rede, como o SOAP.
- <*service*> e seu elemento <*port*>: incluem a localização da implementação de um serviço na rede.

A Figura 15 resume os principais elementos da WSDL que podem ocorrer em um documento.

```
<definitions>
  <types>
    [definições de tipos]
  </types>
  <message>
    [definições de mensagens]
  </message>
  <portType>
    [definições de portas]
  </portType>
  <binding>
    [definições de dos protocolos de rede para invocação]
  </binding>
  <service>
    <port>
      [definições de localização do serviço]
    </port>
  </service>
</definitions>
```

FIGURA 15 – ESTRUTURA PRINCIPAL DE UM DOCUMENTO WSDL

A Figura 16 mostra um exemplo de documento WSDL contendo todos os elementos, com exceção do `type`.

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions name="ServicoOlaMundo"
   targetNamespace="http://localhost:8080/wsdl"
     xmlns:tns="http://localhost:8080/wsdl"
     xmlns="http://schemas.xmlsoap.org/wsdl/">
  <types/>
  <message name="olaMundo">
    <part name="String_1" type="xsd:string"/>
  </message>
  <message name="olaMundoResponse">
    <part name="result" type="xsd:string"/>
  </message>
  <portType name="olaIF">
    <operation name="olaMundo">
      <input message="tns:olaMundo"/>
      <output message="tns:olaMundoResponse"/>
    </operation>
  </portType>
  <binding name="olaIFBinding" type="tns:olaIF">
    <operation name="olaMundo">
      <input>
        <soap:body
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
use="encoded" namespace=" http://localhost:8080/wsdl"/>
      </input>
      <output>
        <soap:body
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
use="encoded" namespace=" http://localhost:8080/wsdl"/>
      </output>
      <soap:operation soapAction=""/>
    </operation>
    <soap:binding
        transport="http://schemas.xmlsoap.org/soap/http"
        style="rpc"/>
  </binding>
  <service name="WService">
    <port name="olaIFPort" binding="tns:olaIFBinding">
      <soap:address
location="http://localhost:8080/wservice"/>
    </port>
  </service>
</definitions>
```

FIGURA 16 – EXEMPLO DE DOCUMENTO WSDL – OLAMUNDO.WSDL

O documento olamundo.wsdl indica que o serviço possui duas mensagens: `olaMundo` e `olaMundoResponse`. O elemento `portType` indica quais são as operações disponibilizadas pelo serviço, enquanto que o elemento `operation` especifica se as mensagens são de entrada ou saída. No elemento `binding` são indicados os esquemas que devem ser usados para as operações que foram especificadas no `portType`, além do tipo e maneira como devem ser transportados os dados. O elemento `port` indica o local do serviço na rede.

4.5 UDDI (UNIVERSAL DESCRIPTION DISCOVERY AND INTEGRATION)

Uma das características que destacam os serviços web é justamente conseguir utilizar serviços já implantados por outros desenvolvedores, mas para isso acontecer é necessário conhecer quais os serviços que estão disponíveis para utilização. No ambiente comercial, talvez seja possível esta informação mediante o contato direto com os parceiros envolvidos em um processo. Este contato pode se estabelecer por meio de um e-mail, um telefonema ou uma visita in-loco. No entanto este tipo de abordagem pode despender um tempo muito grande, e mesmo assim há possibilidade de falhas na comunicação. Para piorar, quando existem diversos parceiros (e não apenas dois) cada um pode apresentar uma linguagem diferente na apresentação de seus serviços disponíveis, gerando assim um ambiente confuso e heterogêneo.

UDDI é um *framework* independente de plataforma para descrever, descobrir e integrar serviços de negócio através da Internet. A abordagem de UDDI é baseada em um registro de negócios e descrições de seus serviços, em formato XML. Os registros UDDI representam um diretório de interfaces de serviços web descritas em WSDL.

O modelo central usado pelos registros UDDI é definido com um XML *Schema*, definindo quatro tipos principais de informação: informação de negócio (*businessEntity*), informação de serviço (*businessService*), informação de

binding (*bindingTemplate*) e informações sobre especificação de serviços (*tModel*). Nas figuras abaixo estão dispostas os detalhes de cada uma delas.

A *businessEntity* representa a informação armazenada sobre a empresa ou entidade que publica o serviço. Segundo Clement (2004), a estrutura desta informação é constituída conforme representação da Figura 17, destacando o elemento *businessEntity* como sendo a chave do registro UDDI. No campo *name* deve ser armazenado o nome da entidade ou empresa que está publicando o registro e no campo *description* pode ser colocado uma breve descrição dos negócios da empresa.

```
<element name = “businessEntity”>
  <type content = “elementOnly”>
    <group order = “seq”>
     <element ref = “discoveryURLs” minOccurs = “0” maxOccurs = “1” />
     <element ref = “name” />
     <element ref = “description” minOccurs = “0” maxOccurs = “*” />
     <element ref = “contacts” minOccurs = “0” maxOccurs = “1” />
     <element ref = “businessServices” minOccurs = “0” maxOccurs = “1” />
     <element ref = “identifiedBag” minOccurs = “0” maxOccurs = “1” />
     <element ref = “categoryBag” minOccurs = “0” maxOccurs = “1” />
   </group>
   <attribute name = “businessKey” minOccurs = “1” type = “string” />
    <attribute name = “operator” type = “string” />
    <attribute name = “authorizedName” type = “string” />
  </type>
</element>
```

FIGURA 17 – ESTRUTURA DO ELEMENTO BUSINESSENTITY

A Figura 18, que representa a estrutura *businessService*, mostra informações sobre cada um dos serviços oferecidos pela empresa. Para cada *businessEntity* pode-se ter vários *businessService*. Este relacionamento é realizado através do campo *businessKey*. O campo *serviceKey* corresponde a chave do registro. O campo *name* indica o nome do serviço e o campo *description* a descrição do serviço.

```
<element name = “businessService”>
  <type content = “elementOnly”>
    <group order = “seq”>
      <element ref= “name” />
      <element ref= “description” minOccurs = “0” maxOccurs = “*”/>
      <element ref= “contacts” minOccurs = “0” maxOccurs = “1” />
      <element ref= “bindingTemplates” />
      <element ref= “categoryBag” minOccurs = “0” maxOccurs = “1”/>
    </group>
    <attribute name= “serviceKey” minOccurs = “1” type = “string”/>
    <attribute name= “businessKey” type = “string” />
  </type>
</element>
```

FIGURA 18 – ESTRUTURA DO ELEMENTO BUSINESSSERVICE

A estrutura *bingindTemplate* (Figura 19) é composta por informações técnicas sobre serviços web. O campo *bindingKey* corresponde a chave do registro e o campo *serviceKey* ao relacionamento com a estrutura *businessService.* Em *accessPoint* deve ser alocado o ponto de entrada para o serviço web, que normalmente corresponde a uma URL.

```
<element name = “bindingTemplate”>
 <type content = “elementOnly”>
  <group order = “seq”>
    <element ref = “description” minOccurs = “0” maxOccurs = “*” />
     <group order = “choice”>
      <element ref=“accessPoint” minOccurs=“0” maxOccurs= “1”/>
      <element ref=“hostingRedirector” minOccurs=“0” maxOccurs=“1”/>
     </group>
    <element ref = “tModelInstanceDetails” />
  </group>
  <attribute name = “bindingKey” minOccurs = “1” type = “string” />
  <attribute name = “serviceKey” type = “string” />
 </type>
</element>
```

FIGURA 19 – ESTRUTURA DO ELEMENTO BINDINGTEMPLATE

A estrutura *tModel* (Figura 20) armazena os metadados do serviço. É uma estrutura relativamente simples, tendo os campo *tModelKey, name, description* e URL, esta última é normalmente utilizada para encontrar mais informações sobre o serviço.

```
<element name = “tModel”>
  <type content = “elementOnly”>
    <group order = “seq”>
     <element ref=“name” />
     <element ref=“description” minOccurs = “0” maxOccurs = “*” />
     <element ref=“overviewDoc” minOccurs = “0” maxOccurs = “1” />
     <element ref=“identifierBag” minOccurs = “0” maxOccurs = “1” />
     <element ref=“categoryBag” minOccurs = “0” maxOccurs = “1” />
    </group>
    <attribute name = “tModelKey” minOccurs = “1” type = “string” />
    <attribute name = “operator” type = “string” />
    <attribute name = “autorizedName” type = “string” />
  </type>
</element>
```

FIGURA 20 – ESTRUTURA DO ELEMENTO TMODEL

5 WS-CDL

Neste capítulo será detalhada a linguagem WS-CDL para a descrição de coreografia de serviços web. A linguagem WS-CDL é o foco principal desta dissertação de mestrado. O principal objetivo desta linguagem é garantir a interoperabilidade entre diversos serviços web que compartilham de um único propósito ou resultado final, através da coordenação da execução dos serviços condicionando a interpretação e influência das etapas seguintes e requisições entre os participantes. Ressalva mais uma vez, que a coreografia estende a execução lógica de cada um dos serviços (como as condições, seqüências, paralelismo e exceção de execuções), preocupando-se com a visão macro das interações entre serviços web distintos.

5.1 DEFINIÇÃO DA EXECUÇÃO DA COREOGRAFIA

A coreografia define as regras e interações da colaboração entre duas ou mais entidades de negócios (KAVANTZAS, 2005). A coreografia é descrita da perspectiva de todos os participantes (visão comum), e define o comportamento entre participantes de uma colaboração. Estende-se visão comum como a definição do estado do compartilhamento das interações entre entidades de negócios, e isto pode ser usado para determinar necessidades especiais de implementações por parte de cada entidade individual.

No cenário real de execução de um sistema que seja constituído de diversas entidades compostas de parcerias comerciais ou àquelas que resguardam alguma propriedade intelectual, ou mesmo uma regra importante de um negócio, dificilmente delegarão o controle destes processos, e desta imposição, pode-se afetar os resultados da colaboração de serviços distintos. Coreografia possibilita que as regras de colaboração sejam claramente definidas, capacitando cada entidade implementar uma ação, por exemplo, mantendo a visão geral do processo como um todo.

A Figura 21 apresenta um exemplo da execução de um processo coreografado.

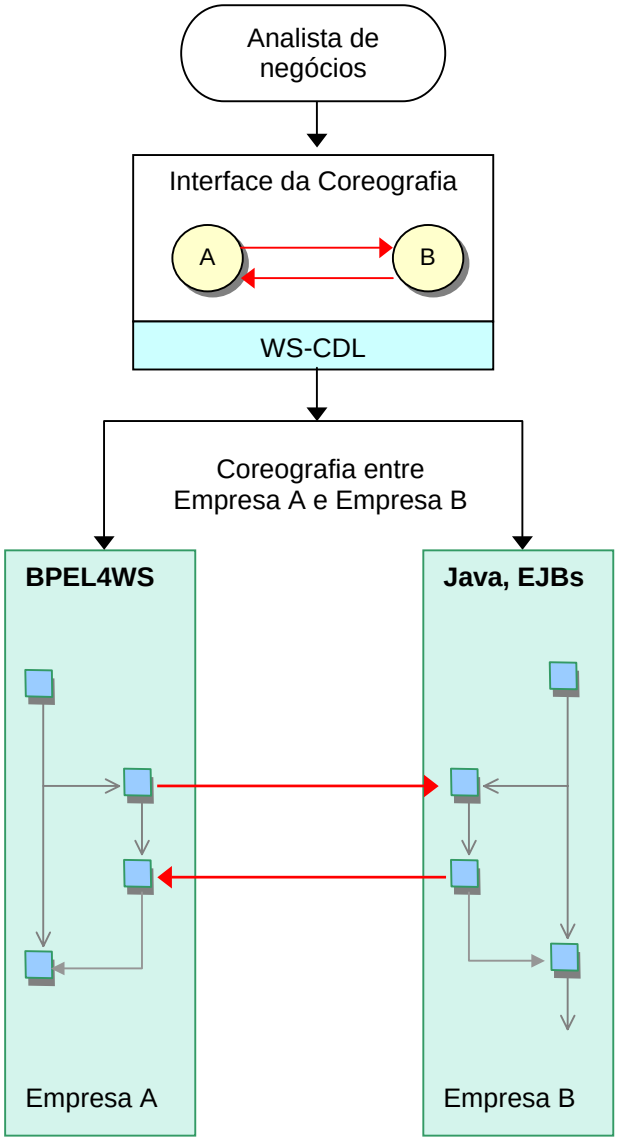

FIGURA 21 – COREOGRAFIA EM AÇÃO (KAVANTZAS, 2005)

Na figura acima, duas empresas (A e B) integram seus processos. Da perspectiva do analista de negócio, ou responsável pela colaboração entre os processos, visualiza-se as regras envolvidas na colaboração, criando assim uma representação WS-CDL. Esta representação pode ser utilizada para gerar o fluxo BPEL4WS da empresa A, e ao mesmo tempo determinar as necessidades de implementação da empresa B, em tecnologias diferentes (Java, EJBs). Garante-se neste exemplo o princípio da interoperabilidade.

5.2 OBJETIVOS

Em uma abordagem macro, define-se a WS-CDL como uma linguagem baseada em XML que especifica, de um ponto de vista comum entre entidades (provedoras de serviços) participantes de uma colaboração, as trocas de mensagens que ocorrem e regras que precisam ser satisfeitas, assim como as relações permissíveis. Mais especificamente, os objetivos da WS-CDL são permitir (KAVANTZAS, 2005):

- Usabilidade (*Reusability*): A mesma definição de coreografia é empregada em diferentes contextos, com diferentes softwares;
- Cooperação (*Cooperation*): Coreografia define a seqüência das trocas de mensagens entre duas (ou mais) partes independentes ou descrevem os processos que participam de uma colaboração;
- Multi-colaboração (*Multi-party collaboration*): Coreografias podem ser definidas envolvendo inúmeras partes ou processos;
- Semântica (*Semantics*): Coreografias podem incluir documentações e semânticas de todos os componentes da coreografia;
- Composição (*Composability*): Coreografia existentes podem ser combinadas com novas coreografia, e que podem ser reusadas em diferentes contextos;
- Modularidade (*Modularity*): Coreografia pode ser definida usando um “facilitador de inclusão” permitindo criar uma nova coreografia através de partes contidas em outras diferentes coreografias;
- Definir o percurso da colaboração (*Information Driven Collaboration*): Coreografias descrevem como as partes fazem o progresso dentro de uma colaboração;

- Alinhamento das informações (*Information Alignment*): Coreografias permitem, as partes envolvidas, comunicar e sincronizar as informações geradas.

- Controle das Exceções (*Exception Handling*): Coreografias permitem controlar as exceções ou condições incomuns geradas na execução da coreografia;

- Transações (*Transactionality*): Os processos ou partes que fazem parte na coreografia podem trabalhar de uma maneira "transacional", com habilidade para coordenar os resultados de colaborações de longa duração, que incluem participantes múltiplos, mantendo a privacidade das regras de negócios de cada parte envolvida.

- Composição de especificações (*Specification Composability*): a WS-CDL complementa outras especificações, principalmente as linguagens que descrevem os processos de negócios, como exemplo a BPEL.

5.3 RELAÇÃO COM AS LINGUAGENS DE DESCRIÇÃO DOS PROCESSOS DE NEGÓCIOS

A WS-CDL não é uma linguagem de descrição de processos de negócios e não depende de uma determinada linguagem de descrição de negócios para implementar uma coreografia. Este aspecto determina a interoperabilidade da execução do processo de coreografia, pois suporta a presença de qualquer parte envolvida, independente de como cada uma foi implementada. Cada parte presente na representação da colaboração definida pela WS-CDL pode ser implementada usando mecanismos completamente diferentes, como exemplo:

- Aplicações com implementações baseadas em linguagens de descrição de processos de negócios (XLANG, WSFL, WSBPEL, BPML, etc);

- Aplicações implementadas por linguagens de programação (Java, .Net, PHP, etc);
- Ou agentes de softwares controlados (intervenção humana).

5.4 MODELO DA LINGUAGEM DE DESCRIÇÃO DA COREOGRAFIA

Nesta subseção será introduzido o modelo da *Web Services Choreography Description Language (*WS-CDL).

5.4.1 DESCRIÇÃO DO MODELO WS-CDL

O modelo WS-CDL consiste das seguintes partes (KAVANTZAS, 2005):

- *Participant Types*, *Role Types* e *Relationship Types*: Em uma coreografia as informações respeitam o fluxo de troca concebido entre os serviços envolvidos. A *Role Type* (Tipos de papéis, funções) enumera os procedimentos de um serviço, classificando-os em ordem da colaboração com outras partes. Um *Relationship Type* (Tipo de relacionamento) identifica as ligações mutuas que precisam ser executadas entre duas partes para o sucesso da execução da colaboração. Um *Participant Type* (Tipo de participante) agrupa simultaneamente as partes relacionadas ao mesmo procedimento que precisam ser implementadas de uma mesma entidade lógica ou organização.
- *Information Types, Variables* e *Tokens*: *Variables* (Variáveis) contêm informações sobre objetos comuns numa colaboração, como uma informação trocada ou a informação gerada por uma função exigida. *Tokens* são utilizados para referenciar partes de uma Variável. *Information Types* define o tipo do dado presente na coreografia.
- *Choreographies*: define as colaborações entre os serviços participantes.

- *Choreography Life-line*: Expressa a progressão de uma colaboração. Inicialmente, a colaboração é estabelecida entre partes, em seguida tarefas são realizadas podendo ser concluídas como normal ou irregular.

- *Choreography Exception Blocks*: Especifica quais interações deverão ocorrer quando uma coreografia retorna um estado irregular.

- *Choreography Finalizer Blocks*: Descreve o modo de fazer interações adicionais específicas que poderão ocorrer para modificar o comportamento de uma coreografia completada com sucesso, como exemplo para confirmar ou retroceder um efeito.

- *Channels* (Canais): Um canal concebe um ponto de colaboração entre partes, especificando onde e como as informações serão trocadas.

- *Work Units*: Determina as regras de consistência que precisam ser completadas na execução de uma tarefa específica dentro de uma coreogarfia.

- *Activities and Ordering Structures*: *Activities* são o mais baixo nível de componentes da coreografia que efetuam a tarefa atual. *Ordering Structures* combinam *Activities* com outros *Ordering Structures* numa estrutura aninhada visando expressar a ordem das condições na qual as informações são trocadas dentro da coreografia.

5.4.2 ESTRUTURA DO DOCUMENTO WS-CDL

Um documento WS-CDL é simplesmente um conjunto de definições. Cada definição é construída no formato de um bloco nomeado que pode ser

referenciado. Há um elemento *package* definido com raiz, e as definições individuais dos tipos das coreografias são elucidadas no interior deste.

Um pacote de coreografia (*Choreography Package)* agrega um conjunto de definições que funcionam como um cabeçalho do documento, apresentando algumas informações básicas além de outras definições que descrevem elementos do processo de colaboração. A Figura 22 apresenta a sintaxe da definição do elemento *package.*

```
<package
   name="ncname"
   author="xsd:string"?
   version="xsd:string"?
   targetNamespace="uri"
   xmlns="http://www.w3.org/2004/12/ws-chor/cdl">
   informationType*
   token*
   tokenLocator*
   roleType*
   relationshipType*
   participantType*
   channelType*
   Choreography-Notation*
</package>
```

FIGURA 22– SINTAXE DO ELEMENTO PACKAGE

Um documento WS-CDL contêm um conjunto de uma ou mais coreografias e um conjunto de uma ou mais definições de tipos de colaboração, por sua vez, um pacote de coreografia contêm:

- Zero ou mais *Information Types*
- Zero ou mais *Tokens* e *Tokens Locators*
- Zero ou mais *Role Types*
- Zero ou mais *Relationship Types*
- Zero ou mais *Participant Types*
- Zero ou mais *Channel Types*

- Zero ou mais *Package-level Choreographies*

Descrevendo o interior da Figura 22, os atributos *name, author* e *version* definem as propriedades de autoria do documento de coreografia. O *targetNamespace* fornece o *namespace* associado com todas as definições contidas neste pacote. Os elementos *informationType, token, tokenLocator, roleType, relationshipType, participantType* e *channelType* podem ser usados como elementos de todas as demais coreografias definidas neste pacote de coreografia.

5.4.3 COLABORAÇÃO DOS SERVIÇOS WEB PARTICIPANTES

A especificação WSDL descreve a funcionalidade de um serviço fornecido por um provedor de serviços. As aplicações emergentes baseadas no ambiente Internet requerem a habilidade de trocar informações em um ambiente de colaboração. Nestes tipos de ambiente um participante solicita serviços fornecidos por outro participante e é ao mesmo tempo um fornecedor dos serviços pedidos de outros participantes, criando assim dependências mútuas.

Um documento WS-CDL descreve como um serviço é capaz de relacionar-se com participantes diferentes e até com ele mesmo. Os *Role Types, Participant Types, Relationship Types* e *Channel Types* definem o acoplamento dos participantes da colaboração (KAVANTZAS, 2005).

5.4.3.1 TIPOS DE PAPÉIS (*ROLE TYPES*)

Um Tipo de papel enumera os procedimentos esperados que sejam desempenhados por um serviço para dispor e participar de uma colaboração com outros serviços. Por exemplo, o papel denominado “Comprador” é associado com a compra de bens ou serviços e o papel “Fornecedor” é dispor esses bens ou serviços mediante um pagamento. A ordem de declaração dos procedimentos indica a ordem correta para estabelecer a colaboração. A Figura 23 apresenta a sintaxe do elemento *roleType*:

```
<roleType name="ncname">
   <behavior name="ncname" interface="qname"? />+
</roleType>
```

FIGURA 23 – SINTAXE DO ELEMENTO ROLETYPE.

O atributo *name* é utilizado para distinguir cada elemento *roleType* declarado dentro de uma pacote de coreografia. O elemento *behavior* especifica um subconjunto de procedimentos. Um *roleType* precisa conter um ou mais elementos *behavior.* O atributo *name* dentro do elemento *behavior* é utilizado para distinguir cada *behavior* dentro daquele elemento *roleType.*

O elemento *behavior* define um atributo opcional declarado como *interface,* que identifica um tipo de interface WSDL. Um *behavior* sem um elemento *interface* descreve um tipo de papel que não requer uma interface específica de serviço web.

5.4.3.2 TIPOS DE RELACIONAMENTO (*RELATIONSHIP TYPE*)

Um tipo de relacionamento identifica os tipos de papéis e procedimentos (*behaviors)* que devem ser executados para o sucesso de uma atividade colaborativa. Como exemplo, tipos de relacionamentos entre um comprador e um fornecedor podem ser:

- Um tipo de relacionamento denominado “Compra”, para a obtenção inicial dos bens ou dos serviços, e;
- Um tipo de relacionamento denominado “Gerenciamento do consumidor” para permitir que o fornecedor disponibilize o serviço satisfazendo alguns princípios básicos, como qualidade, prazo de entrega, forma de entrega, etc.

Embora tipos de relacionamentos estejam sempre entre dois tipos de papéis, coreografia envolvendo mais que dois papéis são possíveis, como exemplo a inserção de uma função “entrega” no relacionamento definido no exemplo anterior. A Figura 24 indica a sintaxe do elemento *relationship type.*

```
<relationshipType name="ncname">
   <role type="qname" behavior="list of ncname"? />
   <role type="qname" behavior="list of ncname"? />
</relationshipType>
```

FIGURA 24 – SINTAXE DO ELEMENTO RELATIONSHIPTYPE.

Um elemento *relationshipType* deve ter exatamente dois tipos de papéis definidos. Cada tipo de papel é especificado através do atributo *type* dentro do elemento *role.* Há também o atributo opcional *behavior*, que identifica o procedimento declarado no tipo do papel necessário para o estabelecimento da relação. Se este atributo for suprido, então todos os procedimentos do tipo de papel são identificados como necessário.

5.4.3.3 TIPO DE PARTICIPANTE (*PARTICIPANT TYPE)*

Um tipo de participante identifica um conjunto de tipos de papéis que devem ser executados pela mesma entidade ou organização lógica. Sua finalidade é agrupar partes de procedimentos que devem ser implementados por uma mesma entidade ou organização lógica. A Figura 25 apresenta a sintaxe do elemento *participantType*.

```
<participantType name="ncname">
   <role type="qname" />+
</participantType>
```

FIGURA 25 – SINTAXE DO ELEMENTO PARTICIPANTTYPE.

No interior do elemento *participantType*, um ou mais elementos *role* identificam os tipos de papéis que devem ser executados por este tipo de participante. Cada tipo de papel é especificado pelo atributo *type*. Um tipo específico de papel não deve ser declarado em mais que um elemento *participantType.*

Um exemplo é dado na Figura 26 onde o tipo do papel "SellerForBuyer" que pertence ao tipo do relacionamento "Buyer-Seller" é executado pelo tipo de participante "Broker" (*Corretor)* que executa também o tipo do papel

"SellerForShipper" que pertence a um tipo do relacionamento do "Seller-Shipper".

```
<roleType name="Buyer">
   . . .
</roleType>
<roleType name="SellerForBuyer">
    <behavior name="sellerForBuyer"
interface="rns:sellerForBuyerPT"/>
</roleType>
<roleType name="SellerForShipper">
    <behavior name="sellerForShipper"
interface="rns:sellerForShipperPT"/>
</roleType>
<roleType name="Shipper">
   . . .
</roleType>
<relationshipType name="Buyer-Seller">
    <role type="tns:Buyer" />
    <role type="tns:SellerForBuyer" />
</relationshipType>
<relationshipType name="Seller-Shipper">
    <role type="tns:SellerForShipper" />
    <role type="tns:Shipper" />
</relationshipType>
<participantType name="Broker">
   <role type="tns:SellerForBuyer" />
   <role type="tns:SellerForShipper" />
</participantType>
```

FIGURA 26 – EXEMPLO DA DECLARAÇÃO DO ELEMENTO PARTICIPANTTYPE

5.4.3.4 TIPO DE CANAL (*CHANNEL TYPE*)

Um canal constrói um ponto de colaboração entre partes, especificando onde e como as informações são trocadas entre essas partes envolvidas em um processo de colaboração. Adicionalmente, as informações dos canais podem ser passadas entre os participantes envolvidos na troca de informações.

Um tipo de canal precisa descrever o tipo de papel e o tipo de referência do canal que está sendo construído, determinando assim onde e como enviar ou receber informações. A sintaxe do *channelType* pode ser observada na Figura 27.

```
<channelType  name="ncname"
              usage="once"|"unlimited"?
              action="request-respond"|"request"|"respond"? >
   <passing  channel="qname"
             action="request-respond"|"request"|"respond"?
             new="true"|"false"? />*
   <role  type="qname"  behavior="ncname"? />
   <reference>
      <token name="qname"/>
   </reference>
   <identity>
      <token name="qname"/>+
   </identity>?
</channelType>
```

FIGURA 27 – SINTAXE DO ELEMENTO CHANNELTYPE

O atributo opcional *usage* é usado para restringir o número de vezes que um determinado *channelType* pode ser utilizado como um canal. O atributo opcional *action* é usuado para definir o tipo da troca de dados que pode ser realizado quando faz-se o uso de um determinado *channelType.* O elemento *role* é usado para identificar o tipo de papel de um participante envolvido no canal estabelecido. O elemento *reference* é usado para descrever a referência do participante, determinando onde e como enviar ou receber informações. O elemento opcional *identity* pode ser usado para identificar a ligação lógica entre as partes declaradas no canal.

5.4.3.5 TIPOS DE INFORMAÇÕES (INFORMATIONTYPES)

Este elemento define os tipos das informações usadas dentro de uma coreografia, evitando a declaração dos tipos de dados diretamente no documento que especifica a coreografia, e sim referenciando aos tipos declarados nos documentos WSDL ou XML *Schema*. A Figura 28 apresenta a sintaxe do elemento *informationType*.

```
<informationType name="ncname"
                 type="qname"?|element="qname"?
                 exceptionType="true"|"false"? />
```

FIGURA 28 – SINTAXE DO ELEMENTO INFORMATIONTYPE

Os atributos opcionais *type* e *element* descrevem o tipo de uma informação usada dentro da coreografia como um tipo de mensagem WSDL, um tipo XML *Schema* ou como um elemento XML *Schema*.

5.4.3.6 VARIÁVEIS (VARIABLES)

Variáveis capturam informações sobre objetos em uma coreografia. A construção de *variableDefinitions* é usada para definir uma ou mais variáveis dentro da coreografia. Sua sintaxe pode ser visualizada na Figura 29.

```
<variableDefinitions>
   <variable   name="ncname"
       informationType="qname"?|channelType="qname"?
       mutable="true|false"?
       free="true|false"?
       silent="true|false"?
       roleTypes="list of qname"? />+
</variableDefinitions>
```

FIGURA 29 – SINTAXE DO BLOCO VARIABLEDEFINITIONS

A variável definida usando o atributo *informationType* especifica variáveis que capturam informações provindas das trocas realizadas entre os serviços participantes de uma coreografia. A variável definida usando o atributo *channelType* especifica variáveis que capturam informações referente a um canal. Os atributos *informationType* e *channelType* são mutuamente exclusivos.

O atributo opcional *mutable*, quando definido como *false*, especifica que a informação de uma variável não pode ser alterada uma vez inicializada. O valor padrão deste atributo é *true.*

O atributo opcional *free*, quando definido como *true*, especifica que uma variável definida dentro de uma determinada coreografia é também compartilhada com outras coreografias. Quando o atributo *free* é definido como *false* determina que a variável é válida somente dentro da coreografia no qual a mesma é declarada. O valor padrão para o atributo *free* é *false.*

5.4.3.7 TOKENS

Um *token* provê um mecanismo para definir um pseudônimo para um *informationType. Tokens* diferem de variáveis, sendo que variáveis contêm valores enquanto *tokens* contêm informações que definem uma parte de um dado que é relevante na coreografia. A sintaxe do elemento *token* é exibido na Figura 30.

```
<token  name="ncname"  informationType="qname" />
```

FIGURA 30 – SINTAXE DO ELEMENTO TOKEN

Para localizar um determinado *token* pode ser definido um elemento *tokenLocator*, que indica um mecanismo de consulta para localizar um *token* dentro de uma mensagem WSDL ou XML *Schema.* A Figura 31 apresenta a sintaxe do *tokenLocator.*

```
<tokenLocator  tokenName="ncname"
               informationType="qname"
               part="ncname"?
               query="XPath-expression" />
```

FIGURA 31 – SINTAXE DO ELEMENTO TOKENLOCATOR

5.4.3.8 COREOGRAFIAS (CHOREOGRAPHIES)

Uma coreografia define um conjunto de regras que coordenam a ordem das trocas de mensagens e os recursos dos papéis da colaboração envolvendo duas ou mais partes interagentes.

O pacote de coreografia contém exatamente uma coreografia identificada como inicial (*root*). Esta coreografia é ativada por padrão. Uma coreografia não definida como inicial é ativada quando executada (*performed*).

Uma coreografia deve conter ao menos um tipo de relacionamento, descrevendo os tipos de papéis requeridos nesta coreografia. Um ou mais tipos de relacionamentos podem ser definidos dentro de uma coreografia, modelando assim colaborações com múltiplas partes envolvidas.

A coreografia pode conter uma ou mais referências de outras coreografias que podem ser executadas dentro desta determinada coreografia. A sintaxe da definição de uma coreografia pode ser visualizada na Figura 32.

```
<choreography  name="ncname"
      root="true"|"false"? >
   <relationship  type="qname" />+
   variableDefinitions?
   Choreography-Notation*
      Activity-Notation*
   <exceptionBlock  name="ncname">
      WorkUnit-Notation+
   </exceptionBlock>?
   <finalizerBlock  name="ncname">
      WorkUnit-Notation*
   </finalizerBlock>*
</choreography>
```

FIGURA 32 – SINTAXE DO ELEMENTO CHOREOGRAPHY (COREOGRAFIA)

O elemento *relationship* declarado no interior da coreografia enumera os relacionamentos estabelecidos dentro da coreografia. O elemento opcional *variableDefinitions* define as variáveis da coreografia. O atributo opcional *root* marca uma coreografia como inicial entre todas as coreografias definidas no pacote (*package*). A definição opcional *choreography-notation* especificada na sintaxe do elemento *choreography,* define as demais coreografias que serão executas dentro de uma determinada coreografia, criando assim a composição de coreografias.

5.4.3.9 INTERAÇÕES (*INTERACTING*) E COMPOSIÇÕES (*PERFORM*)

Uma interação é o bloco básico de construção de uma coreografia que define as trocas de informações entre os participantes da colaboração. A interação constitui a base de composição da coreografia, onde múltiplas interações são combinadas para formar uma coreografia, que por sua vez pode ser usada em diferentes contextos de negócios.

Uma interação é iniciada no momento que um dos serviços participantes enviam uma mensagem, através de um canal estabelecido, para outro serviço que participa da interação, que por sua vez recebe a mensagem enviada. Se a mensagem inicial é uma requisição, então o serviço que recebe a mensagem pode opcionalmente gerar uma resposta.

A sintaxe básica do elemento *interaction* é apresentada na Figura 33.

```
<interaction  name="ncname"
              channelVariable="qname"
              operation="ncname"
              align="true"|"false"?
              initiate="true"|"false"? >
   <participate  relationshipType="qname"
                  fromRole="qname" toRole="qname" />
   <exchange  name="ncname"
              informationType="qname"?|channelType="qname"?
              action="request"|"respond">
     <send      variable="XPath-expression"?
                recordReference="list of ncname"?
                causeException="true"|"false"? />
     <receive   variable="XPath-expression"?
                recordReference="list of ncname"?
                causeException="true"|"false"? />
   </exchange>*
</interaction>
```

FIGURA 33 – SINTAXE BÁSICA DO ELEMENTO INTERACTION

A composição de coreografias é realizada através das declarações dos elementos *perform.* Este elemento permite a uma coreografia especificar que outra coreografia deve ser executada num determinado momento. A Figura 34 apresenta a sintaxe básica do elemento *perform*.

```
<perform  choreographyName="qname">
   <bind  name="ncname">
     <this variable="XPath-expression" role="qname"/>
     <free variable="XPath-expression" role="qname"/>
   </bind>*
   Choreography-Notation?
</perform>
```

FIGURA 34 – SINTAXE BÁSICA DO ELEMENTO PERFORM

6 PLATAFORMA DE EXECUÇÃO DE SERVIÇOS WEB COREOGRAFADOS

Neste capítulo serão definidas as estruturas da plataforma proposta, assim como todas as suas características e funcionalidades existentes. Inicialmente apresenta-se a motivação principal para construção da plataforma, destacando a necessidade de execução da coreografia de serviços web e as oportunidades e facilidades proporcionadas mediante esta execução apoiada em um ambiente estruturado e previamente definido. Posteriormente, detalha-se os recursos e funcionalidades previstos no ambiente de processamento de serviços web. Concluindo esta seção, expõem-se os detalhes relativos a implementação de um protótipo da plataforma proposta.

6.1 APLICAÇÃO DA EXECUÇÃO DE SERVIÇOS WEB

A execução de serviços web está condicionada a procedimentos de chamada dos aplicativos que representam cada serviço. Quando o ambiente de execução dos serviços é simples (envolvendo um único serviço específico ou um número baixo de serviços participantes) a tarefa de chamar pelos serviços pode ser feita manualmente (por intermédio de comandos de linguagem de desenvolvimento de um serviço participante) sem dificultar ou prejudicar o objetivo do responsável pelo desenvolvimento do sistema.

No entanto, quando há um ambiente complexo de execução de serviços web, caracterizando-se pelo número elevado de serviços participantes, além da possibilidade de indefinição dos serviços necessários, executar os serviços manualmente pode acarretar numa dedicação de tempo muito alto por parte do desenvolvedor, podendo influenciar no tempo total destinado ao projeto. Entende-se indefinição dos serviços necessários como a necessidade de chamar por um serviço específico (disponibilidade de assento em um vôo para uma localidade específica, por exemplo) sem conhecer o provedor do serviço, ou ainda a possibilidade de concorrência entre entidades distintas (existência de várias companhias aéreas, por exemplo).

Além do fator tempo, há outros inconvenientes que podem prejudicam a execução de serviços web, se isto necessitar ser feito manualmente pelo desenvolvedor. Entre estes inconvenientes, destacam-se: a necessidade de envolver no desenvolvimento de um serviço específico as regras de interação entre todos os serviços participantes; inexistência de rotinas de controle de exceções nas chamadas de cada serviço; falta de conhecimento dos resultados do processamento de cada serviço; alto custo na manutenção dos serviços e dificuldade na alteração (inclusão ou exclusão) de serviços em um ambiente já em execução.

Todos os fatores supramencionados possibilitam exatamente o uso da coreografia dos serviços, tornando possível a dedicação exclusiva do desenvolvedor para as funcionalidades e regras de negócios de cada serviço, sem se preocupar com a execução dos serviços envolvidos em um processo de colaboração.

Neste contexto, para aproveitar efetivamente os benefícios da coreografia é necessário fazer uso de um ambiente que torne transparente ao desenvolvedor as funcionalidades já lançadas nesta subseção, ressaltando:

- Capacidade de analisar um ambiente de colaboração de serviços web;
- Capacidade de identificar e controlar as interações entre cada serviço participante de um ambiente de colaboração mútua;
- Capacidade de localizar os serviços necessários para execução de um ambiente de colaboração;
- Capacidade de executar os serviços, gerenciando os resultados e as exceções observadas durante a execução;
- Capacidade de reportar ao usuário do ambiente de colaboração – normalmente o desenvolvedor do(s) serviço(s) – os resultados do processamento dos serviços no ambiente de colaboração.

Naturalmente, cada desenvolvedor pode implementar as funcionalidades apresentadas neste trabalho, no entanto elevaria substancialmente o tempo de desenvolvimento de qualquer projeto de coreografia de serviços web, além de influenciar negativamente os objetivos de cada serviço devido a necessidade de desenvolver toda esta parte estrutural do ambiente de execução. Neste sentido, propõe-se com base nestas funcionalidades, a construção da plataforma apresentada a partir da próxima subseção.

6.2 PROPOSTA DE UMA PLATAFORMA PARA A EXECUÇÃO DE SERVIÇOS WEB COREOGRAFADOS

A plataforma proposta é composta por entidades capazes de disponibilizar ao desenvolvedor de serviços voltados para um ambiente de colaboração, a transparência na execução e gerenciamento dos resultados obtidos após a execução de cada serviço participante. A Figura 35 apresenta a macro-definição da referida plataforma.

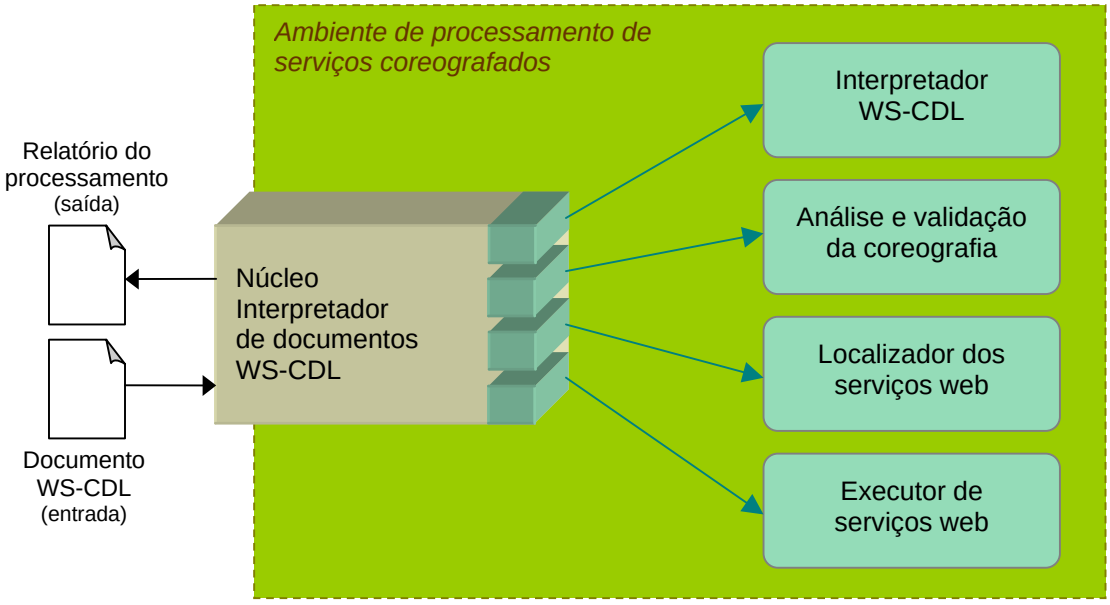

FIGURA 35 – MACRO DEFINIÇÃO DA PLATAFORMA PROPOSTA

6.2.1 ENTIDADES DA PLATAFORMA DE EXECUÇÃO DE SERVIÇOS

A capacidade da plataforma de dispor o ambiente de processamento de serviços coreografados, transparente ao desenvolvedor, é possível através do funcionamento e relacionamento de cinco entidades essenciais, sendo elas:

- Núcleo Interpretador de documentos WS-CDL: capaz de processar a entrada do sistema e distribuir as atividades entre as demais entidades disponíveis. Esta entidade realiza a função de cérebro da plataforma, sendo responsável pela identificação inicial do documento recebido do usuário e gerenciamento de todas as demais entidades.

- Interpretador WS-CDL: responsável pela leitura e tradução completa do documento WS-CDL disponível no ambiente da plataforma.

- Análise e validação da coreografia: interpretação da tradução realizada pela entidade definida acima. Esta entidade será responsável pela identificação das portas de comunicação entre cada uma das entidades participantes.

- Localizador de serviços: entidade responsável pela verificação da disponibilidade dos serviços participantes da colaboração.

- Executor de serviços: motor da plataforma. Responsável pela execução de cada um dos serviços, sendo capaz de gerenciar os resultados e controlar as exceções ou falhas que eventualmente podem ocorrer.

6.2.1.1 NÚCLEO INTERPRETADOR

Esta entidade é responsável pelo gerenciamento do relacionamento entre as entidades Interpretador, Análise e Validação, Localizador e Executor de serviços, controlando assim o fluxo de dados durante a execução da plataforma e o escalamento do uso de cada entidade. Embora o núcleo interpretador delegará as atribuições das atividades principais do ambiente para as outras entidades (como exemplo a interpretação completa do documento e a

execução dos serviços), ele por sua vez será capaz de realizar algumas atividades importantes no ambiente, sendo elas:

- Interface com o ambiente externo realizando a recepção do documento WS-CDL.
- Centralizar as chamadas às outras entidades do ambiente. Ao assumir esta responsabilidade, o núcleo torna os processos internos transparente ao usuário da plataforma.
- Elaboração e disponibilização do relatório de processamento da plataforma, notificando o usuário das principais ocorrências durante a execução do ambiente.

O funcionamento do núcleo se dará somente através da identificação e leitura de documentos WS-CDL, pois se objetiva disponibilizar a capacidade de coreografar os serviços participantes de uma colaboração, e conforme discutido nos capítulos 3 e 5, esta linguagem abrange todas as funcionalidades necessárias para implementação da coreografia. No entanto o modelo da plataforma viabiliza a adição de novos módulos, capazes de interpretar outras linguagens e especificações. Mais detalhes desta possibilidade de expansão da plataforma será apresentado no capítulo 8.

6.2.1.2 INTERPRETADOR WS-CDL

Esta entidade é responsável pela verificação da integridade do documento WS-CDL. O interpretador WS-CDL se preocupará em analisar todo o documento em busca de possíveis falhas em sua estrutura que podem prejudicar, ou até mesmo parar, a execução dos serviços coreografados. Suas principais funcionalidades são:

- Validar as declarações das *tags* do documento WS-CDL
- Identificar todos os elementos contidos na coreografia.

- Identificar todas as mensagens, portas e operações contidas na coreografia.

6.2.1.3 ANÁLISE E VALIDAÇÃO DA COREOGRAFIA

Após a identificação dos elementos, mensagens, portas e operações da coreografia a entidade de análise e validação será responsável em definir ao núcleo interpretador as relações entre os serviços participantes da coreografia. Esta definição será a base para a futura localização e execução dos serviços web. Especificamente, esta entidade será responsável em:

- Delinear as ligações entre todos os serviços web participantes da coreografia.

- Informar ao núcleo interpretador a ordem de execução dos serviços, assim como as portas dos serviços e estrutura das mensagens.

Conceituando de outra forma, esta entidade será responsável em definir o modelo abstrato da coreografia especificada no documento WS-CDL, servindo inclusive de base para elaboração do relatório de processamento.

6.2.1.4 LOCALIZADOR DE SERVIÇOS

O localizador de serviços exerce uma função simples, porém importante para otimizar os recursos da plataforma. Depois de identificado todos os serviços participantes na coreografia, esta entidade busca a disponibilidade física de cada um dos serviços. Esta verificação de disponibilidade poderia ser exercida também pelo executor de serviços, no entanto a eventualidade de um serviço, disposto no final da ordem de execução do ambiente da plataforma, se tornar indisponível, todos os recursos dispendidos na execução dos serviços anteriores serão inúteis.

As tarefas desta entidade são:

- Verificar a disponibilidade física de cada um dos serviços participantes do ambiente de colaboração

- Consultar a possibilidade de aplicar outro serviço web na medida que surgirem serviços indisponíveis.

6.2.1.5 EXECUTOR DE SERVIÇOS

O executor de serviços realiza as chamadas dos serviços web participantes da coreografia. São funções desta entidade:

- Executar cada um dos serviços participantes do ambiente de colaboração.

- Carregar as mensagens de entrada e saída do processamento entre os serviços coreografados.

- Controlar as exceções e falhas na execução dos serviços, notificando o usuário da plataforma.

- Compor o relatório de processamento com os resultados de processamento dos serviços.

6.2.2 ENTRADA E SAÍDA DA PLATAFORMA

A alimentação da plataforma é realizada exclusivamente por um documento WS-CDL. Criar uma linguagem especifica desta plataforma inviabilizaria a difusão do uso desta ferramenta, limitando-se seu destino apenas aos usuários que dominassem a linguagem em questão. Por outro lado, a linguagem WS-CDL responde as necessidades da plataforma. Alguns comentários no formato de contribuição à linguagem WS-CDL serão expostas no capítulo 8, após a avaliação da plataforma proposta.

Uma característica potencial da plataforma proposta corresponde a saída do núcleo interpretador por um documento denominado Relatório de Processamento. Como a plataforma tornará transparente ao seu usuário o

processamento dos serviços coreografados, este documento informará todas as ocorrência geradas no ambiente de execução, servindo principalmente como depurador da colaboração entre os serviços.

O relatório de processamento informará ao usuário os seguintes dados:

- Modelo abstrato da coreografia contida no documento WS-CDL, apresentando graficamente a disposição dos serviços web e o fluxo de dados entre eles.
- Situação após a execução de cada um dos serviços
- Valores das mensagens indicadas no documento WS-CDL após a execução dos serviços
- Erros ou falhas geradas ao executar os serviços web.
- Resultado final do processamento dos serviços no ambiente.

6.3 IMPLEMENTAÇÃO DE UM PROTÓTIPO DA PLATAFORMA PROPOSTA

A questão inicial relativa a implementação é sobre o ambiente que será utilizado para desenvolver todas as funcionalidades previstas na plataforma proposta. Pelo contexto geral do trabalho, é necessário que a plataforma seja disponibilizada em um ambiente que estimule e facilite sua execução, ou seja, não é conveniente se empregar tecnologias que por ventura poderiam limitar a difusão da plataforma, podendo assim inviabilizar a utilização por parte de todos aqueles envolvidos no processo de colaboração a ser processado pela plataforma.

Nesse sentido idealiza-se a disponibilidade da plataforma em um ambiente capaz de ser acessado por todos aqueles interessados no processo de colaboração em questão, e certamente um ambiente capaz de dispor todas essas características é a Internet. Observando todas essas qualidades, surge

como ideal a utilização das linguagens Java, Asp.net e PHP, sendo esta última a escolhida devido a alta familiaridade do autor deste trabalho.

6.3.1 CARACTERIZAÇÃO DO AMBIENTE DA PLATAFORMA

Toda a plataforma foi desenvolvida empregando-se a linguagem PHP, sendo esta linguagem mundialmente utilizada na construção de soluções capazes de serem disponibilizados na Internet. Esta linguagem é possível de ser executada em diversos sistemas operacionais, como exemplo o Windows, o Unix, o Linux o OpenBSD, o Solaris, entre outros (CHOI, 2001). Esta característica se agrega a plataforma que está sendo proposta neste trabalho, estendendo a independência do sistema operacional para funcionamento. Outro ponto interessante é o fato da linguagem PHP ser de código aberto e livre. Para executar os *scripts* PHP é preciso também ter instalado um servidor web, como o IIS ou o Apache.

Dessa forma, para o perfeito funcionamento da plataforma é necessário configurar um ambiente com um servidor web capaz de chamar pelos *scripts* PHP e instalar os recursos da linguagem PHP. Os procedimentos detalhados de instalações desses serviços podem ser encontrados nas respectivas páginas na Internet destinadas a disponibilização dessas ferramentas.

Em especial, é necessário configurar o PHP para interagir com as seguintes bibliotecas:

- php_gd2.dll: Utilizada para geração de imagens;
- php_soap.dll: Utilizada para executar serviços web.

Todas as bibliotecas citadas acima podem ser encontradas na página da Internet relativo a linguagem PHP.

6.3.2 FUNCIONALIDADES DA PLATAFORMA PROPOSTA

A plataforma está estruturada sobre o Núcleo Interpretador de documentos WS-CDL, sendo que este núcleo é o responsável pela recepção do documento

WS-CDL e seu direcionamento para todas as rotinas da plataforma. O núcleo também tem o papel de gerar a interface da plataforma para seu usuário final.

Inicialmente o núcleo disponibiliza ao usuário um formulário para indicação do local físico onde se encontra o documento WS-CDL que será processado. Posteriormente ao recebimento do documento o núcleo encaminha-o para a realização da interpretação (implementado pelo *script* interpretador.php). Após a realização da interpretação, e durante todas as demais rotinas da plataforma, o núcleo disponibiliza na parte superior do navegador botões gráficos para que o usuário controle o fluxo das chamadas aos procedimentos da plataforma.

A Figura 36 apresenta a tela inicial da plataforma gerada pelo núcleo interpretador.

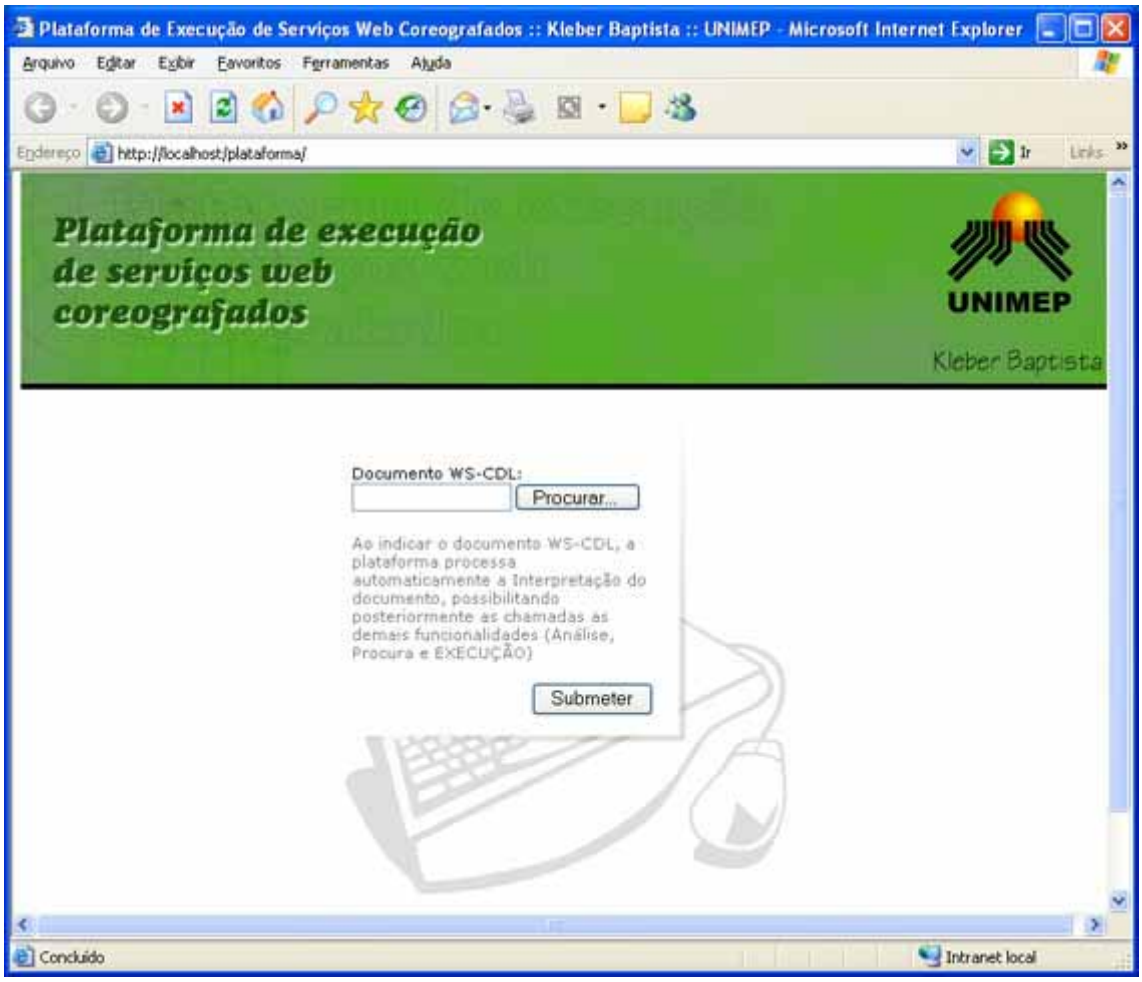

FIGURA 36 – RECEPÇÃO DO DOCUMENTO WS-CDL

Resumidamente, o núcleo recebe o documento WS-CDL e possibilita ao usuário navegar entre os processos da plataforma. Após a execução dos serviços, o núcleo também disponibiliza na parte superior do navegador o relatório do processamento da plataforma, no modelo de texto não formatado.

6.3.2.1 INTERPRETADOR WS-CDL (INTERPRETADOR.PHP)

O interpretador WS-CDL verifica a integridade do arquivo encaminhado pelo núcleo da plataforma, e havendo alguma inconsistência notifica ao usuário e suspende a execução da plataforma. Na atual versão, o interpretador analisa apenas a estrutura das declarações xml do documento WS-CDL, não verificando a sintaxe dos elementos, assim possíveis inconsistências na declaração dos elementos da linguagem WS-CDL não são percebidos. Esta limitação, apesar de não afetar os objetivos deste trabalho, deverá ser corrigida futuramente.

Como resposta desta verificação, o interpretador apresenta ao usuário uma interface gráfica contendo todos os elementos WS-CDL encontrados no documento. O Anexo A detalha o código fonte do interpretador.php.

Os elementos são divididos em 8 grupos específicos: *Channel Types, Choreographies, Information Types, Participant Types, Relationship Types, Role Types, Token Locators* e *Tokens.*

Pela exposição dos elementos WS-CDL contidos no documento, o usuário da coreografia pode aproveitar para interpretar o conteúdo do documento, ou seja, além da necessidade da plataforma em identificar todos os elementos WS-CDL para posterior processamento dos serviços, os usuários podem utilizar o interpretador para estudar e entender um documento WS-CDL através da interface gráfica gerada, ao invés de praticar a leitura do documento em um editor de texto qualquer e sem recursos gráficos.

Além de exibir os elementos contidos nos grupos definidos pelo interpretador, a interface gráfica possibilita navegar entre todos os atributos dos elementos, assim como seus sub-elementos e seus respectivos atributos.

Resumindo, o interpretador WS-CDL identifica todos os elementos do documento WS-CDL para possibilitar a execução dos demais procedimentos da plataforma. Após o processo de interpretação do documento ser realizado com sucesso, o núcleo interpretador habilita o acesso a análise e validação da coreografia.

6.3.2.2 ANÁLISE E VALIDAÇÃO DA COREOGRAFIA (COREOGRAFIA.PHP)

O interpretador do documento (Anexo B) identifica todos os elementos do documento WS-CDL entre eles os *choreography*, no entanto apenas pela identificação dos elementos a plataforma não consegue entender como deve ser realizada a execução dos serviços, principalmente a ordem de chamada pelos serviços web e as possíveis relações entre todos os elementos *choreography.*

A análise e validação da coreografia identifica qual a coreografia inicial do documento WS-CDL e ordena todas as demais coreografias na seqüência correta de serem executadas. Neste momento também são identificados todas as operações que devem ser realizadas por cada coreografia, assim como suas respectivas ordens de execução.

Pode-se afirmar que esta funcionalidade implementa a inteligência da plataforma proposta, definindo o modo como as demais funcionalidades da plataforma devem ser executadas.

Além de analisar as coreografia contidas no documento WS-CDL, há também a validação da coreografia. A validação verifica as relações de todas as declarações de coreografia do documento e suas determinadas referências. Como exemplo, pode-se referenciar uma determinada coreografia dentro da coreografia inicial, sendo que não há a declaração dessa segunda coreografia no documento WS-CDL. Casos como esses, a plataforma emite um sinal de alerta identificando a coreografia que não foi declarada e bloqueia a execução da próxima funcionalidade da plataforma, o localizador dos serviços web.

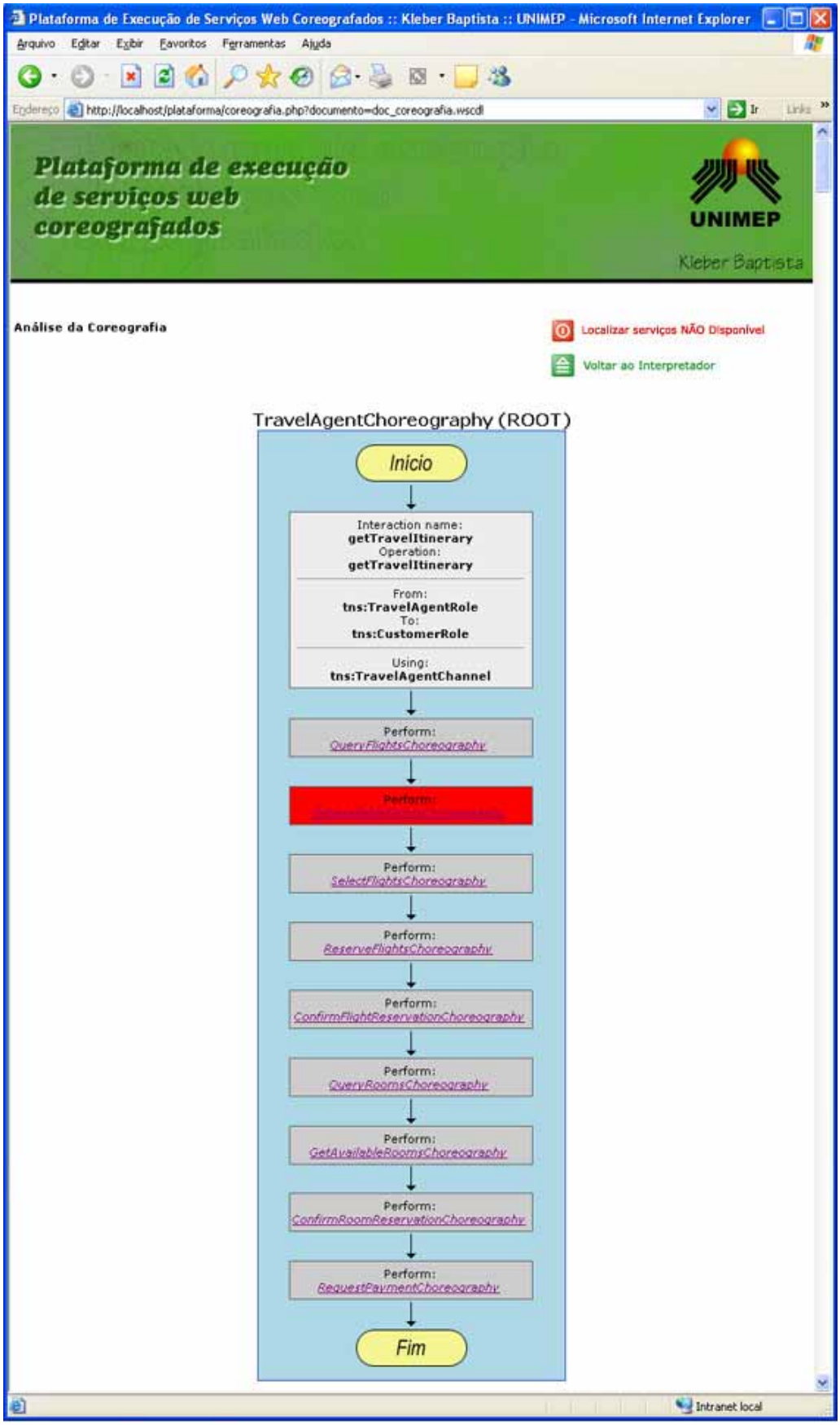

FIGURA 37 – FUNCIONAMENTO DA ANÁLISE E VALIDAÇÃO DA COREOGRAFIA

Assim como acontece com o interpretador WS-CDL, a análise e validação da coreografia apresenta como resultado um esquema gráfico simbolizando as declarações de todas as coreografias e suas relações, possibilitando reforçar a compreensão do processo de coreografia por parte do usuário da plataforma.

A Figura 37 mostra o destaque gerado pela plataforma para identificar uma coreografia referenciada e não declarada (fundo com o preenchimento na cor vermelha) e o bloqueio imposto pelo núcleo na continuidade de execução da plataforma (botão localizar serviços não disponível).

6.3.2.3 LOCALIZADOR DOS SERVIÇOS WEB (LOCALIZADOR.PHP)

Após a interpretação do documento e a análise e validação da coreografia, a plataforma têm conhecimento de todos os serviços participantes da coreografia, assim como a maneira como devem ser estabelecidas as relações entre todos.

Nesse ponto a plataforma está apta a chamar pelos serviços e executar as operações previstas na coreografia, mas antes da execução acontecer o localizador de serviços verifica a disponibilidade de cada um dos serviços envolvidos. Essa funcionalidade poderia ser realizada no momento da execução dos serviços, no entanto caso um determinado serviço esteja indisponível, as execuções das operações realizadas antes desse serviço em questão seriam descartadas, ocasionando em um processamento inútil ou ainda na necessidade de reverter algumas das operações já realizadas.

Dessa forma, o localizador de serviços assessora o executor de serviços no sentido de permitir somente as execuções das coreografia em que todos os serviços encontram-se disponíveis.

Indiretamente o localizador de serviços auxília o usuário informando graficamente como os serviços estão relacionados, melhorando ainda mais a análise da coreografia por parte de todos os envolvidos. A plataforma gera uma imagem no formato PNG do resultado da localização dos serviços. Essa

imagem pode ser utilizada para representar a disposição dos serviços na coreografia.

Para facilitar a análise pelo usuário dos resultados da localização dos serviços, a plataforma representa os serviços em três cores distintas: preto, vermelho e verde. Os serviços representados na cor verde significam que foram localizados com sucesso, os representados na cor vermelha significam que não foram localizados e aqueles representados na cor preta significam que a coreografia não prevê a disponibilidade do serviço. Quando qualquer um dos serviços não for localizado a plataforma impede a execução dos serviços.

A Figura 38 destaca as representações de serviços não localizados e a negativa da execução dos serviços. O código fonte do localizador.php está detalhado no Anexo C.

FIGURA 38 – LOCALIZADOR DE SERVIÇOS

6.3.2.4 EXECUTOR DE SERVIÇOS WEB (EXECUTOR.PHP)

O executor de serviços (Anexo D) analisa todas as operações previstas do documento WS-CDL e realiza as chamadas dessas operações aos seus respectivos serviços. Ao analisar cada operação o executor de serviços identifica as variáveis e imediatamente as instâncias. As variáveis podem ser declaradas como locais ou globais, sendo que essa diferenciação é realizada pelo próprio localizador de serviços.

Uma vez declaradas as variáveis o executor de serviços coordena o fluxo de troca dos conteúdos das variáveis entre os serviços participantes da coreografia. Necessariamente, o executor de serviços controla as chamadas dos serviços e o fluxo das variáveis de três formas distintas, sendo elas:

- *request*: serviços que necessitam do recebimento de uma variável, no entanto não produzem retorno.
- *respond*: serviços que retornam uma variável, no entanto não necessitam do recebimento de uma variável.
- *request-respond*: serviços que necessitam do recebimento de uma variável e também retornar uma outra variável.

Todas as ações praticadas pelo executor de serviços assim como os resultados produzidos por essas ações, são registrados em um arquivo de *log*, que têm o papel de notificar ao usuário os efeitos da execução dos serviços coreografados. Esse arquivo é disponibilizado pelo núcleo da plataforma com o nome de relatório de processamento.

Detalhadamente, o executor de serviços realiza as seguintes ações:

- Registra os atributos básicos do documento de coreografia e marca a data e hora de início da execução dos serviços. Os registro desses atributos básicos possibilitam a identificação do documento analisado.
- Registra cada coreografia analisada, iniciando pela coreografia identificada como *root*.

- Instância cada variável especificada na coreografia, identificando o tipo da variável e o modo como esta deve ser tratada durante a execução de todas as coreografias (local ou global). É efetuado também o registro de cada uma das variáveis instanciadas, indicando o tipo de dados e o modo como foi declarada.

- Registra cada uma das operações previstas na coreografia, indicando o nome da operação, os serviços participantes dessa operação e a forma de controle das chamadas dos serviços (*request, respond* ou *request-respond*).

- Registra o envio da variável, exibindo inclusive seu conteúdo. Isso possibilita ao usuário acompanhar os resultados reais da execução dos serviços.

- Realiza a chamada de cada operação declarada no documento WS-CDL. Neste momento o executor de serviços registra o local do serviço e data e hora da chamada da operação. Logo a seguir, há a indicação de sucesso da chamada da operação (mensagem '>>>OK') ou falha da operação (mensagem '---FALHOU!').

- Registra o recebimento da variável processada pela operação, exibindo inclusive seu conteúdo. Mais uma vez isso possibilita ao usuário acompanhar os resultados reais da execução dos serviços.

No próximo capítulo será realizado o teste de todas as funcionalidades da plataforma declaradas nesta seção mediante a aplicação de um estudo de caso específico.

7 UTILIZAÇÃO DA PLATAFORMA PROPOSTA

Este capítulo destina-se em apresentar a utilização da plataforma proposta neste trabalho por intermédio da aplicação de um exemplo que reproduza a coreografia de serviços web. Inicialmente serão indicados diversos cenários nos quais a plataforma poderá ser aplicada, posteriormente será definido o contexto do exemplo formulado para avaliação da plataforma e finalmente detalhado todos os resultados obtidos na execução da plataforma, assim como as principais contribuições geradas pela plataforma. O objetivo principal do exemplo aplicado é certificar-se que a plataforma é capaz de chamar as operações e coordenar o fluxo de dados dos serviços envolvidos.

7.1 EXEMPLOS DE APLICAÇÃO DA PLATAFORMA

A plataforma proposta pode ser aplicada em diversos cenários, desde que esses cenários estimulem a participação e interação de serviços distintos. A partir do envolvimento de dois serviços web independentes, já é possível utilizar-se da plataforma para efetivar a execução desses serviços. A plataforma está preparada para executar qualquer número de serviços envolvidos, não havendo restrição alguma com relação a isso, no entanto é necessário que o documento WS-CDL especifique corretamente o modo como realizar as colaborações.

Exemplos de cenários possíveis de aplicar a plataforma proposta são:

- gerenciamento da cadeia de suprimentos de uma organização qualquer, envolvendo todas as relações possíveis (fornecedores, departamentos, clientes, etc).
- oferta de um serviço qualquer composto por outros produtos ou serviços, provenientes de fornecedores distintos, por exemplo a oferta de um pacote de turismo composto por hotéis, transportes, passeios, jantares, etc.

- comércio eletrônico, diversificando meios de pagamento, forma de entrega, fornecedores, etc.

- aplicações específicas direcionadas em atender um objetivo específico de uma organização, como exemplo uma solicitação de um serviço de metrologia requisitado por uma indústria química específica, tendo que considerar diversos fatores peculiares de sua estrutura organizacional.

7.2 APLICAÇÃO LIVRARIA-CENTRAL DE DISTRIBUIÇÃO

Especificamente, a aplicação escolhida para avaliar a plataforma proposta corresponde a uma entidade denominada Livraria que necessita consultar disponibilidades de produtos (livros) em uma outra entidade, denominada Central de Distribuição. Essa aplicação retrata um cotidiano típico de diversas organizações que praticam a troca de dados entre parceiros ou clientes.

Neste exemplo há colaboração entre dois serviços web criados, sendo eles:

- Livraria: representa uma organização que comercializa livros diversos, tendo a necessidade de procurar em uma central de distribuição a disponibilidade de um determinado livro escolhido por um cliente.

- Central: responde consultas de qualquer potencial parceiro ou cliente. Comumente recebe a consulta especificando um determinado produto e retorna o total de itens disponíveis.

Embora haja envolvimento de somente duas entidades, através da execução desta aplicação é possível avaliar todas as funcionalidades previstas na plataforma, conforme se detalha abaixo:

- Interpretar o documento WS-CDL que representa a coreografia entre esses serviços, exibindo todos os elementos previstos na linguagem e declarados no documento.

- Analisar e validar a coreografia definida entre os serviços Livraria e Central.
- Localizar a disponibilidade dos serviços Livraria e Central.
- Executar as operações previstas na coreografia, coordenando e controlando as trocas de dados necessárias entre os serviços Livraria e Central.
- Processar o relatório de processamento resultado da execução dos serviços Livraria e Central.

Ambos os serviços utilizados nessa aplicação foram desenvolvidos utilizando a linguagem Java. O fato dos serviços serem desenvolvidos em uma linguagem de programação diferente da plataforma proposta certifica a interoperabilidade de execução da plataforma, capacitando-a em processar qualquer tipo de serviço web.

A Figura 39 expõe a implementação do serviço Livraria.

```
import java.util.Random;
import java.lang.Double;
import java.util.Date;

public class Livraria
{
     static String livros[] = { "Java Como Programar",
                          "Fortaleza Digital",
                          "O Monge e o Executivo",
                          "O Doce Veneno do Escorpiao",
                          "A Hora da Verdade",
                          "Estratégia Competitiva" };

     public static String EscolherLivro()
     {
       java.util.Random r = new Random(new Date().getTime());
       int i = r.nextInt(livros.length);
       return livros[i];
     }
}
```

FIGURA 39 – SERVIÇO WEB LIVRARIA (LIVRARIA.JAVA)

O serviço Livraria simula a escolha de um determinado livro para efetivação de uma transação de venda, ficando pendente a disponibilidade do item. Embora no exemplo o livro é escolhido aleatoriamente, recursos mais avançados, como consultas a uma base de dados, poderiam ser implementados sem afetar a execução na plataforma proposta, lembrando que a coreografia trata dos processos comuns aos serviços envolvidos e não dos procedimentos realizados internamente por cada serviço.

O serviço Central de distribuição também compartilha dos mesmos princípios, ou seja, a busca pelo produto solicitado poderia ser realizada por consultas a base de dados ou outras rotinas que diferem do serviço implantado neste exemplo, no entanto os procedimentos internos do serviço não interferem na execução da coreografia. A implementação do serviço Central pode ser visualizada na Figura 40.

```
public class Central {

     static String livros[] = { "Java Como Programar",
                          "Fortaleza Digital",
                          "O Monge e o Executivo",
                          "O Doce Veneno do Escorpiao",
                          "A Hora da Verdade",
                          "Estratégia Competitiva" };

     static int estoque[] = {0,2,6,0,3,1};

     public static int QuantidadeEstoque( String Livro )
     {
       int quantidade_estoque = 0;

       for(int contador=0;contador<livros.length;contador++ )
       {
                if ( livros[contador].equals(Livro) ) {
                     quantidade_estoque = estoque[contador];
                }
       }

           return quantidade_estoque;
     }
}
```

FIGURA 40 – SERVIÇO WEB CENTRAL DE DISTRIBUIÇÃO (CENTRAL.JAVA)

7.3 COREOGRAFIA DOS SERVIÇOS WEB LIVRARIA E CENTRAL

A colaboração entre o serviço Livraria e o serviço Central se estabelece inicialmente na requisição da especificação do produto realizado pela Central ao serviço Livraria e após, a requisição da quantidade disponível do produto na Central pela Livraria. A Figura 41 apresenta o documento WS-CDL que descreve a coreografia entre esses serviços, sendo que este documento será a entrada necessária para processamento dos serviços na plataforma proposta.

```
<?xml version="1.0" encoding="UTF-8"?>
<package name="CoreografiaLivraria"
         targetNamespace="http://localhost:8080/axis"
         autor="Kleber Baptista"
         version="0.1">

<!-- Information Types -->
<informationType name="LivroConsultado"
type="QuantidadeEstoqueRequest"/>
<informationType name="QuantidadeDisponivel"
type="QuantidadeEstoqueResponse"/>

<!-- Tokens -->
<token informationType="uriType" name="centralRef"/>

<!-- Role Types -->
<roleType name="Livraria">
      <behavior name="Livraria-Central"
interface="Livraria.jws?wsdl"/>
</roleType>
<roleType name="Central">
      <behavior name="Central-Livraria"
interface="Central.jws?wsdl"/>
</roleType>

<!-- Relationship Types -->
<relationshipType name="Livraria_Central">
      <role type="Livraria" behavior="Livraria-Central"/>
      <role type="Central" behavior="Central-Livraria"/>
</relationshipType>

<!-- Channel Types -->
<channelType name="CentralChannel">
      <role type="Central"/>
      <reference>
            <token name="centralRef"/>
      </reference>
</channelType>
```

```
<!-- Choreographies -->

<choreography name="ConsultaEstoqueCentral" root="true">
  <relationship type="Livraria_Central"/>

    <variableDefinitions>
     <variable name="Livro" informationType="LivroConsultado"
mutable="false"/>
     <variable name="Quantidade"
informationType="QuantidadeDisponivel" mutable="false"/>
     <variable name="centralLibrary"
channelType="CentralChannel"/>
    </variableDefinitions>

    <sequence>
     <interaction channelVariable="centralLibrary"
name="EscolherLivro" operation="EscolherLivro"
initiate="true">
       <participate relationshipType="Livraria_Central"
fromRole="Central" toRole="Livraria" />
       <exchange action="respond" name="LivroExchange"
informationType="LivroConsultado">
         <send variable="cdl:getVariable(Livro,'','')"/>
         <receive variable="cdl:getVariable(Livro,'','')"/>
       </exchange>
     </interaction>

     <interaction channelVariable="centralLibrary"
name="ConsultarDisponibilidade" operation="QuantidadeEstoque">
       <participate relationshipType="Livraria_Central"
fromRole="Livraria" toRole="Central" />
       <exchange action="request" name="LivroExchange"
informationType="LivroConsultado">
         <send variable="cdl:getVariable(Livro,'','')"/>
         <receive variable="cdl:getVariable(Livro,'','')"/>
       </exchange>
       <exchange action="respond" name="QuantidadeExchange"
informationType="QuantidadeDisponivel">
         <send variable="cdl:getVariable(Quantidade,'','')"/>
         <receive
variable="cdl:getVariable(Quantidade,'','')"/>
       </exchange>
     </interaction>

   </sequence>
</choreography>
</package>
```

FIGURA 41 – COREOGRAFIA DOS SERVIÇOS LIVRARIA E CENTRAL (LIVRARIA.XML)

Destaca-se no documento WS-CDL a indicação dos serviços pelos elementos *roleTypes*, a declaração da coreografia responsável em colaborar os serviços Livraria e Central, assim como as declarações das operações necessárias e o fluxo das variáveis (elementos *exchange*).

A execução da plataforma proposta iniciará com a inserção do documento WS-CDL descrito acima, e os resultados serão detalhados na subseção subseqüente.

7.4 RESULTADOS DO PROCESSAMENTO DA PLATAFORMA

A etapa inicial da execução da plataforma proposta é a interpretação do documento WS-CDL. O resultado da interpretação do documento referente a coreografia dos serviços Livraria e Central pode ser visualizado na Figura 42. Nota-se que a plataforma realizou o processamento de todos os elementos contidos no documento. Ainda na Figura 42 destaca-se a exibição de todos os atributos dos elementos, assim como seus sub-elementos e seus respectivos atributos. Ao utilizar a plataforma o usuário é quem controla a visualização dos elementos no modo expandido, clicando sob os ícones representados pelos símbolos mais (+) e menos (-).

O processamento correto do interpretador de documentos demonstra a capacidade da plataforma em acessar todos os elementos contidos na linguagem WS-CDL, sendo que o resultado desta etapa estrutura a execução de todas as demais funcionalidades da plataforma.

Após a interpretação do documento WS-CDL a plataforma analisa e valida a coreografia dos serviços Livraria e Central exibindo a coreografia definida no documento e todas as operações previstas (Figura 43). O sucesso na execução dessa fase gera a possibilidade de processar a localização dos serviços envolvidos.

A Figura 44 apresenta o resultado da localização dos serviços Livraria e Central. Nota-se que os elementos estão representados graficamente na cor

verde, indicando que ambos foram localizados pela plataforma com sucesso. Neste momento a plataforma habilita a execução dos serviços Livraria e Central.

FIGURA 42 – INTERPRETAÇÃO DO DOCUMENTO COREOGRÁFICO LIVRARIA-CENTRAL

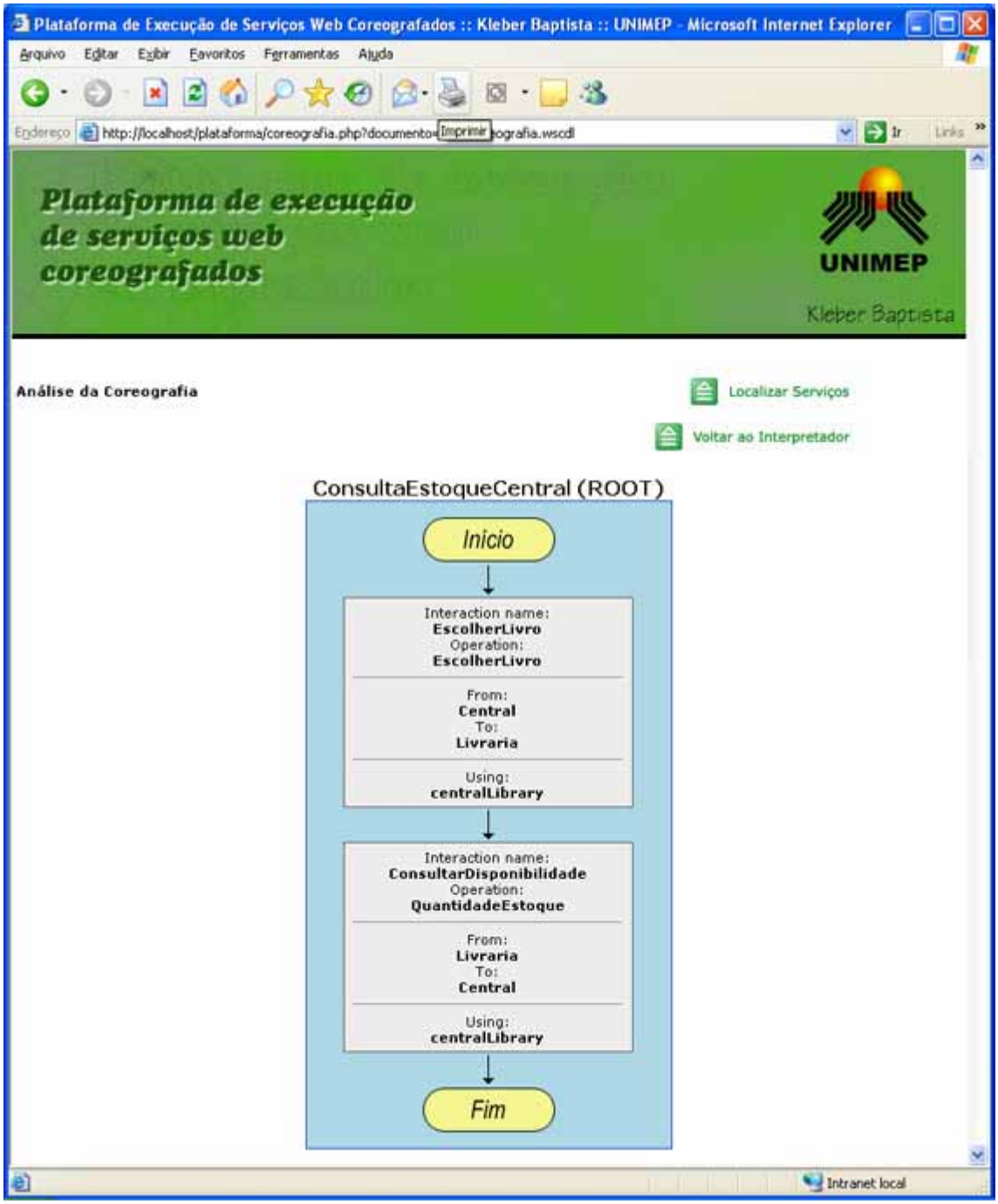

FIGURA 43 – COREOGRAFIA DOS SERVIÇOS LIVRARIA E CENTRAL

FIGURA 44 – LOCALIZAÇÃO DOS SERVIÇOS LIVRARIA E CENTRAL

Finalmente ocorre a execução da coreografia inserida na plataforma (Figura 45). Todo o processamento da coreografia é apresentado ao usuário da plataforma possibilitando visualizar o andamento de cada ação e os resultados obtidos. Na execução da coreografia entre os serviços Livraria e Central a plataforma identificou a coreografia inicial para execução assim como a ordem de chamada aos serviços e operações. Todas as variáveis foram declaradas (livro, quantidade e centralLibrary) e a primeira interação foi especificada (EscolherLivro), indicando os serviços envolvidos (de Central [fromRole] para Livraria [toRole]) assim como o tipo de chamada a ser realizada (*action: respond*). Em seguida o serviço responsável por esta interação foi instanciado, indicando o local e a data e hora exata da ação. A mensagem logo a seguir ('>>>OK.') representa que o serviço está preparado para receber a chamada da operação.

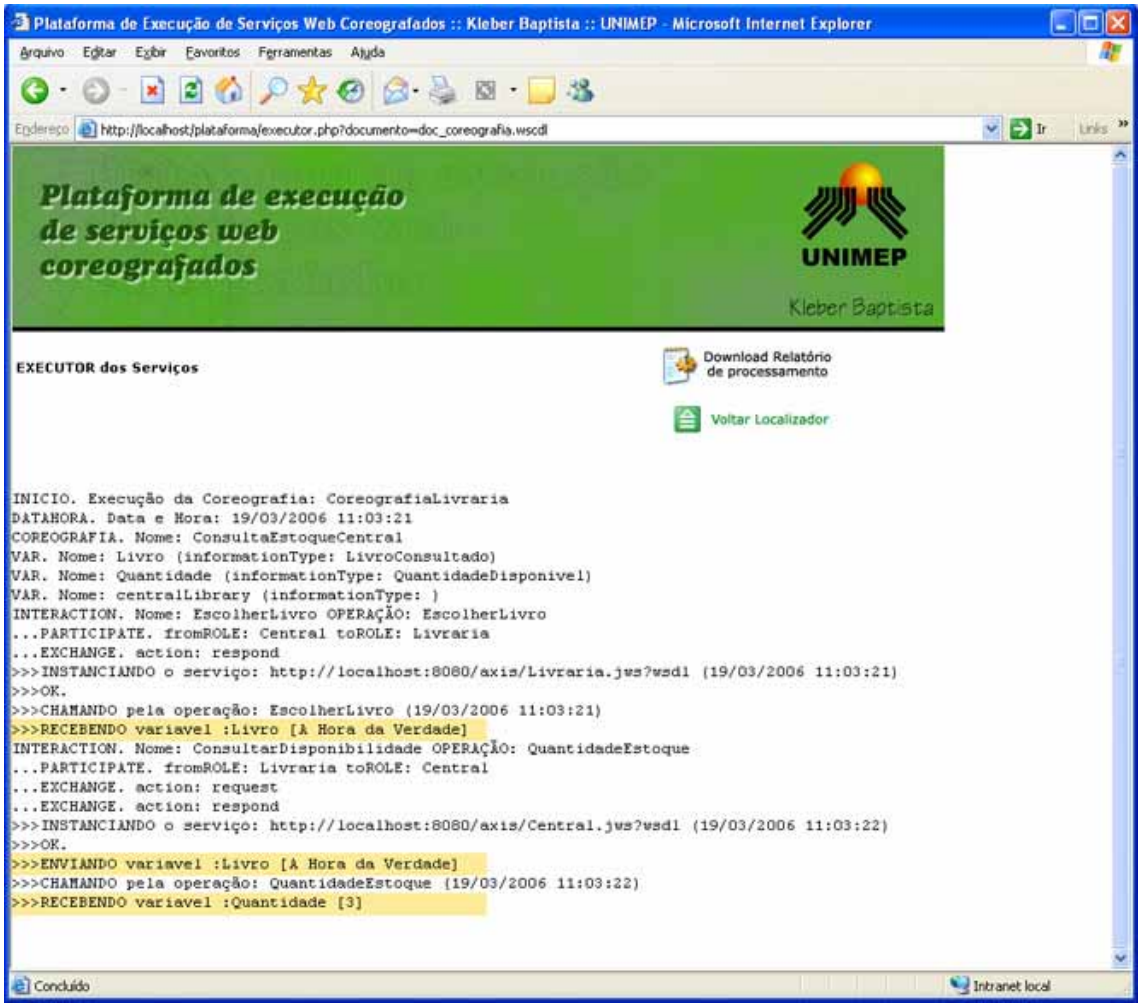

FIGURA 45 – RESULTADO DA EXECUÇÃO DOS SERVIÇOS LIVRARIA E CENTRAL

A chamada pela primeira operação (EscolherLivro) demonstra que a plataforma está criando automaticamente um *client* para assegurar a comunicação como o serviço Livraria, ou seja, quando uma operação é chamada a própria plataforma organiza os processos de interface com o serviço, evitando assim a necessidade do desenvolvedor em criar serviços específicos para chamar as operações dos serviços web.

Ao chamar a operação EscolherLivro o executor registra a data e a hora da ação possibilitando avaliar o tempo decorrido da efetivação da operação.

A ação a seguir é fundamental para a coreografia dos serviços, sendo que a plataforma coordena o recebimento do resultado da operação encaminhando o conteúdo retornado à variável específica. Nota-se no destaque da Figura 45 a

indicação do recebimento do texto ‘A hora da Verdade’ alocado à variável ‘Livro’.

Após a primeira interação a plataforma processa a segunda interação (ConsultarDisponibilidade), identificando novamente a operação, os serviços envolvidos (*from* Livraria *to* Central) e o tipo da chamada a ser realizada (desta vez o tipo *request-respond*). O serviço Central é instanciado com sucesso e desta vez a plataforma envia a variável recebida na interação anterior à operação QuantidadeEstoque. A operação é processada (registrando mais uma vez a data e hora do processamento) e por fim a plataforma coordena o recebimento da resposta da última ação prevista no documento WS-CDL: a quantidade em estoque na Central do produto solicitado pelo serviço Livraria.

7.5 CONTRIBUIÇÕES DA PLATAFORMA

Com a capacidade de executar a coreografia de serviços web participantes de um ambiente de colaboração e ainda prover uma interface gráfica que detalhe a interpretação de documentos WS-CDL a plataforma contribui em 2 aspectos principais.

A primeira contribuição está no favorecimento da compreensão, por parte de qualquer desenvolvedor que necessite participar de uma colaboração, dos requisitos e características que devem ser respeitadas para de fato possibilitar o sucesso na execução da coreografia de serviços web.

Ao criar diversas interfaces gráficas a plataforma apóia a interpretação da coreografia, evitando que todo desenvolvedor envolvido na colaboração necessite compreender todos os elementos técnicos contidos na linguagem WS-CDL. Essas facilidades se transformam diretamente em redução do tempo de desenvolvimento das soluções participantes da coreografia, no entanto este trabalho não avaliou o índice exato relativo a diminuição do tempo no desenvolvimento dos serviços, mas esse fator é dado como certo devido a minimização de algumas tarefas conforme relacionado ainda no decorrer desta subseção. Ainda nesse aspecto, a plataforma por ser totalmente disponibilizada

em um ambiente que facilita o acesso de seus usuários (a Internet, como exemplo) ajuda a propagar facilmente os detalhes relativos ao ambiente colaborativo dos serviços, existindo assim a possibilidade de empregar a plataforma como uma ferramenta de apoio ao desenvolvimento de projetos de serviços web em equipe, embora esse objetivo não é previsto neste trabalho.

O segundo aspecto se relaciona com os benefícios alcançados ao capacitar a plataforma em executar serviços web participantes de uma coreografia, tornando possível atender aos diversos objetivos planejados nesta dissertação de mestrado.

A plataforma é capaz de localizar todos os serviços web dispostos localmente ou remotamente, de instanciar todos esses serviços evitando que essa tarefa seja efetuada pelos desenvolvedores dos serviços e executar cada operação prevista na coreografia dos serviços.

Ao dispor todas essas funcionalidades a plataforma evita que o desenvolvedor das soluções implemente todas essas ações, como a localização de cada serviço, a chamada por esse serviço e a execução de cada operação prevista. Independente da facilidade propiciada aos desenvolvedores a plataforma evita a realização destas diversas tarefas. Para esclarecer melhor esses benefícios relacionados com o desenvolvimento dos serviços, relata-se abaixo a implementação de uma coreografia sem a aplicação da plataforma, utilizando-se do mesmo exemplo empregado para avaliar a plataforma.

Inicialmente, para coreografar os serviços Livraria e Central necessita-se da análise de cada um dos serviços visando elucidar todas as operações previstas para serem executadas no momento de por em prática a coreografia dos serviços. Através desta análise, cada desenvolvedor poderá mensurar a quantidade de implementações relativas as chamadas às operações dos serviços. Cada implementação identificada representa o desenvolvimento de um outro serviço-cliente que então executará cada operação prevista pelos serviços.

No caso específico dos serviços Livraria e Central, para executar as operações suportadas pelos respectivos serviços, há necessidade de desenvolver os serviços-clientes conforme apresentado na Figura 46 e Figura 47.

```
public class ConsultaLivraria {
     public static void main (String [] args) {
          String resposta;
          Livraria livraria = new Livraria();

          resposta = livraria.EscolherLivro();
          System.out.print("Livro requisitado: ");
          System.out.println(resposta);
     }
}
```

FIGURA 46 – CHAMADA AO SERVIÇO LIVRARIA (CONSULTALIVRARIA.JAVA)

```
public class ConsultaCentral {
   public static void main (String [] args) {
       int resposta;
       Central central = new Central();

       if(args.length > 0) {
         resposta = central.QuantidadeEstoque(args[0]);
         System.out.print("Quantidade de itens no estoque: ");
         System.out.println(resposta);
       } else {
         System.out.println("Informe o Livro");
       }
   }
}
```

FIGURA 47 – CHAMADA AO SERVIÇO CENTRAL (CONSULTACENTRAL.JAVA)

A Figura 46 implementa um serviço para chamar a operação EscolherLivro do serviço web Livraria e a Figura 47 implementa a chamada da operação QuantidadeEstoque do serviço Central. Após o desenvolvimento das chamadas aos serviços, há ainda a necessidade de os profissionais envolvidos na coreografia dos serviços compilar todas as implementações, respeitando as características da linguagem utilizada para gerar esses serviços. Ainda depois há a necessidade de efetuar manualmente ou implementar outro serviço para de fato executar e coordenar o fluxo de dados trocados entre os serviços

participantes da coreografia. A Figura 48 apresenta essas ações resultantes do exemplo Livraria-Central.

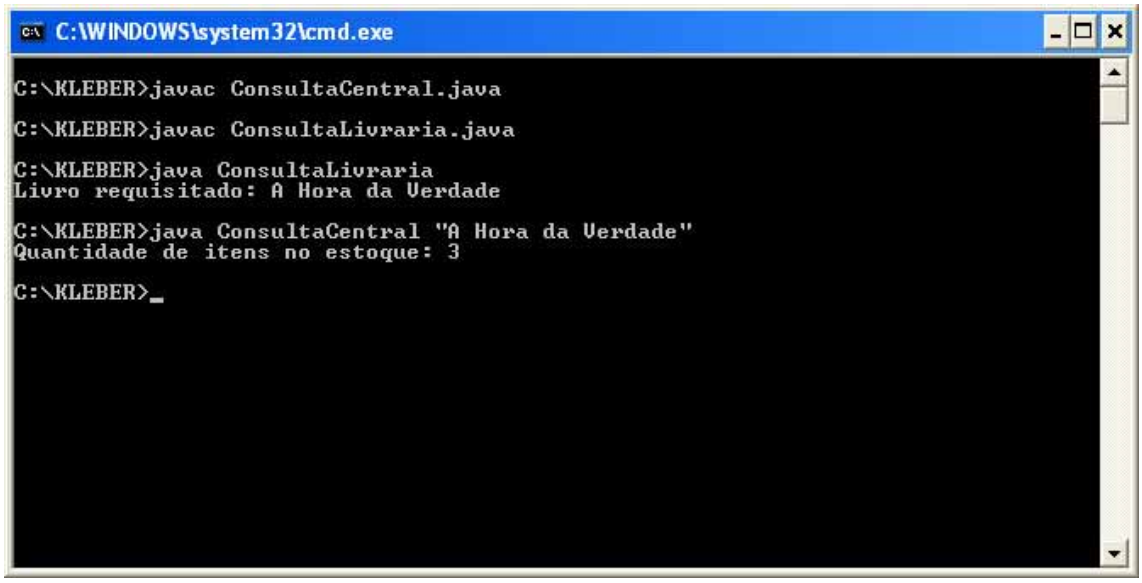

FIGURA 48 – CHAMADAS AOS SERVIÇOS LIVRARIA E CENTRAL

Nota-se na figura acima, inicialmente a execução de dois comandos *javac* necessários para compilar e gerar as classes da implementação referente as chamadas às operações dos serviços Livraria e Central. Posteriormente ao sucesso das compilações dos serviços de chamada são efetuadas manualmente suas execuções, obrigando inclusive a digitação na linha de comando do conteúdo da variável resultante do serviço anterior.

Todas as atividades descritas nesta subseção correspondem aos benefícios gerados pela plataforma, sendo que a plataforma assume a execução da coreografia dos serviços web. No exemplo Livraria-Central denota-se a facilidade apresentada pela plataforma, no entanto aplicações envolvendo mais serviços e mais operações elevariam exponencialmente os ganhos ao empregar a plataforma proposta, pois a quantidade maior de operações obrigaria ao desenvolvedor implementar mais serviços de chamadas além de dificultar a coordenação do fluxo de dados gerado pela colaboração dos serviços.

8 CONCLUSÕES

As coreografias de serviços impulsionam as integrações de aplicações que precisam compartilhar dados para executar operações com objetivos comuns. Essa necessidade está presente em qualquer tipo de aplicação que realiza troca de dados, no entanto tende-se a elevar a aplicação da coreografia em cenários envolvendo múltiplas entidades de negócios diferentes.

As tecnologias e padrões já existentes envolvendo serviços web disseminam aplicações capazes de prover serviços e realizar as trocas de dados que são necessárias para estabelecer a integração entre diferentes entidades, mas qualquer nova tecnologia ou ferramenta que agregue mais facilidade ao ambiente desses serviços proporcionará fortalecimento do uso dessas tecnologias, além de melhorar os resultados operacionais relativos com o seu desenvolvimento, como exemplo o tempo de desenvolvimento, o custo do projeto, a qualidade do serviço, entre outros.

Esta dissertação contribui no contexto do ambiente de serviços web e trata especificamente a capacidade de melhorar o desenvolvimento e a aplicação desses serviços através da utilização de uma plataforma de execução de serviços web que explora a coreografia de serviços por interpretação de documentos WS-CDL, sendo que um protótipo desta plataforma foi desenvolvido e utilizado mediante um exemplo proposto, possibilitando assim captar os benefícios e facilidades reais da execução de serviços web coreografados.

Os principais benefícios gerados e algumas considerações finais sobre a coreografia de serviços web por intermédio da plataforma proposta são:

- A apresentação e o detalhamento gráfico dos elementos que compõem a linguagem WS-CDL facilita a interpretação e a análise da coreografia dos serviços envolvidos em um ambiente de colaboração, agregando

assim benefícios diretos aos desenvolvedores de soluções empregando serviços web.

- A execução pela plataforma de todos os serviços participantes de uma coreografia evita que os desenvolvedores dos serviços web, envolvidos em um ambiente de colaboração, necessitem desenvolver serviços clientes para efetuar as chamadas das operações dos respectivos serviços servidores.

- É válido ressaltar que não há outros trabalhos publicados que objetivam a execução da coreografia de serviços empregando a WS-CDL. Nesse sentido este trabalho contribui também com a avaliação desta linguagem e demonstra que é possível efetuar a execução dos serviços web pela descrição da coreografia em documentos WS-CDL. Como a proposta da linguagem WS-CDL é recente, estima-se que este trabalho desperte a aplicação de ferramentas voltadas para a coreografia dos serviços web.

Avaliando os objetivos propostos por este trabalho concluem-se que a plataforma de coreografia proposta é capaz de interpretar coreografias (especialmente aquelas descritas em documentos WS-CDL), de analisar e validar as coreografias descritas nos respectivos documentos, de apresentar as interfaces dos serviços participantes da coreografia e de encapsular rotinas de chamadas de serviços web criando assim um ambiente de execução de serviços.

Possíveis extensões a este trabalho incluem:

- Fortalecimento do núcleo da plataforma de execução implementando todas as funcionalidades previstas pela linguagem WS-CDL.

- Especialização da plataforma em outras especificações e linguagens capazes de colaborar e orquestrar serviços web, como exemplo a BPEL4WS, criando assim uma ferramenta completa para manipulação de serviços web.

- Criar funcionalidades para melhorar o gerenciamento dos resultados da plataforma, como exemplo um repositório de dados para armazenar os resultados de todas as execuções, controle de usuários da plataforma e histórico dos serviços instanciados pela plataforma.

- Implantação de funcionalidades que capacitem a plataforma em substituir serviços durante a execução de uma coreografia. Descoberta automática.

REFERÊNCIAS BIBLIOGRÁFICAS

AALST, Wil van der. Don't Go with the flow: Web Services Composition Standards Exposed. *IEEE Intelligent Systems*, v. 18, n. 1, p. 72-76, janeiro/fevereiro 2003.

ANDREWS, Tony. et al. *Business Processes Execution Language for Web Services – Version 1.1*. BEA, IBM, Microsoft, SAP AG e Siebel System, fevereiro, 2005. Disponível em <http://www-128.ibm.com/developerworks/library/specification/ws-bpel/>. Acesso em maio, 2006.

ARKIN, Assaf. *Business Process Modeling Language.* Business Process Management Initiative, novembro, 2002. Disponível em: <http://xml.coverpages.org/BPML-2002.pdf>. Acesso em maio, 2006.

ARKIN, Assaf. et al (Ed.). *Web Service Choreography Interface (WSCI) 1.0*. W3C Note, agosto, 2002. Disponível em: <http://www.w3.org/TR/wsci/>. Acesso em maio 2006.

AUSTIN, Daniel. et al (Ed.). *Web Services Architecture Requirements*. W3C Working Group Note, fevereiro, 2004. Disponível em: <http://www.w3.org/TR/wsa-reqs/>. Acesso em maio, 2006.

BEYER, Dirk; CHAKRABERTI, Arindam; HENZINGER, Thomas. Web Service Interfaces. In: *Proceedings of the 14th International World Wide Web Conference (WWW2005)*. Chiba, Japão. Maio 2005.

BOOTH, David. et al (Ed.). *Web Services Architecture*. W3C Working Group Note, fevereiro 2004. Disponível em: <http://www.w3.org/TR/ws-arch/>. Acesso em maio 2006.

BRAY, Tim. et al (Ed.). *Extensible Markup Language (XML) 1.0 (Third Edition)*. W3C Recommendation, fevereiro, 2004. Disponível em: <http://www.w3.org/TR/REC-xml/>. Acesso em maio 2006.

CHOI, Wankyu. et al. *Beginning PHP 4 – Programando.* São Paulo: Makron Books, 2001.

CHRISTENSEN, Erik. et al (Ed.). *Web Services Description Language (WSDL) Version 1.1.* W3C Note, 15 março, 2001. Disponível em: <http://www.w3.org/TR/wsdl>. Acesso em maio 2006.

CLEMENT, Luc. et al (Ed.). *UDDI Version 3.0.2*. UDDI Spec Technical Committee Specification Draft, outubro, 2004. Disponível em: <http://uddi.org/pubs/uddi_v3.htm>. Acesso em maio 2006.

DAVIES, Jonh; FENSEL, Dieter; RICHARDSON, Marc. The Future of Web Services. In: *BT Technology Journal*, v.22, n.1, janeiro, 2004.

FALLSIDE, David C.; WALMSLEY, Priscilla (Ed.). *XML Schema Part 0: Primer Second Edition.* W3C Recommendation, outubro, 2004. Disponível em: <http://www.w3c.org/TR/xmlschema-0/>. Acesso em maio, 2006.

GAO, Xia. et al. Resource optimazation for Web Service Composition. In: *Proceedings of IEEE SCC2005*, julho, 2005.

HENDRICKS, Mack. et al. *Profissional Java Web Services.* 1. ed. Rio de Janeiro: Alta Books, 2002 . ISBN 85-88745-45-3.

KAVANTZAS, Nickolas. et al (Ed.). *Web Services Choreography Description Language Version 1.0*. W3C Candidate Recommendation, novembro, 2005. Disponível em: <http://www.w3.org/TR/ws-cdl-10/>. Acesso em maio 2006.

KOKASH, Natallia; D´ANDREA, Vincenzo. Service Oriented Computing and Coordination Models. In: *Proceedings of Challenges in Collaborative Engineering Workshop*. Sopron, Hungria. Abril 2005. pp. 95-103.

KREGER, Heather. *Web Services Conceptual Architecture1.0*. IBM Software Group, maio, 2001. Disponível em < http://www-3.ibm.com/software/solutions/ webservices/pdf/WSCA.pdf>. Acesso em 15 maio 2006.

LARA, Rubén. et al. Semantic web services: description requirements and current technologies. In: *International Workshop on Electronic Commerce, Agents and Semantic Web Services*, Pittsburg, PA, USA, Setembro, 2003.

LEYMANN, Frank. *Web Services Flow Language*. IBM Software Group, maio, 2001. Disponível em: <http://www-3.ibm.com/software/solutions/webservices/pdf/WSFL.pdf>. Acesso em maio, 2006.

MENDES, Manuel de Jesus. et al. Federation of Web Services In: *Jornadas Chilenas de Computacion*, Copiapó, 2002.

MITRA, Nilo (Ed.). *SOAP Version 1.2 Part 0: Primer*. W3C Recommendation, junho, 2003. Disponível em: <http://www.w3c.org/TR/soap12-part0/>. Acesso em maio, 2006.

PELTZ, Chris. Web Services Orchestration and Choreography. In: *IEEE Computer Society*, v.36, n.10, outubro, 2003. p. 46-52.

PELTZ, Chris. *Web Services Orchestration: a review of emerging technologies, tools, and standards*. Hewlett Packard Company, janeiro, 2003. Disponível em: <http://devresource.hp.com/drc/technical_white_papers/WSOrch/WSOrchestration.pdf>. Acesso em abril 2004.

SMITH, Roger. *Shall We Dance?* Oracle Technology, outubro, 2003. Disponível em: <http://www.oracle.com/technology/oramag/webcolumns/2003/techarticles/smith_wsc.html>. Acesso em 15 de maio 2006.

THATTE, Satish. *XLANG: Web Services for Business Process Design*. Microsoft Corporation, 2001. Disponível em: <http://www.gotdotnet.com/team/xml_wsspecs/xlang-c/default.htm>. Acesso em maio 2006.

TURNER, Mark Turner; Budgen, David; Brereton, Pearl. Turning Software into a Service. In*: IEEE Computer Society*, v.36, n.10, outubro, 2003. p. 38-44.

UDGIN, Martin. et al (Ed.). *SOAP Version 1.2 Part 1: Messaging Framework*. W3C Recommendation, junho, 2003. Disponível em: <http://www.w3c.org/TR/soap12-part1/>. Acesso em maio 2006.

YUSHI, Cheng; WAH, Lee Eng; LIMBU, Dilip Kumar. Web Services Composition – An Overview of Standards. In: *Synthesis Journal*, Fifth issue, ITSC Publication, 2004. pp. 137-150.

ANEXO A – INTERPRETADOR.PHP

```
<?php
/************************************
* INTERPRETADOR DE DOCUMENTOS WS-CDL (interpretador.php)
* Plataforma de execução de serviços web coreografados
* Autor: Kleber Baptista - Programa de Mestrado em Ciência da Computação - UNIMEP
* Versão = 0.1
* Última Atualização = março de 2006
************************************/
?>
<html>

<head>
<meta http-equiv="Content-Language" content="pt-br">
<meta http-equiv="Content-Type" content="text/html; charset=windows-1252">
<title>Plataforma de Execução de Serviços Web Coreografados :: Kleber
Baptista :: UNIMEP</title>
</head>

<style>
TD
{
    PADDING-LEFT: 5px;
    FONT-SIZE: 11px;
    COLOR: #000000;
    FONT-FAMILY: verdana, arial, 'Microsoft Sans Serif', helvetica, sans-serif
}
</style>

<body topmargin="0" leftmargin="0">

<?php
//GRAVAR O ARQUIVO PARA ALIMENTACAO AS DEMAIS FUNCOES DA PLATAFORMA
if( !isset($nome_documento) ) {
    $nome_documento = 'doc_coreografia.wscdl';
    copy($documento, $nome_documento);
```

```
}

//LEITURA DO DOCUMENTO WSCDL
$doc_wscdl = simplexml_load_file($nome_documento) or die("Não é um documento WS-CDL válido");

//DEFINIR ARRAY COM OS ELEMENTOS DO PACOTE (Contar elementos)
$elementos = get_object_vars($doc_wscdl);
?>

<table border="0" width="100%" cellspacing="1" cellpadding="0">
  <tr>
    <td width="100%"><a href="index.php"><img border="0" src="topo.jpg" width="780" height="156"></a></td>
  </tr>
  <tr>
    <td height="20" width="100%">
    </td>
  </tr>
  <tr>
    <td width="100%">
       <table border="0" width="700">
          <tr>
            <td width="50%"><b>Interpretador WS-CDL</b></td>
            <td width="50%"><a href="coreografia.php?documento=<?php echo $nome_documento;?>"><img src="botao_OK_analise.gif"
border="0" align="right" width="183" height="34"></a></td>
          </tr>
       </table>
    </td>
  </tr>
  <tr>
    <td height="20" width="100%"></td>
  </tr>
    <tr>
    <td width="100%">Informações do Pacote (package):<br>
    <?php
    //EXIBIR OS ATRIBUTOS DO ELEMENTO PACKAGE
    foreach($doc_wscdl->attributes() as $a => $b){
       echo $a,'="',$b,'"<br>';
    }
    ?>
    </td>
  </tr>
```

```
  <tr>
    <td height="20" width="100%"></td>
  </tr>
  <tr>
    <td width="100%">


 <TABLE cellSpacing=1 cellPadding=1 width="100%" border=0 bgColor="#000000">
 <tr id="pacote" bgColor="#aaaaaa">
    <td width="20%">Pacote</td>
    <td width="20%">Instâncias</td>
    <td width="20%">Elementos</td>
    <td width="20%">sub-elementos</td>
    <td width="20%">declarações/atributos</td>
 </tr>

                <tr bgColor="#cdcdcd" id="pacote.1" onMouseOver="mOver(this.id);" onMouseOut="mOut(this.id);"
onDblClick="Collapse(this.id)" style="DISPLAY: none">
                        <td><IMG id=img.pacote.1 style="CURSOR: hand" onclick="Collapse('pacote.1')" height =9 src="plus1.gif"
width=9 ><b> Channel Types (<?php echo @count($elementos["channelType"]);?>)</b></td>
                        <td></td>
                        <td></td>
                        <td></td>
                        <td></td>
                </tr>
                <?php
                //EXIBIR OS CHANNELTYPES
                //Contador do nível do channeltype (estrutura da tabela)
                $i=1;
                foreach($doc_wscdl->channelType as $a => $b) {
                    echo '
                <tr bgColor="#ededed" id="pacote.1.'.$i.'" onMouseOver="mOver(this.id);" onMouseOut="mOut(this.id);"
onDblClick="Collapse(this.id)" style="DISPLAY: none">
                    <td><IMG id=img.pacote.1.'.$i.' style="CURSOR: hand" onclick="Collapse(\'pacote.1.'.$i.'\')" height =9
src="plus1.gif" width=9 ></td>
                    <td>'.$b["name"].'</td>
                    <td></td>
                    <td></td>
                    <td>';
                    //EXIBIR OS ATRIBUTOS DO ELEMENTO CHANNELTYPE
                    $padrao_usage = 0;
```

```
                    $padrao_action = 0;
                    foreach($doc_wscdl->channelType[$i-1]->attributes() as $z => $y){
                        if ($z == "usage") {
                           $padrao_usage = 1;
                           echo $z,'="',$y,'"<br>';
                        }
                        else if ($z == "action") {
                           $padrao_action = 1;
                           echo $z,'="',$y,'"<br>';
                        }
                    }
                    if( $padrao_action == 0 )
                        echo 'Action = request<br>';
                    if( $padrao_usage == 0 )
                        echo 'Usage = Unlimited';

                    echo '
                    </td>
                </tr>';

                    //EXIBIR OS ELEMENTOS DECLARADOS DENTRO DO CHANNELTYPE
                    //Contador do nível dos elementos do channel type. Abaixo do $i.
                    $j = 1;
                    foreach($doc_wscdl->channelType[$i-1] as $c => $d) {
                       $verifica_e = get_object_vars($doc_wscdl->channelType[$i-1]->$c);

                       echo '
                        <tr bgColor="#ffffff" id="pacote.1.'.$i.'.'.$j.'" onMouseOver="mOver(this.id);"
onMouseOut="mOut(this.id);" style="DISPLAY: none">
                                <td>'.( count($verifica_e)>0 ? "<IMG id=img.pacote.1.$i.$j style=\"CURSOR: hand\"
onclick=\"Collapse('pacote.1.$i.$j')\" height=9 src=\"plus1.gif\" width=9>" : "" ).'</td>
                                <td></td>
                                <td>'.$c.'</td>
                                <td></td>
                                <td>';
                        foreach($doc_wscdl->channelType[$i-1]->$c->attributes() as $z => $y){
                            echo $z,'="',$y,'"<br>';
                        }
                        echo '
                                </td>
                        </tr>';
```

```
                        //EXIBIR SUBELEMENTOS (ULTIMO NIVEL PREVISTO NO CHANNELTYPE)
                        //Contador do nível dos elementos do channel type. Abaixo do $ie$j.
                        $k = 1;

                        foreach($doc_wscdl->channelType[$i-1]->$c->children() as $e => $f) {

                            echo '
                            <tr bgColor="#ffffff" id="pacote.1.'.$i.'.'$j.'.'.$k.'" onMouseOver="mOver(this.id);"
onMouseOut="mOut(this.id);" style="DISPLAY: none">
                                <td></td>
                                <td></td>
                                <td></td>
                                <td>'.$e.'</td>
                                <td>';

                        foreach($doc_wscdl->channelType[$i-1]->$c->{"$e"}[$k-1]->attributes() as $z => $y){
                            echo $z,'="',$y,'"<br>';
                        }
                            echo '
                                </td>
                        </tr>';
                        $k++;
                        }


                    $j++; //Incremento do foreach dos elementos do channeltype
                    }
                //INCREMENTO (última linha do foreach)
                $i++;
                }
                //FIM DO CHANNELTYPE
                ?>

                <tr bgColor="#cdcdcd" id="pacote.2" onMouseOver="mOver(this.id);" onMouseOut="mOut(this.id);"
onDblClick="Collapse(this.id)" style="DISPLAY: none">
                        <td><IMG id=img.pacote.2 style="CURSOR: hand" onclick="Collapse('pacote.2')" height =9 src="plus1.gif"
width=9 ><b> Choreographies (<?php echo @count($elementos["choreography"]);?>)</b></td>
                        <td></td>
                        <td></td>
                        <td></td>
```

```
                    <td></td>
                </tr>
                <?php
                //EXIBIR OS CHOREOGRAPHIES (CHOREOGRAPHY)
                //Contador do nível do choreographies (estrutura da tabela)
                $i=1;
                foreach($doc_wscdl->choreography as $a => $b) {
                    echo '
                <tr bgColor="#ededed" id="pacote.2.'.$i.'" onMouseOver="mOver(this.id);" onMouseOut="mOut(this.id);"
onDblClick="Collapse(this.id)" style="DISPLAY: none">
                    <td><IMG id=img.pacote.2.'.$i.' style="CURSOR: hand" onclick="Collapse(\'pacote.2.'.$i.'\')" height =9
src="plus1.gif" width=9 ></td>
                    <td>'.$b["name"].'</td>
                    <td></td>
                    <td></td>
                    <td>';
                    //EXIBIR OS ATRIBUTOS DO ELEMENTO choreographies
                    $padrao_root = 0;
                    foreach($doc_wscdl->choreography[$i-1]->attributes() as $z => $y){
                        if ( ($z == "root") and ($y == "true") ) {
                           $padrao_root = 1;
                           echo $z,'="<b>',$y,'"</b><br>';
                        }
                    }
                    if( $padrao_root == 0 )
                        echo 'root = "false"<br>';
                    echo '
                    </td>
                </tr>';

                    //EXIBIR OS ELEMENTOS DECLARADOS DENTRO DO COREOGRAPHY
                    //Contador do nível dos elementos do choreography. Abaixo do $i.
                    $j = 1;
                    foreach($doc_wscdl->choreography[$i-1] as $c => $d) {
                       $verifica_e = get_object_vars($doc_wscdl->choreography[$i-1]->$c);

                       echo '
                        <tr bgColor="#ffffff" id="pacote.2.'.$i.'.'.$j.'" onMouseOver="mOver(this.id);"
onMouseOut="mOut(this.id);" style="DISPLAY: none">
                            <td>'.( count($verifica_e)>0 ? "<IMG id=img.pacote.2.$i.$j style=\"CURSOR: hand\"
onclick=\"Collapse('pacote.2.$i.$j')\" height=9 src=\"plus1.gif\" width=9>" : "" ).'</td>
```

```
                        <td></td>
                        <td>'.$c.'</td>
                        <td></td>
                        <td>';
                    foreach($doc_wscdl->choreography[$i-1]->$c->attributes() as $z => $y){
                        echo $z,'="',$y,'"<br>';
                    }
                    echo '
                        </td>
                    </tr>';

                    //EXIBIR SUBELEMENTOS (ULTIMO NIVEL PREVISTO NO choreography)
                    //Contador do nível dos elementos do choreogarphy. Abaixo do $ie$j.
                    $k = 1;
                    $cont_perform = 0;
                    foreach($doc_wscdl->choreography[$i-1]->$c->children() as $e => $f) {
                        echo '
                        <tr bgColor="#ffffff" id="pacote.2.'.$i.'.'$j.'.'.$k.'" onMouseOver="mOver(this.id);"
onMouseOut="mOut(this.id);" style="DISPLAY: none">
                            <td></td>
                            <td></td>
                            <td></td>
                            <td>'.$e.'</td>
                            <td>';

                    if( $e == "perform" ) {
                        echo $doc_wscdl->choreography[$i-1]->$c->perform[$cont_perform]["choreographyName"];
                        $cont_perform++;
                    }
                    else {
                        foreach($doc_wscdl->choreography[$i-1]->$c->{"$e"}[$k-1]->attributes() as $z => $y){
                           echo $z,'="',$y,'"<br>';
                        }
                    }
                        echo '
                            </td>
                    </tr>';
                    $k++;
                    }
                $j++; //Incremento do foreach dos elementos do channeltype
                }
```

```
                //INCREMENTO (última linha do foreach)
                $i++;
                }
                //FIM DO COREOGRAPHY
                ?>
                <tr bgColor="#cdcdcd" id="pacote.3" onMouseOver="mOver(this.id);" onMouseOut="mOut(this.id);"
onDblClick="Collapse(this.id)" style="DISPLAY: none">
                        <td><IMG id=img.pacote.3 style="CURSOR: hand" onclick="Collapse('pacote.3')" height =9 src="plus1.gif"
width=9 ><b> Information Types (<?php echo @count($elementos["informationType"]);?>)</b></td>
                        <td></td>
                        <td></td>
                        <td></td>
                        <td></td>
                </tr>
                <?php
                //EXIBIR OS INFORMATION TYPES
                //Contador do nível do information types (estrutura da tabela)
                $i=1;
                foreach($doc_wscdl->informationType as $a => $b) {
                    echo '
                <tr bgColor="#ededed" id="pacote.3.'.$i.'" onMouseOver="mOver(this.id);" onMouseOut="mOut(this.id);"
onDblClick="Collapse(this.id)" style="DISPLAY: none">
                    <td></td>
                    <td>'.$b["name"].'</td>
                    <td></td>
                    <td></td>
                    <td>';
                    //EXIBIR OS ATRIBUTOS DO ELEMENTO INFORMATION TYPE
                    foreach($doc_wscdl->informationType[$i-1]->attributes() as $z => $y){
                        if ( $z != "name" )
                           echo $z,'="',$y,'"<br>';
                     }
                    echo '
                    </td>
                </tr>';
                //INCREMENTO (última linha do foreach)
                $i++;
                }
                //FIM DO INFORMATIONTYPE
                ?>
```

```
                <tr bgColor="#cdcdcd" id="pacote.4" onMouseOver="mOver(this.id);" onMouseOut="mOut(this.id);"
onDblClick="Collapse(this.id)" style="DISPLAY: none">
                        <td><IMG id=img.pacote.4 style="CURSOR: hand" onclick="Collapse('pacote.4')" height =9 src="plus1.gif"
width=9 ><b> Participant Types (<?php echo @count($elementos["participantType"]);?>)</b></td>
                        <td></td>
                        <td></td>
                        <td></td>
                        <td></td>
                </tr>
                <?php
                //EXIBIR OS PARTICIPANT TYPES
                //Contador do nível do participant types (estrutura da tabela)
                $i=1;
                foreach($doc_wscdl->participantType as $a => $b) {
                    echo '
                <tr bgColor="#ededed" id="pacote.4.'.$i.'" onMouseOver="mOver(this.id);" onMouseOut="mOut(this.id);"
onDblClick="Collapse(this.id)" style="DISPLAY: none">
                    <td></td>
                    <td>'.$b["name"].'</td>
                    <td></td>
                    <td></td>
                    <td>';
                    //EXIBIR OS ATRIBUTOS DO ELEMENTO PARTICIPANT TYPE
                    foreach($doc_wscdl->participantType[$i-1]->attributes() as $z => $y){
                           echo $z,'="',$y,'"<br>';
                     }
                    echo '
                    </td>
                </tr>';
                //INCREMENTO (última linha do foreach)
                $i++;
                }
                //FIM DO PARTICIPANTTYPE
                ?>
                <tr bgColor="#cdcdcd" id="pacote.5" onMouseOver="mOver(this.id);" onMouseOut="mOut(this.id);"
onDblClick="Collapse(this.id)" style="DISPLAY: none">
                        <td><IMG id=img.pacote.5 style="CURSOR: hand" onclick="Collapse('pacote.5')" height =9 src="plus1.gif"
width=9 ><b> Relationship Types (<?php echo @count($elementos["relationshipType"]);?>)</b></td>
                        <td></td>
                        <td></td>
                        <td></td>
```

```
                    <td></td>
                </tr>
                <?php
                //EXIBIR OS RELATIONSHIPTYPES
                //Contador do nível do relationship types (estrutura da tabela)
                $i=1;
                foreach($doc_wscdl->relationshipType as $a => $b) {
                    echo '
                <tr bgColor="#ededed" id="pacote.5.'.$i.'" onMouseOver="mOver(this.id);" onMouseOut="mOut(this.id);"
onDblClick="Collapse(this.id)" style="DISPLAY: none">
                    <td><IMG id=img.pacote.5.'.$i.' style="CURSOR: hand" onclick="Collapse(\'pacote.5.'.$i.'\')" height =9
src="plus1.gif" width=9 ></td>
                    <td>'.$b["name"].'</td>
                    <td></td>
                    <td></td>
                    <td>';
                    //EXIBIR OS ATRIBUTOS DO ELEMENTO RELATIONSHIPTYPE
                    foreach($doc_wscdl->relationshipType[$i-1]->attributes() as $z => $y){
                        if ( $z != "name" )
                           echo $z,'="',$y,'"<br>';
                    }

                    echo '
                    </td>
                </tr>';
                    //EXIBIR OS ELEMENTOS DECLARADOS DENTRO DO RELATIONSHIPTYPE
                    //Contador do nível dos elementos do relationship type. Abaixo do $i.
                    $j = 1;
                    foreach($doc_wscdl->relationshipType[$i-1] as $c => $d) {
                       $verifica_e = get_object_vars($doc_wscdl->relationshipType[$i-1]->$c);

                       echo '
                        <tr bgColor="#ffffff" id="pacote.5.'.$i.'.'.$j.'" onMouseOver="mOver(this.id);"
onMouseOut="mOut(this.id);" style="DISPLAY: none">
                                <td>'.( count($verifica_e)>0 ? "<IMG id=img.pacote.5.$i.$j style=\"CURSOR: hand\"
onclick=\"Collapse('pacote.5.$i.$j')\" height=9 src=\"plus1.gif\" width=9>" : "" ).'</td>
                                <td></td>
                                <td>'.$c.'</td>
                                <td></td>
                                <td>';
                        foreach($doc_wscdl->relationshipType[$i-1]->{"$c"}[$j-1]->attributes() as $z => $y){
```

```
                        echo $z,'="',$y,'"<br>';
                    }
                    echo '
                            </td>
                    </tr>';

                $j++; //Incremento do foreach dos elementos do relationshiptype
                }
            //INCREMENTO (última linha do foreach)
            $i++;
            }
            //FIM DO RELATIONSHIPTYPE
            ?>
            <tr bgColor="#cdcdcd" id="pacote.6" onMouseOver="mOver(this.id);" onMouseOut="mOut(this.id);"
onDblClick="Collapse(this.id)" style="DISPLAY: none">
                    <td><IMG id=img.pacote.6 style="CURSOR: hand" onclick="Collapse('pacote.6')" height =9 src="plus1.gif"
width=9 ><b> Role Types (<?php echo @count($elementos["roleType"]);?>)</b></td>
                    <td></td>
                    <td></td>
                    <td></td>
                    <td></td>
            </tr>
            <?php
            //EXIBIR OS RoleTYPES
            //Contador do nível do role types (estrutura da tabela)
            $i=1;
            foreach($doc_wscdl->roleType as $a => $b) {
                echo '
            <tr bgColor="#ededed" id="pacote.6.'.$i.'" onMouseOver="mOver(this.id);" onMouseOut="mOut(this.id);"
onDblClick="Collapse(this.id)" style="DISPLAY: none">
                    <td><IMG id=img.pacote.6.'.$i.' style="CURSOR: hand" onclick="Collapse(\'pacote.6.'.$i.'\')" height =9
src="plus1.gif" width=9 ></td>
                    <td>'.$b["name"].'</td>
                    <td></td>
                    <td></td>
                    <td>';
                    //EXIBIR OS ATRIBUTOS DO ELEMENTO ROLETYPE
                    foreach($doc_wscdl->roleType[$i-1]->attributes() as $z => $y){
                        if ( $z != "name" )
                           echo $z,'="',$y,'"<br>';
                    }
```

```
                    echo '
                    </td>
                </tr>';
                    //EXIBIR OS ELEMENTOS DECLARADOS DENTRO DO ROLETYPE
                    //Contador do nível dos elementos do role type. Abaixo do $i.
                    $j = 1;
                    foreach($doc_wscdl->roleType[$i-1] as $c => $d) {
                       $verifica_e = get_object_vars($doc_wscdl->roleType[$i-1]->$c);

                       echo '
                        <tr bgColor="#ffffff" id="pacote.6.'.$i.'.'.$j.'" onMouseOver="mOver(this.id);"
onMouseOut="mOut(this.id);" style="DISPLAY: none">
                                <td>'.( count($verifica_e)>0 ? "<IMG id=img.pacote.6.$i.$j style=\"CURSOR: hand\"
onclick=\"Collapse('pacote.6.$i.$j')\" height=9 src=\"plus1.gif\" width=9>" : "" ).'</td>
                                <td></td>
                                <td>'.$c.'</td>
                                <td></td>
                                <td>';
                        foreach($doc_wscdl->roleType[$i-1]->{"$c"}[$j-1]->attributes() as $z => $y){
                            echo $z,'="',$y,'"<br>';
                        }
                        echo '
                                </td>
                        </tr>';

                    $j++; //Incremento do foreach dos elementos do roletype
                    }
                //INCREMENTO (última linha do foreach)
                $i++;
                }
                //FIM DO ROLE TYPE
                ?>

                <tr bgColor="#cdcdcd" id="pacote.7" onMouseOver="mOver(this.id);" onMouseOut="mOut(this.id);"
onDblClick="Collapse(this.id)" style="DISPLAY: none">
                        <td><IMG id=img.pacote.7 style="CURSOR: hand" onclick="Collapse('pacote.7')" height =9 src="plus1.gif"
width=9 ><b> Token Locators (<?php echo @count($elementos["tokenLocator"]);?>)</b></td>
                        <td></td>
                        <td></td>
                        <td></td>
```

```
                    <td></td>
                </tr>
                <?php
                //EXIBIR OS TOKEN LOCATORS
                //Contador do nível do token locators (estrutura da tabela)
                $i=1;
                foreach($doc_wscdl->tokenLocator as $a => $b) {
                    echo '
                <tr bgColor="#ededed" id="pacote.7.'.$i.'" onMouseOver="mOver(this.id);" onMouseOut="mOut(this.id);"
onDblClick="Collapse(this.id)" style="DISPLAY: none">
                    <td></td>
                    <td>'.$b["tokenName"].'</td>
                    <td></td>
                    <td></td>
                    <td>';
                    //EXIBIR OS ATRIBUTOS DO ELEMENTO TOKEN LOCATOR
                    foreach($doc_wscdl->tokenLocator[$i-1]->attributes() as $z => $y){
                        if ( $z != "tokenName" )
                           echo $z,'="',$y,'"<br>';
                     }
                    echo '
                    </td>
                </tr>';

                //INCREMENTO (última linha do foreach)
                $i++;
                }
                //FIM DO TOKEN LOCATOR
                ?>
                <tr bgColor="#cdcdcd" id="pacote.8" onMouseOver="mOver(this.id);" onMouseOut="mOut(this.id);"
onDblClick="Collapse(this.id)" style="DISPLAY: none">
                    <td><IMG id=img.pacote.8 style="CURSOR: hand" onclick="Collapse('pacote.8')" height =9 src="plus1.gif"
width=9 ><b> Token (<?php echo @count($elementos["token"]);?>)</b></td>
                    <td></td>
                    <td></td>
                    <td></td>
                    <td></td>
                </tr>
                <?php
                //EXIBIR OS TOKEN
                //Contador do nível do token (estrutura da tabela)
```

```
                $i=1;
                foreach($doc_wscdl->token as $a => $b) {
                    echo '
                <tr bgColor="#ededed" id="pacote.8.'.$i.'" onMouseOver="mOver(this.id);" onMouseOut="mOut(this.id);"
onDblClick="Collapse(this.id)" style="DISPLAY: none">
                    <td></td>
                    <td>'.$b["name"].'</td>
                    <td></td>
                    <td></td>
                    <td>';
                    //EXIBIR OS ATRIBUTOS DO ELEMENTO TOKEN
                    foreach($doc_wscdl->token[$i-1]->attributes() as $z => $y){
                        if ( $z != "name" )
                           echo $z,'="',$y,'"<br>';
                     }
                    echo '
                    </td>
                </tr>';
                //INCREMENTO (última linha do foreach)
                $i++;
                }
                //FIM DO TOKEN
                ?>
</TABLE>
<SCRIPT LANGUAGE=javascript>
//<!--
      OpenNode('pacote');
//-->
</SCRIPT>
    </td>
  </tr>
  <tr>
    <td height="35" width="100%"></td>
  </tr>
</table>

</body>
</html>
```

ANEXO B – COREOGRAFIA.PHP

```
<?php
/************************************
* ANALISE DE DOCUMENTOS WS-CDL (coreografia.php)
* Plataforma de execução de serviços web coreografados
* Autor: Kleber Baptista - Programa de Mestrado em Ciéncia da Computação - UNIMEP
* Versão = 0.1
* Última Atualização = março de 2006
************************************/
?>
<html>

<head>
<meta http-equiv="Content-Language" content="pt-br">
<meta http-equiv="Content-Type" content="text/html; charset=windows-1252">
<title>Plataforma de Execução de Serviços Web Coreografados :: Kleber
Baptista :: UNIMEP</title>
</head>

<script language=javascript>
        var nodeHighlightColor = "#CDCDCD";
        var nodeMouseOverColor = "wheat";

        function mOver(Node)
        {       objNode = document.getElementById(Node);
                objNode.style.backgroundColor=nodeMouseOverColor;
        }
        function mOut(Node)
        {       objNode = document.getElementById(Node);
                objNode.style.backgroundColor=nodeHighlightColor;
        }
</script>

<body topmargin="0" leftmargin="0">

<table border="0" width="780" cellspacing="1" cellpadding="0">
```

```
  <tr>
    <td width="100%"><a href="index.php"><img border="0" src="topo.jpg" width="780" height="156"></a></td>
  </tr>
  <tr>
    <td height="20" width="100%">
    </td>
  </tr>
  <tr>
    <td width="100%">
       <table border="0" width="700">
          <tr>
            <td width="40%"><b><font face="Verdana" style="font-size:11px">Análise da Coreografia</font></b></td>
            <td width="60%"><a href="localizador.php?documento=<?php echo $documento;?>"><img src="botao_OK_localizador.gif"
border="0" align="right"></a></td>
          </tr>
          <tr>
            <td width="40%"></td>
            <td width="60%"><a href="interpretador.php?nome_documento=<?php echo $documento;?>"><img
src="botao_voltar_interpretador.gif" border="0" align="right"></td>
          </tr>
       </table>
    </td>
  </tr>
  <tr>
    <td height="20" width="100%"></td>
  </tr>
  <tr>
    <td width="100%">



<?php
//LEITURA DO DOCUMENTO WSCDL
$doc_wscdl = simplexml_load_file($documento) or die("Não é um documento WS-CDL válido");

//DEFINIR ARRAY COM OS ELEMENTOS DO PACOTE (Contar elementos)
$elementos = get_object_vars($doc_wscdl);
?>

<table border="0" width="100%" cellspacing="1" cellpadding="0">
  <tr>
```

```
    <td width="100%">
      <p align="center"></td>
  </tr>


<?php
//EXIBIR A COREOGRAFIA ROOT
  $i=0;
  foreach($doc_wscdl->choreography as $a => $b) {

    if( $b["root"] == "true" )  {

?>

  <tr>
    <td width="100%">
      <p align="center"><b><font face="Verdana"><a name="root"></a><?php echo $b["name"];?> (ROOT)</font></b></td>
  </tr>
  <tr>
    <td width="100%">
      <div align="center">
        <center>
        <table border="0" width="300" cellspacing="0" cellpadding="0" bgcolor="#ADD8E6">
          <tr>
            <td width="100%" style="border: 1 solid #2A4CB8">
              <table border="0" width="100%" cellspacing="1" cellpadding="0">
                <tr>
                  <td width="100%" height="10"></td>
                </tr>
                <tr>
                  <td width="100%">
                    <p align="center"><img border="0" src="flow_inicio.gif" width="110" height="38"></td>
                </tr>
                <tr>
                  <td width="100%">
                    <p align="center"><img border="0" src="flow_seta.gif" width="11" height="26"></td>
                </tr>

    <?php
       $k = 0;
       foreach( $doc_wscdl->choreography[$i]->sequence->children() as $e => $f ) {
```

```
        if( $e == "interaction" ) {
    ?>
                <tr>
                  <td width="100%">
                    <div align="center">
                      <center>
                      <table border="0" width="80%" cellspacing="0" cellpadding="0" bgcolor="#EDEDED">
                        <tr>
                          <td width="100%" style="border: 1 solid #666666">
                            <div align="center">
                              <center>
                              <table border="0" width="96%" cellspacing="0" cellpadding="0">
                                <tr>
                                  <td width="100%" align="center" height="5"></td>
                                </tr>
                                <tr>
                                  <td width="100%" align="center"><font face="Verdana" style="font-size:11px">Interaction
                                    name: <br><b><?php echo $f["name"];?></b></font></td>
                                </tr>
                                <tr>
                                  <td width="100%" align="center"><font face="Verdana" style="font-
size:11px">Operation:<br><b><?php echo $f["operation"];?></b></font></td>
                                </tr>
                                <tr>
                                  <td width="100%" align="center">
                                    <hr width="98%" size="0" color="#999999">
                                  </td>
                                </tr>
                                <tr>
                                  <td width="100%" align="center"><font face="Verdana" style="font-size:11px">From:<br><b><?php
echo $doc_wscdl->choreography[$i]->sequence->{"interaction"}[$k]->participate["fromRole"];?> </b></font></td>
                                </tr>
                                <tr>
                                  <td width="100%" align="center"><font face="Verdana" style="font-size:11px">To:<br><b><?php
echo $doc_wscdl->choreography[$i]->sequence->{"interaction"}[$k]->participate["toRole"];?> </b></font></td>
                                </tr>
                                <tr>
                                  <td width="100%" align="center">
                                    <hr width="98%" size="0" color="#999999">
                                  </td>
```

```
                                </tr>
                                <tr>
                                  <td width="100%" align="center"><font face="Verdana" style="font-
size:11px">Using:<br><b><?php echo $f["channelVariable"];?></b></font></td>
                                </tr>
                                <tr>
                                  <td width="100%" align="center" height="5"></td>
                                </tr>
                               </table>
                               </center>
                             </div>
                           </td>
                         </tr>
                       </table>
                       </center>
                     </div>
                   </td>
                 </tr>
                 <tr>
                   <td width="100%">
                     <p align="center"><img border="0" src="flow_seta.gif" width="11" height="26"></td>
                 </tr>

<?php
          } // fim do if $e == "interaction"
          else if ( $e == "perform" ) {

          $flag_existe = 0;
          foreach($doc_wscdl->choreography as $o => $p) {
             if ( $p["name"] == $f["choreographyName"] )
                $flag_existe = 1;
          }

?>
                 <tr>
                   <td width="100%">
                     <div align="center">
                       <table border="0" width="80%" cellspacing="0" cellpadding="0" <?php echo ( $flag_existe == 1 ?
'bgcolor="#CDCDCD"' : 'bgcolor="#FF0000"');?>>
                         <tr>
```

```
                        <td width="100%" id="perform<?php echo $k;?>" style="border: 1 solid #666666" <?php echo (
$flag_existe == 1 ? 'onMouseOver="mOver(this.id);" onMouseOut="mOut(this.id);" onDblClick="Collapse(this.id)"' : '');?>>
                          <div align="center">
                            <table border="0" width="96%" cellspacing="0" cellpadding="0">
                              <tr>
                                <td width="100%" align="center" height="5"></td>
                              </tr>
                              <tr>
                                <td width="100%" align="center"><font style="font-size: 11px" face="Verdana">Perform:
                                  </font></td>
                              </tr>
                              <tr>
                                <td width="100%" align="center"><font style="font-size: 11px" face="Verdana"><i><a
href="#<?php echo $f["choreographyName"];?>"><?php echo $f["choreographyName"];?></a></i></font></td>
                              </tr>
                              <tr>
                                <td width="100%" align="center" height="5"></td>
                              </tr>
                            </table>
                          </div>
                        </td>
                      </tr>
                    </table>
                  </div>
                </td>
              </tr>
              <tr>
                <td width="100%">
                  <p align="center"><img border="0" src="flow_seta.gif" width="11" height="26"></td>
              </tr>

<?php
          } // fim do if $e == perform

       $k++;
       }  //fim do foreach do conteudo do elemento sequence
?>

              <tr>
                <td width="100%">
                  <p align="center"><img border="0" src="flow_fim.gif" width="110" height="38"></td>
```

```
                </tr>
                <tr>
                  <td width="100%" height="10"></td>
                </tr>
              </table>
            </td>
          </tr>
        </table>
        </center>
      </div>
    </td>
  </tr>

<?php
     } //fim do if root == true (EXIBIR A COREOGRAFIA ROOT)
  //final do foreach
  $i++;
  }
?>


<?php
//EXIBIR AS DEMAIS COREOGRAFIAS
  $i=0;
  foreach($doc_wscdl->choreography as $a => $b) {

    if( $b["root"] != "true" )  {

?>
  <tr>
    <td width="100%" height="50"></td>
  </tr>
  <tr>
    <td width="100%">
      <p align="center"><font face="Verdana"><a name="<?php echo $b["name"];?>"></a><?php echo $b["name"];?></font></td>
  </tr>
  <tr>
    <td width="100%">
      <div align="center">
        <center>
        <table border="0" width="300" cellspacing="0" cellpadding="0" bgcolor="#ADD8E6">
```

```
        <tr>
          <td width="100%" style="border: 1 solid #2A4CB8">
            <table border="0" width="100%" cellspacing="1" cellpadding="0">
              <tr>
                <td width="100%" height="10"></td>
              </tr>
              <tr>
                <td width="100%">
                  <p align="center"><img border="0" src="flow_inicio.gif" width="110" height="38"></td>
              </tr>
              <tr>
                <td width="100%">
                  <p align="center"><img border="0" src="flow_seta.gif" width="11" height="26"></td>
              </tr>

    <?php
       $k = 0;
       foreach( $doc_wscdl->choreography[$i]->sequence->children() as $e => $f ) {

          if( $e == "interaction" ) {
    ?>
              <tr>
                <td width="100%">
                  <div align="center">
                    <center>
                    <table border="0" width="80%" cellspacing="0" cellpadding="0" bgcolor="#EDEDED">
                      <tr>
                        <td width="100%" style="border: 1 solid #666666">
                          <div align="center">
                            <center>
                            <table border="0" width="96%" cellspacing="0" cellpadding="0">
                              <tr>
                                <td width="100%" align="center" height="5"></td>
                              </tr>
                              <tr>
                                <td width="100%" align="center"><font face="Verdana" style="font-size:11px">Interaction
                                  name: <br><b><?php echo $f["name"];?></b></font></td>
                              </tr>
                              <tr>
                                <td width="100%" align="center"><font face="Verdana" style="font-
size:11px">Operation:<br><b><?php echo $f["operation"];?></b></font></td>
```

```
                                </tr>
                                <tr>
                                  <td width="100%" align="center">
                                    <hr width="98%" size="0" color="#999999">
                                  </td>
                                </tr>
                                <tr>
                                  <td width="100%" align="center"><font face="Verdana" style="font-size:11px">From:<br><b><?php
echo $doc_wscdl->choreography[$i]->sequence->{"interaction"}[$k]->participate["fromRole"];?> </b></font></td>
                                </tr>
                                <tr>
                                  <td width="100%" align="center"><font face="Verdana" style="font-size:11px">To:<br><b><?php
echo $doc_wscdl->choreography[$i]->sequence->{"interaction"}[$k]->participate["toRole"];?> </b></font></td>
                                </tr>
                                <tr>
                                  <td width="100%" align="center">
                                    <hr width="98%" size="0" color="#999999">
                                  </td>
                                </tr>
                                <tr>
                                  <td width="100%" align="center"><font face="Verdana" style="font-
size:11px">Using:<br><b><?php echo $f["channelVariable"];?></b></font></td>
                                </tr>
                                <tr>
                                  <td width="100%" align="center" height="5"></td>
                                </tr>
                              </table>
                              </center>
                            </div>
                          </td>
                        </tr>
                      </table>
                      </center>
                    </div>
                  </td>
                </tr>
                <tr>
                  <td width="100%">
                    <p align="center"><img border="0" src="flow_seta.gif" width="11" height="26"></td>
                </tr>
```

```
<?php
          } // fim do if $e == "interaction"
          else if ( $e == "perform" ) {

?>
                <tr>
                  <td width="100%">
                    <div align="center">
                      <table border="0" width="80%" cellspacing="0" cellpadding="0" bgcolor="#CDCDCD">
                        <tr>
                          <td width="100%" id="perform<?php echo $k;?>" style="border: 1 solid #666666"
onMouseOver="mOver(this.id);" onMouseOut="mOut(this.id);" onDblClick="Collapse(this.id)">
                            <div align="center">
                              <table border="0" width="96%" cellspacing="0" cellpadding="0">
                                <tr>
                                  <td width="100%" align="center" height="5"></td>
                                </tr>
                                <tr>
                                  <td width="100%" align="center"><font style="font-size: 11px" face="Verdana">Perform:
                                    </font></td>
                                </tr>
                                <tr>
                                  <td width="100%" align="center"><font style="font-size: 11px" face="Verdana"><i><a
href="#<?php echo $f["choreographyName"];?>"><?php echo $f["choreographyName"];?></a></i></font></td>
                                </tr>
                                <tr>
                                  <td width="100%" align="center" height="5"></td>
                                </tr>
                              </table>
                            </div>
                          </td>
                        </tr>
                      </table>
                    </div>
                  </td>
                </tr>
                <tr>
                  <td width="100%">
                    <p align="center"><img border="0" src="flow_seta.gif" width="11" height="26"></td>
                </tr>
```

```
<?php
          } // fim do if $e == perform

       $k++;
       }  //fim do foreach do conteudo do elemento sequence
?>
                <tr>
                  <td width="100%">
                    <p align="center"><img border="0" src="flow_fim.gif" width="110" height="38"></td>
                </tr>
                <tr>
                  <td width="100%" height="10"></td>
                </tr>
                <tr>
                  <td width="100%" height="10">
                    <p align="right"><font color="#3592B0"><a href="#root">root</a></font></td>
                </tr>
              </table>
            </td>
          </tr>
        </table>
        </center>
      </div>
    </td>
  </tr>
<?php
     } //fim do if root != true (EXIBIR AS DEMAIS COREOGRAFIAs)
  //final do foreach
  $i++;
  }
?>
     </td>
  </tr>
  <tr>
    <td height="35" width="100%"></td>
  </tr>
</table>

</body>
</html>
```

ANEXO C – LOCALIZADOR.PHP

```
<?php
header("Content-type: image/png");

/************************************
* LOCALIZADOR DE SERVICOS PARTICIPANTES NO DOCUMENTO WS-CDL (localizador.php)
* Plataforma de execução de serviços web coreografados
* Autor: Kleber Baptista - Programa de Mestrado em Ciéncia da Computação - UNIMEP
* Versão = 0.1
* Última Atualização = março de 2006
************************************/

//MONTAR ARRAY COM TODOS OS NAMESPACES DECLARADOS NO PACKAGE
$array_xmlns = array();

$xml = xml_parser_create();
xml_set_element_handler($xml,'start_handler','end_handler');
xml_set_character_data_handler($xml,'character_handler');
function start_handler($xml, $tag, $attributes) {
    global $level;
    global $array_xmlns;
    if( $tag == 'package') {
        foreach( $attributes as $key => $value ) {
           if( substr($key,0,5) == 'xmlns' )
               $array_xmlns[$key] = $value;
        }
    }
    $level++;
}
xml_parser_set_option($xml, XML_OPTION_CASE_FOLDING, false);
function end_handler($xml, $tag) {}
function character_handler($xml, $data) {}
xml_parse($xml,file_get_contents('doc_coreografia.wscdl'));
xml_parser_free($xml);

//LEITURA DO DOCUMENTO WSCDL
```

```
$doc_wscdl = simplexml_load_file('doc_coreografia.wscdl') or die("Não é um documento WS-CDL válido");

//DEFINIR ARRAY COM OS ELEMENTOS DO PACOTE (Contar elementos)
$elementos = get_object_vars($doc_wscdl);

//Instancia a variavel $var_namespace para informar no momento de localizar os serviços
$var_namespace = $doc_wscdl["targetNamespace"];

$array_elementos = "";
$array_interface = "";
$array_source = "";
$array_target = "";

//MONTAR O ARRAY_ELEMENTOS COMO O NAME DE TODOS ROLE TYPES
//E MONTAR ARRAY COM AS DESCRICOES DAS INTERFACES OU NAO QUANDO OCULTO (localizar servicos-WSDL)
$i = count($elementos["roleType"]);
for( $c=0; $c < $i; $c++) {
     $array_elementos[$c] = (string) $doc_wscdl->roleType[$c]["name"];
     $j=0;
     $interface = 'NAO';
     foreach($doc_wscdl->roleType[$c] as $k => $l) {
         foreach($doc_wscdl->roleType[$c]->{"$k"}[$j]->attributes() as $z => $y) {
             if( $z == 'interface' ) {
                 $interface = (string) $y;
             }
         }
         $array_interface["$array_elementos[$c]"] = $interface;
         $j++;
     }
}

$i = count($elementos["relationshipType"]);
for( $c=0; $c < $i; $c++) {
     $array_source[$c] = (string) $doc_wscdl->relationshipType[$c]->role[0]["type"];
     $array_target[$c] = (string) $doc_wscdl->relationshipType[$c]->role[1]["type"];
}

$array_cont_source = array_count_values($array_source);
$array_cont_target = array_count_values($array_target);

//echo '<pre>';
```

```
//var_dump($array_cont_source);
//var_dump($array_cont_target);

// Identificar os elementos com target = 0 (1º nível)
$k = 0;
$array_elementos_0 = "";
for( $i=0 ; $i<count($array_elementos) ; $i++ ) {

   if ( !array_key_exists($array_elementos[$i],$array_cont_target) ) {
      $array_elementos_0[$k] = $array_elementos[$i];
      $k++;
   }

} //linha 50

    $imagem = imagecreate(780,500);
    $branco   = imagecolorallocate($imagem, 255, 255, 255);
    $preto    = imagecolorallocate($imagem, 0, 0, 0);
    $verde    = imagecolorallocate($imagem, 0, 128, 0);
    $vermelho = imagecolorallocate($imagem, 204, 51, 0);

    $array_nucleos = array(); //armazenar o ponto central de cada elemento desenhado

function desenha_nivel($imagem, $array_nivel, $altura_inicio) {

    global $array_source;
    global $array_target;
    global $array_nucleos;
    global $array_interface;
    global $array_xmlns;
    global $var_namespace;

    global $branco;
    global $preto;
    global $verde;
    global $vermelho;

    $ponto_x = (imagesx($imagem)/2) - (((strlen(implode($array_nivel))*8) + (count($array_nivel)*25))/2);
    //imagestring($imagem,2,0,0,$ponto_x,$preto);

    for($i=0; $i<count($array_nivel); $i++) {
```

```
        $valor_acx = strlen($array_nivel[$i])*8;  //Valor a acrescentar a x

        //Localizar serviço e definir cor de preenchimento do elemento
        if( $array_interface["$array_nivel[$i]"] == 'NAO' )
            $cor_preenchimento = 'preto';
        else {
            $servico = '';
            if( stripos($array_interface["$array_nivel[$i]"],':') === false )
                $servico = $var_namespace . '/' . $array_interface["$array_nivel[$i]"];
            else {
                $ns = substr($array_interface["$array_nivel[$i]"],0,stripos($array_interface["$array_nivel[$i]"],':'));
                $ns = 'xmlns:'.$ns;
                $servico = $array_xmlns["$ns"] . '/'.
substr($array_interface["$array_nivel[$i]"],stripos($array_interface["$array_nivel[$i]"],':')+1,strlen($array_interface["$array
_nivel[$i]"]));
            }
            try {
               $client = new SoapClient("$servico");
               $cor_preenchimento = 'verde';
            } catch (Exception $exception) {
               $cor_preenchimento = 'vermelho';
            }
        }

        imagefilledrectangle($imagem,$ponto_x,$altura_inicio,$ponto_x + $valor_acx,$altura_inicio+50,$$cor_preenchimento);
        $valor_x_string = ($ponto_x + $valor_acx) - (strlen($array_nivel[$i])*6.8);
        imagestring($imagem,2,$valor_x_string,$altura_inicio+18,$array_nivel[$i],$branco);

        $array_nucleos["$array_nivel[$i]"]["x"] = ($valor_acx / 2) + $ponto_x;
        $array_nucleos["$array_nivel[$i]"]["y"] = $altura_inicio + 25;

        $ponto_x = $ponto_x + $valor_acx + 25;
    }

    //Montar array com os elementos do próximo nível
    $array_proximo = array();
    reset($array_nivel);
    foreach ($array_nivel as $key => $valor) {
       reset($array_source);
       foreach($array_source as $key2 => $valor2) {
          if( $valor == $valor2 ) {
```

```
            $array_proximo[] = (string) $array_target[$key2];
          }
      }
    }
    //retirar os elementos duplicados
    if( count($array_proximo) > 0 ) {
       $array_proximo = array_unique($array_proximo);
          desenha_nivel($imagem, $array_proximo, $altura_inicio+85); //linha 100
    }

}

    //Chamada a funçao para desenhar todos os elementos (roles)
    desenha_nivel($imagem, $array_elementos_0, 25);

function desenha_relacao($imagem) {

    global $array_source;
    global $array_target;
    global $array_nucleos;
    global $branco;
    global $preto;

    reset($array_source);
    for($i=0;$i<count($array_source);$i++) {
        $t1 = 0;
        $t2 = 0;
        $h = 0;
        $x = 0;

        if( $array_nucleos["$array_source[$i]"]["x"] > $array_nucleos["$array_target[$i]"]["x"] )
            $t1 = $array_nucleos["$array_source[$i]"]["x"] - $array_nucleos["$array_target[$i]"]["x"];
      else
            $t1 = $array_nucleos["$array_target[$i]"]["x"] - $array_nucleos["$array_source[$i]"]["x"];

      $t2 = $array_nucleos["$array_target[$i]"]["y"] - $array_nucleos["$array_source[$i]"]["y"];

        if( $t1 == 0 ) {
          $x1 = $array_nucleos["$array_source[$i]"]["x"];
            $x2 = $array_nucleos["$array_target[$i]"]["x"];
        }
```

```
        else {

           $hyp = sqrt( pow($t1,2) + pow($t2,2) );
           $seno = ($array_nucleos["$array_target[$i]"]["y"]-$array_nucleos["$array_source[$i]"]["y"])/$hyp;
           $seno = sin($seno); //linha 130
         $x = (25/tan($seno));

           if( $array_nucleos["$array_source[$i]"]["x"] > $array_nucleos["$array_target[$i]"]["x"] ) {
                $x1 = $array_nucleos["$array_source[$i]"]["x"] - $x;
                $x2 = $array_nucleos["$array_target[$i]"]["x"] + $x;
           }
           else {
                $x1 = $array_nucleos["$array_source[$i]"]["x"] + $x;
                $x2 = $array_nucleos["$array_target[$i]"]["x"] - $x;
           }
        }

         $y1 = $array_nucleos["$array_source[$i]"]["y"] + 25;
         $y2 = $array_nucleos["$array_target[$i]"]["y"] - 25;

        imageline($imagem,$x1,$y1,$x2,$y2,$preto);

    }
}

    desenha_relacao($imagem);

    imagepng($imagem);
    imagedestroy($imagem);
?>
```

ANEXO D – EXECUTOR.PHP

```
<?php
/************************************
* INTERPRETADOR DE DOCUMENTOS WS-CDL (executor.php)
* Plataforma de execução de serviços web coreografados
* Autor: Kleber Baptista - Programa de Mestrado em Ciéncia da Computação - UNIMEP
* Versão = 0.1
* Última Atualização = março de 2006
************************************/
?>
<html>

<head>
<meta http-equiv="Content-Language" content="pt-br">
<meta http-equiv="Content-Type" content="text/html; charset=windows-1252">
<title>Plataforma de Execução de Serviços Web Coreografados :: Kleber
Baptista :: UNIMEP</title>
</head>

<body topmargin="0" leftmargin="0">

<table border="0" width="100%" cellspacing="1" cellpadding="0">
  <tr>
    <td width="100%"><a href="index.php"><img border="0" src="topo.jpg" width="780" height="156"></a></td>
  </tr>
  <tr>
    <td width="100%"></td>
  </tr>
  <tr>
    <td width="100%">
       <table border="0" width="700">
          <tr>
            <td width="40%"><b><font face="Verdana" style="font-size:11px">EXECUTOR dos Serviços</font></b></td>
            <td width="60%"><a href="relatorio_wscdl.txt"><img src="botao_relatorio.gif" border="0" align="right"></a></td>
          </tr>
          <tr>
```

```
            <td width="20%"></td>
            <td width="80%"><a href="localizador.php?documento=<?php echo $documento;?>"><img
src="botao_voltar_localizador.gif" border="0" align="right"></td>
          </tr>
       </table>
    </td>
  </tr>
  <tr>
    <td height="20" width="100%"></td>
  </tr>
</table>

<?php

//LEITURA DO DOCUMENTO WSCDL
$doc_wscdl = simplexml_load_file($documento) or die("Não é um documento WS-CDL válido");
//DEFINIR ARRAY COM OS ELEMENTOS DO PACOTE (Contar elementos)
$elementos = get_object_vars($doc_wscdl);
//Instancia a variavel $var_namespace para informar no momento de localizar os serviços
$var_namespace = $doc_wscdl["targetNamespace"];

//MONTAR ARRAY COM TODOS OS NAMESPACES DECLARADOS NO PACKAGE
$array_xmlns = array();

$xml = xml_parser_create();
xml_set_element_handler($xml,'start_handler','end_handler');
xml_set_character_data_handler($xml,'character_handler');
function start_handler($xml, $tag, $attributes) {
    global $level;
    global $array_xmlns;
    if( $tag == 'package') {
        foreach( $attributes as $key => $value ) {
           if( substr($key,0,5) == 'xmlns' )
               $array_xmlns[$key] = $value;
        }
    }
    $level++;
}
xml_parser_set_option($xml, XML_OPTION_CASE_FOLDING, false);
function end_handler($xml, $tag) {}
function character_handler($xml, $data) {}
```

```
xml_parse($xml,file_get_contents('doc_coreografia.wscdl'));
xml_parser_free($xml);

//MONTAR O ARRAY_ELEMENTOS COMO O NAME DE TODOS ROLE TYPES
//E MONTAR ARRAY COM AS DESCRICOES DAS INTERFACES OU NAO QUANDO OCULTO (localizar servicos-WSDL)
$i = count($elementos["roleType"]);
for( $c=0; $c < $i; $c++) {
     $array_elementos[$c] = (string) $doc_wscdl->roleType[$c]["name"];
     $j=0;
     $interface = 'NAO';
     foreach($doc_wscdl->roleType[$c] as $k => $l) {
         foreach($doc_wscdl->roleType[$c]->{"$k"}[$j]->attributes() as $z => $y) {
             if( $z == 'interface' ) {
                 $interface = (string) $y;
             }
         }
         $array_interface["$array_elementos[$c]"] = $interface;
         $j++;
     }
}

//MONTAR ARRAY LOCALIZADOR (Posicao) DAS COREOGRAFIAS
$array_coreografia = array();
$i = 0;
foreach($doc_wscdl->choreography as $a => $b) {
   $nome = (string) $b["name"];
   $array_coreografia["$nome"] = $i;
   $i++;
}

//CRIAR ARQUIVO PARA GRAVAR LOG DA EXECUCAO
$rel = fopen('relatorio_wscdl.txt','w+');
fwrite($rel,"INICIO. Execução da Coreografia: ".$doc_wscdl["name"]."\n");
fwrite($rel, "DATAHORA. Data e Hora: ". date("d/m/Y H:m:s",time()) ."\n");

//FUNCAO PARA EXECUTAR AS COREOGRAFIA (POSSIBILITA CHAMADAS ANINHADAS)
function executar_coreografia($nome_coreografia) {

    global $rel;
    global $array_coreografia;
    global $doc_wscdl;
```

```
    global $array_xmlns;
    global $var_namespace;
    global $array_interface;

    fwrite($rel,"COREOGRAFIA. Nome: ".$nome_coreografia."\n");

    //SETAR AS VARIAVES DA COREOGRAFIA
    foreach( $doc_wscdl->choreography[$array_coreografia["$nome_coreografia"]]->variableDefinitions->children() as $chave =>
$valor ) {
         fwrite($rel,"VAR.");
         if( $valor["free"] == "true" ) {
             $nome_variavel = $valor["name"];
             fwrite($rel,"GLOBAL.");
             global $$nome_variavel;
         }
         fwrite($rel," Nome: ".$valor["name"]." (informationType: ".$valor["informationType"].")\n");
    }
    //IDENTIFICAR AS INTERACTIONS DA COREOGRAFIA
    $k=0;
    foreach( $doc_wscdl->choreography[$array_coreografia["$nome_coreografia"]]->sequence->children() as $chave => $valor ) {
      if( $chave == "interaction" ) {
         fwrite($rel,"INTERACTION. Nome: ".$valor["name"]." OPERAÇÃO: ".$valor["operation"]."\n");
         $operacao = $valor["operation"];
         //nivel de participate e exchange
         $z = 0;
         $tipos_exchanges = '';
         foreach( $doc_wscdl->choreography[$array_coreografia["$nome_coreografia"]]->sequence->{"$chave"}[$k]->children() as
$chave2 => $valor2 ) {
            if($chave2 == "participate" ) {
               fwrite($rel,"...PARTICIPATE. fromROLE: ".$valor2["fromRole"]." toROLE: ".$valor2["toRole"]."\n");
               $fromRole = $valor2["fromRole"];
               $toRole = $valor2["toRole"];
            }
            elseif ($chave2 == "exchange") {
               fwrite($rel,"...EXCHANGE. action: ".$valor2["action"]."\n");
               $tipos_exchanges.= $valor2["action"];
               //nivel de send e receive (elementos dentro de exchange)
               foreach( $doc_wscdl->choreography[$array_coreografia["$nome_coreografia"]]->sequence->{"$chave"}[$k]-
>{"$chave2"}[$z]->children() as $chave3 => $valor3 ) {
                 $action = $valor2["action"];
                 $vars_exchanges["$action"]["$chave3"] = (string) $valor3["variable"];
```

```
      }
      $z++;
   }
}
//instanciar o servico
if( stripos($toRole,':') === false )
    $servico = $var_namespace . '/' . $array_interface["$toRole"];
else {
    $ns = substr($toRole,0,stripos($toRole,':'));
    $ns = 'xmlns:'.$ns;
    $servico = $array_xmlns["$ns"] . '/'. substr($toRole,stripos($toRole,':')+1,strlen($toRole));
}
try {
    fwrite($rel,">>>INSTANCIANDO o serviço: ".$servico." (".date("d/m/Y H:m:s",time()) .")\n");
    $client = new SoapClient("$servico");
    fwrite($rel,">>>OK.\n");

    //definir a maneira de chamar pelo servico
    if( (stripos($tipos_exchanges,'request') !== false) AND (stripos($tipos_exchanges,'respond') !== false) ) {
        $variavel = substr($vars_exchanges["request"]["receive"],16,strlen($vars_exchanges["request"]["receive"]));
        $variavel = substr($variavel,0,strpos($variavel,','));
        if( strpos($variavel,":") !== false )
            $variavel = substr($variavel,strpos($variavel,':')+1,strlen($variavel));
        fwrite($rel, ">>>ENVIANDO variavel :".$variavel." [");
        if( isset($$variavel) )
            fwrite($rel,$$variavel);
        fwrite($rel,"]\n");
        fwrite($rel, ">>>CHAMANDO pela operação: ".$operacao." (".date("d/m/Y H:m:s",time()) .")\n");

        $variavel_2 = substr($vars_exchanges["respond"]["receive"],16,strlen($vars_exchanges["respond"]["receive"]));
        $variavel_2 = substr($variavel_2,0,strpos($variavel_2,','));
        if( strpos($variavel_2,":") !== false )
            $variavel_2 = substr($variavel_2,strpos($variavel_2,':')+1,strlen($variavel_2));

        fwrite($rel,">>>RECEBENDO variavel :".$variavel_2." [");
        try {
            if( isset($$variavel) ) {
                $contanier = $$variavel;
                $$variavel_2 = $client->__call("$operacao",array("$contanier"));
            }
            else
```

```
            $$variavel_2 = $client->__call("$operacao",array());
        fwrite($rel,$$variavel_2."]\n");

    } catch(Exception $exception) {
        fwrite($rel,"---FALHOU!");
    }
}
elseif (stripos($tipos_exchanges,'request') !== false) {
    $variavel = substr($vars_exchanges["request"]["receive"],16,strlen($vars_exchanges["request"]["receive"]));
    $variavel = substr($variavel,0,strpos($variavel,','));
    if( strpos($variavel,":") !== false )
        $variavel = substr($variavel,strpos($variavel,':')+1,strlen($variavel));
    fwrite($rel, ">>>ENVIANDO variavel :".$variavel." [");
    if( isset($$variavel) )
        fwrite($rel,$$variavel);
    fwrite($rel,"]\n");
    fwrite($rel, ">>>CHAMANDO pela operação: ".$operacao." (".date("d/m/Y H:m:s",time()) .")\n");

    try {
        if( isset($$variavel) )
            $client->__call("$operacao",array("$$variavel"));
        else
            $client->__call("$operacao",array());
    } catch(Exception $exception) {
        fwrite($rel,"---FALHOU!");
    }
}
elseif (stripos($tipos_exchanges,'respond') !== false) {
    fwrite($rel, ">>>CHAMANDO pela operação: ".$operacao." (".date("d/m/Y H:m:s",time()) .")\n");
    $variavel_2 = substr($vars_exchanges["respond"]["receive"],16,strlen($vars_exchanges["respond"]["receive"]));
    $variavel_2 = substr($variavel_2,0,strpos($variavel_2,','));
    if( strpos($variavel_2,":") !== false )
        $variavel_2 = substr($variavel_2,strpos($variavel_2,':')+1,strlen($variavel_2));

    fwrite($rel,">>>RECEBENDO variavel :".$variavel_2." [");
    try {
        $$variavel_2 = $client->__call("$operacao",array());
        fwrite($rel,$$variavel_2."]\n");
    } catch(Exception $exception) {
        fwrite($rel,"---FALHOU!");
    }
```

```
            }
         } catch (Exception $exception) {
            fwrite($rel,"---FALHOU!\n");
         }
         //echo '<pre>';
         //var_dump($vars_exchanges);
      }
      //chamar pelas outras coreografias - aninhadas pela funcao executar_coreografia
      elseif( $chave = 'perform' ) {
          executar_coreografia($valor["choreographyName"]);
      }
      $k++;
    }
} //FIM Da Funcao de executar os servicos (executar_coreografia())

//LOCALIZAR A COREOGRAFIA ROOT
foreach($doc_wscdl->choreography as $a => $b) {

   if( $b["root"] == "true" )  {

       executar_coreografia($b["name"]);

   }

}

//FECHAR RELATORIO WSCDL (LOG DE EXECUCAO)
fclose($rel);

echo '<pre>';
readfile('relatorio_wscdl.txt');
?>
```