UNIVERSIDADE DO VALE DO RIO DOS SINOS

Programa Interdisciplinar de Pós-Graduação em

Computação Aplicada

Sérgio Larentis Junior

SINS – Um Ambiente para Geração de Aplicações Baseadas em Serviços

Sérgio Larentis Junior

SINS – Um Ambiente para Geração de Aplicações Baseadas em Serviços

Dissertação apresentada à Universidade do Vale do Rio dos Sinos como requisito parcial para a obtenção do título de Mestre em Computação Aplicada.

Orientador: Prof. Dr. Jorge Luis Victória Barbosa

São Leopoldo

2007

L321s Larentis Júnior, Sérgio

SINS - Um ambiente para geração de aplicações baseadas em serviços / por Sérgio Larentis Júnior, 2007.

110 f. il. ; 30cm.

Dissertação (mestrado) -- Universidade do Vale do Rio dos Sinos, Programa de Pós-Graduação em Computação Aplicada, 2007.

"Orientação: Prof. Dr. Jorge Luis Victória Barbosa, Ciências Exatas e Tecnológicas".

1. *Web service*. 2. *Service Oriented Architecture* (SOA). 3. Arquitetura orientada a serviços. I. Título.

CDU 004.738.52:004.4

Catalogação na Publicação: Eliete Mari Doncato Brasil - CRB 10/1184

AGRADECIMENTOS

A minha esposa, Andrêsa Larentis, pelo apoio pessoal e intelectual, disciplina e companheirismo.

Aos meus pais, Sérgio Larentis e Clair Larentis, por terem ajudado, direta ou indiretamente, no meu caminho até aqui.

Ao meu pai, Sérgio Larentis, pelo exemplo de superação e sucesso.

Ao professor Sérgio Crespo, pelas críticas construtivas ao longo do mestrado.

Ao meu orientador, Jorge Barbosa, pela paciência, dedicação e apoio recebido ao longo do mestrado.

RESUMO

A arquitetura SOA (*service oriented architecture*) possibilita que serviços sejam desenvolvidos em linguagens diversas, orquestrados e combinados de modo a se obter aplicações, as chamadas *Composite Applications.* Apesar dos grandes avanços no que diz respeito à *Web Services* e IDEs de desenvolvimento, ainda não há um ambiente que permita a geração destas *Composite Applications* sem a necessidade de codificação e sem a necessidade de intervenção de um profissional da área de desenvolvimento de software.

O SINS, apresentado por este trabalho, é um ambiente capaz de gerar *Composite Applications* em tempo real, consumindo serviços pré-existentes. Tendo a vantagem de não necessitar de codificação ou da intervenção de um profissional da área de *software*.

Palavras-chave : *Web Services, composite applications*, SOA, *services oriented architecture*, arquitetura orientada a serviços.

ABSTRACT

SOA (service oriented architecture) makes possible that services developed in several languages be organized and combined to obtain applications (called Composite Applications). Despite the advances in Web Services and development IDEs, there is not an environment capable of generate Composite Application without the need of coding and without the need of a software development professional participation.

SINS, showed in this work, is an environment capable of generate Composite Applications in real time, consuming the existent services with the advantage that it doesn't need any coding to do this task.

Key-words: *Web Services, composite applications, SOA, services oriented architecture.*

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS

API	*Application Program Interface*
BAM	*Business Activity Monitor*
BPEL	*Business Process Execution Languagem*
BPM	*Business Process Management*
DCOM	*Distributed Component Object Model*
EJB	*Enterprise Java Beans*
ERP	*Enterprise Resource Planning*
ESA	*Enterprise Services Architecture*
ESB	*Enterprise Service Bus*
FTP	*File Transfer Protocol*
HTML	*HiperText Markup Language*
HTTP	*HiperText Transfer Protocol*
IDE	*Integrated Develoment Environment*
IIOP	*Internet Inter-ORB Protocol*
JMS	*Java Message Service*
MIME	*Multipurpose Internet Mail Extension*
OASIS	*Organization for the Advancement of Structured Information Standards*
QoS	Qualidade de serviço
RMI	*Remote Method Invocation*
RPC	*Remote Procedure Call*
SGML	*Standard General Markup Language*
SINS	*SINS Is Not SOA*
SMTP	*Simple Object Access Protocol*
SOA	*Service Oriented Architecture*
TI	Tecnologia da Informação
UDDI	*Universal Distribution Discovery and Interoperability*
W3C	*Word Wide Web Consortium*
WSDL	*Web Services Description Language*
WSRP	*Web Service Remote Portlets*
XML	*Extensible Markup Language*

SUMÁRIO

1 INTRODUÇÃO

Assim como houve a revolução industrial, onde linhas de montagem e processos deram um grande ganho de capacidade de produção baseando-se na idéia de que componentes padronizados, ainda que oriundos de fornecedores diferentes pudessem ser pragmaticamente unidos até se chegar a um resultado final, eis que se chega ao que já está sendo informalmente denominado "era da industrialização do software". Essa idéia, assim como as linhas de produção das indústrias, está baseada na máxima de que pequenos componentes podem ser feitos de diferentes formas, desde que respeitem os padrões de integração para um dito resultado final.

Nesse novo contexto, surge o desafio de normalizar o que poderiam ser esses componentes. Os paradigmas atuais, baseados em objetos, classes ou *includes*, não conseguem dar a correta idéia de componentização pela falta de interoperabilidade entre plataformas ou até mesmo paradigmas. Fato de fácil comprovação pela atual dificuldade entre integração de componentes em .Net e Java ou de extração de dados de um *mainframe* para uso em outra aplicação.

Uma forma de tentar resolver os problemas citados anteriormente e possibilitar a analogia com as linhas de produção é o chamado SOA (*Service Oriented Architecture*), que tem como pilares principais a interoperabilidade, a independência de componentes e o reuso. Na prática, isso se dá não mais desenvolvendo software pensando nas funcionalidades, mas sim em pacotes de serviços comuns que possam ser agrupados a fim de permitir o reuso e a escalabilidade das aplicações. Os *Web Services*, que podem ser entendidos como um dos principais viabilizadores dessa idéia (ainda que os serviços possam se comunicar através de corba, *sockets* ou similares), podem ser escritos em linguagens diversas, plataformas diversas e, ainda assim, interagirem de forma a conseguir o que seria o resultado desta "linha de produção". [TIInside, 2007a]

É possível inferir, então, que a construção de um sistema desse tipo seja demasiado oneroso na etapa de especificação, modelagem e construção. Essa desvantagem é compensada pelo ganho obtido com o reuso no desenvolvimento de novas funcionalidades (ou até mesmo novos sistemas), além da economia proporcionada pelo fato da arquitetura ser distribuída e de

plataforma independente, permitindo o uso da tecnologia de melhor custo x benefício, sem a fidelização costumeira dos paradigmas de confecção de software atuais, acabam por compensar o maior investimento inicial. [IWeek, 2006]

Resumindo, há uma arquitetura de produção de software que permite o desenvolvimento em escala, independência de fornecedores e tecnologia (apenas atrelada a padrões) chamada SOA, há um bom meio para uso dessa arquitetura, que são os *Web Services* e há, ainda, o fato que não há nenhuma grande revolução técnica atrelada a isso, o que poderia significar que o tempo necessário para a aplicação do SOA seria bastante aceitável. Sendo assim, uma pergunta naturalmente surge: Por que ele já não é amplamente usado? A resposta a essa questão pode ser feita com outras duas perguntas: O SOA já é maduro o suficiente para uma aplicação de larga escala? O que fazer com as aplicações legadas? A resposta à primeira pergunta só o tempo dará (ainda que os poucos *cases* existentes atualmente apontem que sim) e a resposta à segunda pode ser desde um simplista e oneroso "Reescreva!" até um complexo e desafiador "Compatibilize!". É bastante razoável que formas de compatibilizar os esforços de produção de software despendidos ao longo dos últimos anos, já com tecnologias atuais, não sejam perdidos por conta dessa nova arquitetura e, ainda que grandes empresas já estejam olhando para isso, o que temos hoje são iniciativas isoladas que não constituem um padrão.

Obviamente, o espaço criado pela falta de soluções contundentes de como compatibilizar as aplicações legadas com as novas que sejam desenvolvidas em SOA ou de como gerar aplicações baseadas em serviços já existentes de forma rápida, é de interesse também acadêmico o que, por fim, inspirou a realização deste trabalho.

1.1 Motivação

Nos últimos anos houve um grande desenvolvimento no que diz respeito à engenharia de software e, esse desenvolvimento, parece estar se focando principalmente em duas vertentes: Qualidade e Produtividade.

Quanto à qualidade, podemos ver que modelos de maturidade do processo de desenvolvimento de software têm sido criados, melhorados e implementados por empresas privadas em larga escala, provando que o esforço despendido nessa competência não tem sido em vão.

Quanto à produtividade, podemos ver paradigmas, linguagens e IDEs evoluindo de forma acentuadamente rápida, afirmação facilmente constatável em uma análise ao *framework* .net, da Microsoft, que em poucos anos já está em sua quarta versão e sua IDE, o *Visual Studio*, em sua terceira versão ou então pelo *Ruby on Rails*, que trabalha com um conceito de "pouca codificação" onde quase tudo se propõe a estar pronto.

Paralelo a tudo isso nota-se que o SOA surge como proposta complementar aos esforços atuais já despendidos nestas duas vertentes. Tendo a idéia de que as regras de negócio e acesso a dados de um software ficam encapsuladas em serviços independentes, temos um novo cenário onde o reuso pode se tornar algo mais cotidiano dos arquitetos de software, talvez até mesmo chegando ao ponto que aplicações baseadas em serviços já existentes poderiam ser feitas sem a interferência ou necessidade de um desenvolvedor de software. [TIInside, 2007b]

Pode-se, então, inferir que:

- o SOA tende a ajudar na qualidade, pois serviços já implementados e testados serão usados mais repetidamente diminuindo a chance de haver necessidade de reimplementar algo;
- o SOA tende a ajudar na produtividade, pois estes mesmos serviços serão usados em todas as aplicações em que se fizerem necessários, reduzindo a necessidade de criação de interfaces, migração de plataforma ou afins.

No entanto podemos notar que, apesar de grandes empresas como Microsoft, Oracle, IBM e outras estarem bastante focadas em SOA, ainda não há um ambiente que organize de forma intuitiva os vários serviços disponíveis e gere as aplicações de forma completa e dinâmica, e principalmente, com meios que um usuário possa gerar suas aplicações sem intervenção de um profissional da área de software, o que motiva a criação do SINS (acrônimo recursivo de *Sins Is Not* SOA) proposto por esse trabalho.

1.2 Problema

De acordo com pesquisa feita pelo *Gartner Group* no último trimestre do ano de 2006, praticamente 100% dos CIOs (*Chief Information Office*, cargo análogo ao Diretor de

Tecnologia) das maiores empresas do mundo pretendem desenvolver novas aplicações sob a plataforma SOA em um prazo máximo de cinco anos [Gartner, 2006].

Ainda que muitas empresas e arquitetos de software estejam se preparando para esta nova realidade, nota-se que o SOA está bastante aderente apenas a realidade dos profissionais de informática, que irão implementar os serviços, e aos profissionais de negócio, que são fundamentais na especificação dos serviços. Ou seja, ainda não há nenhum grande esforço no que diz respeito a tornar a manipulação dos serviços acessível ao usuário final sem intervenção de profissionais de TI, o que representaria uma grande economia de tempo e dinheiro as empresas.

Sendo assim, pode-se dizer que o intuito deste trabalho é responder as perguntas abaixo:

"Uma vez que se tenha um grande conjunto de serviços, como gerenciá-los de forma a facilitar seu uso?".

"Como permitir que usuários criem aplicativos baseados em serviços de forma rápida e sem a necessidade da intervenção de um profissional de software?".

Uma vez respondidas, crê-se que este trabalho terá dado a contribuição proposta no contexto em que está inserido.

1.3 Objetivos do Trabalho

1.3.1 Objetivos Gerais

Esse trabalho explana a criação do SINS (acrônimo recursivo de *Sins Is Not* SOA); um ambiente para geração de aplicações baseadas em serviços. Este ambiente é capaz de utilizar serviços existentes em um contexto e disponibilizados em forma de *Web Service*, catalogá-los para posterior uso e, por fim, prover uma interface ao usuário que lhe permita compor aplicações baseadas nestes serviços.

Para validação do trabalho, criou-se um estudo de caso sobre um protótipo do ambiente, onde serviços (em *Web Services*) foram mapeados e disponibilizados ao usuário final em uma interface gráfica que foi usada para criar aplicações. Por fim, tais aplicações

ficaram acessíveis através de uma interface web provando que, tanto o mapeamento quanto a geração de aplicações através de uma interface ao usuário, é factível e representa uma considerável economia no que diz respeito a desenvolvimento baseado em SOA.

1.3.2 Objetivos Específicos

Pontualmente, almeja-se atender os objetivos abaixo, além dos objetivos macros já citados:

- Estudar a arquitetura orientada a serviços;
- Possibilitar que serviços disponibilizados em *Web Services* diversos possam ser utilizados de forma conjunta para criação de uma aplicação;
- Permitir que o próprio usuário, sem necessidade de programação, possa criar uma aplicação baseada nos serviços disponibilizados pelo ambiente;
- Provar a viabilidade do trabalho através do estudo de caso.

1.4 Organização do Trabalho

Esta proposta possui 8 capítulos, sendo que no primeiro encontra-se a introdução. Os demais capítulos são descritos a seguir:

- Capítulo 2: Revisão Bibliográfica – descreve as tecnologias utilizadas no desenvolvimento do trabalho. O texto descreve a tecnologia de *Web Services*, SOA e Decomposição de serviços;
- Capítulo 3: Trabalhos Relacionados – mostra uma plataforma que fornece suporte a integração de sistemas legados da IBM, o *IBM WebSphere*, a plataforma da Microsoft, o *BizTalk Server 2006*, a plataforma da Oracle, o *Oracle SOA Suite*, e o *SAP NetWeaver*, todas as plataformas possuem objetivos semelhantes para integração de sistemas legados e envolvem o uso das tecnologias descritas no capítulo 2. Além disso, apresenta um trabalho acadêmico com suporte a *Portles* também utilizado no comparativo com este trabalho;
- Capítulo 4: SINS – apresenta a idéia do ambiente SINS, seu projeto e os modelos das interações que ocorrem no ambiente;

- Capítulo 5: Implementação – apresenta o projeto de implementação do protótipo do ambiente para sua utilização;
- Capítulo 6: Estudo de Caso – apresenta um estudo de caso da utilização do ambiente e os resultados obtidos;
- Capítulo 7: Conclusão – apresenta as conclusões e considerações finais sobre o trabalho realizado;
- Capítulo 8: Referências Bibliográficas – apresenta as referências utilizadas no decorrer deste trabalho.

2 REVISÃO BIBLIOGRÁFICA

Este capítulo oferece uma visão geral das principais tecnologias aplicadas no trabalho: *Web Services* e SOA

2.1 *Web Services*

A necessidade de conectar informações e processos mudaram a forma como o software vem sendo desenvolvido. Sistemas bem-sucedidos de (TI) exigem cada vez mais interoperabilidade entre plataformas e serviços flexíveis que possam evoluir facilmente com o tempo. Segundo o *Word Wide Web Consortium*[1] (W3C), a tecnologia de *Web Services* fornece um mecanismo padrão de interoperabilidade entre diferentes aplicações de softwares, executando em uma variedade de plataformas e/ou *frameworks* [W3C, 2007].

Os *Web Services* são aplicações modulares que podem ser descritas, publicadas e invocadas sobre uma rede, geralmente a Web. Ou seja, é uma interface que descreve uma coleção de operações que são acessíveis pela rede através de mensagens em formato XML padronizadas. Sua estrutura arquitetural permite a comunicação entre aplicações, assim, um serviço pode ser invocado remotamente, ou ser utilizado para compor um novo serviço juntamente com outros [Hansen, 2003].

Um *Web Service* pode ser visto como um componente de software independente de implementação ou plataforma, que pode ser: descrito utilizando-se uma linguagem de descrição de serviços; publicado em um registro de serviço; descoberto através de um mecanismo de busca padrão; ser invocado através de uma *Application Program Interface* (API), via rede; e ser combinado com outros serviços [Newcomer, 2002].

Segundo definição do W3C, um *Web Service* é: “uma aplicação de software identificada por um URI[2], cujas interfaces e ligações são capazes de ser definidas, descritas e descobertas como artefatos XML. Um serviço Web suporta interações diretas com outros

[1] O site oficial do W3C pode ser acessado em: <http://www.w3.org/>

[2] http://www.w3.org/Addressing

Agentes de software usando mensagens baseadas em XML, trocadas via protocolos baseados na Internet" [W3C, 2007].

Web Services combinam os melhores aspectos do desenvolvimento baseado em componentes na Web. Assim, como Componentes de Software, *Web Services* representam uma funcionalidade *black box* que pode ser reutilizada sem a preocupação com a linguagem e o ambiente utilizados em seu desenvolvimento [Crespo, 2000], [Graham et al., 2002], [Hansen, 2003]. Os *Web Services* permitem uma integração de serviços de maneira mais rápida e eficiente [Kreger, 2001].

Um *Web Service* descreve funcionalidades específicas do "negócio" com o propósito de fornecer um caminho para a utilização deste serviço. A exposição de um serviço se faz por meio de: identificar ou definir funções de valor no negócio ou processos; definir uma interface baseada no serviço para os processos; e, descrever estas interfaces em um formato baseado na Web. Para tornar um serviço disponível na Web normalmente é necessário: publicar a interface do serviço para que este possa ser encontrado e utilizado; aceitar requisições e enviar respostas usando protocolo padrão e mensagens em formato XML; e, fazer uma ligação entre requisições externas e implementações das funções dos negócios [Hansen, 2003].

Web Services são acessados por outras aplicações via protocolos de transporte como: *Hiper Text Transfer Protocol* (HTTP), *File Transfer Protocol* (FTP) e *Simple Mail Transfer Protocol* (SMTP) e não via protocolos específicos de modelos de objetos: *Distributed Component Object Model* (DCOM), *Remote Method Invocation* (RMI) ou *Internet Inter-ORB Protocol* (IIOP) [Oellermann, 2001].

2.1.1 Arquitetura de *Web Services*

A arquitetura de *Web Services* para disponibilização e acesso aos serviços está baseada nas interações de três papéis: provedor, solicitante e registro de serviço [Kreger, 2001], [Oellermann, 2001]. A figura 2.1 apresenta as interações entre esses papéis.

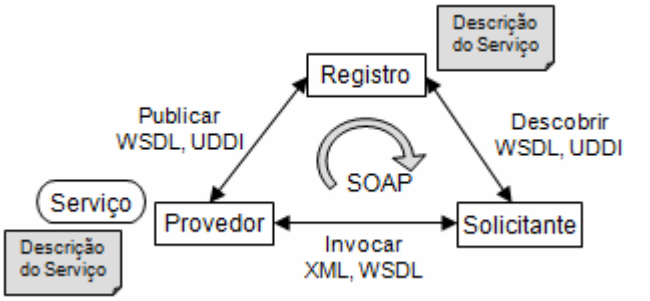

Figura 2.1 – Papéis, operações e artefatos de *Web Services* [Kreger, 2001].

O provedor de serviço é a plataforma acessada na solicitação do serviço. Trata-se da entidade que cria o *Web Service*, sendo responsável por fazer sua descrição em formato padrão e publicar os detalhes em um registro de serviço central.

O registro de serviço é o local onde os provedores publicam as descrições dos serviços. Com a descrição de serviço é possível descobrir onde está um *Web Service* e como invocá-lo. O funcionamento inicia quando o provedor cria uma descrição que detalha a interface do serviço, ou seja, suas operações e as mensagens de entrada e saída para cada operação; uma descrição de ligação é criada, apresentando como enviar cada mensagem para o endereço onde o *Web Service* está localizado.

O solicitante de serviço é uma aplicação que invoca ou inicia uma interação com um serviço. Pode ser um *browser* ou um programa sem interface com o usuário, por exemplo, outro *Web Service*. Um solicitante de serviço encontra uma descrição de serviço, ou consulta o registro de serviço para o tipo requerido, e obtém as informações de ligação da descrição do serviço durante a fase de desenvolvimento (ligação estática) ou em tempo de execução (ligação dinâmica).

A execução das interações entre os papéis ocorre via rede. As tecnologias e padrões utilizados no desenvolvimento de um *Web Service* podem ser representados conforme a figura 2.2, traduzindo esse cenário para um conjunto de camadas conceituais descritas a seguir [Kreger, 2001], [Rheinheimer, 2004]:

- Camada de rede: camada base que abrange os protocolos de transporte como: HTTP, FTP e SMTP, podendo ser utilizada para implementação de necessidades das aplicações, tais como: disponibilidade, desempenho, segurança e confiabilidade;

- Mensagem: as mensagens têm como base a tecnologia XML e o protocolo SOAP para realizar a troca de mensagens entre provedor, solicitante e registro de serviço;
- Descrição do serviço: a descrição do serviço é feita com uma linguagem específica denominada WSDL, que define uma interface e mecanismos de interação dos serviços. Também define descrições adicionais como contexto, qualidade do serviço e o relacionamento de serviço para serviço;
- Publicação e descoberta do serviço: essas camadas utilizam o registro UDDI para fazer a descoberta e a publicação de informações sobre *Web Services*.

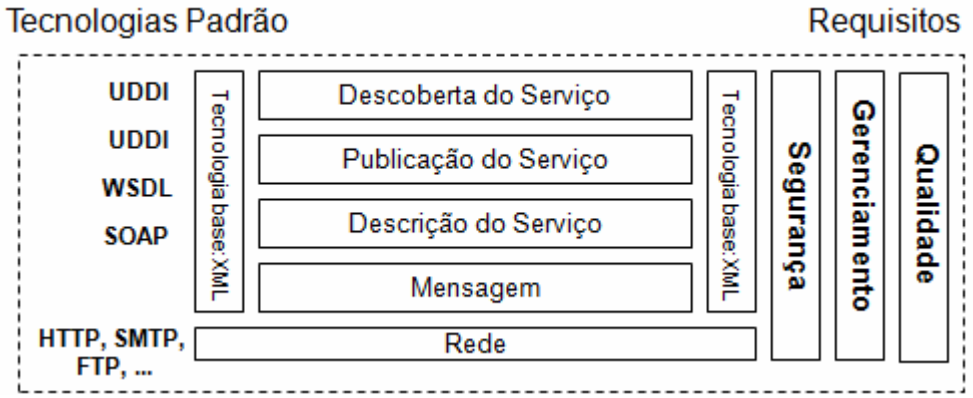

Figura 2.2 – Camadas conceituais de *Web Services* [Kreger, 2001].

2.1.2 Tecnologias padrão utilizadas na arquitetura de *Web Services*

A seguir, serão descritas as tecnologias padrão utilizadas na construção de *Web Services*: WSDL, SOAP e UDDI. Essas tecnologias são baseadas em XML e permitem invocar um serviço sem a necessidade de conhecer a plataforma ou linguagem de programação usada na sua construção.

2.1.2.1 XML

Extensible Markup Language (XML) é uma das tecnologias-chave para a construção e utilização de *Web Services* [Roy and Ramanujan, 2001]. XML foi criada em 1996 pelo W3C, liderado por Jon Bosak, da Sun Microsystems, sendo um subconjunto da *Standard General Markup Language* (SGML). SGML é um padrão complexo para descrever a estrutura do conteúdo de documentos [Sampaio, 2006].

XML é uma linguagem para organização dos dados de um documento, em formato textual, que contém marcadores como um arquivo *Hypertext Markup Language* (HTML). Esses marcadores são definidos através de uma linguagem onde sua sintaxe é baseada em marcas (*tags*). As linguagens de marca são bem simples e de fácil entendimento. Para uso das *tags* não existe um padrão definido, assim, qualquer pessoa pode definir conforme a necessidade de uso [Sampaio, 2006]. A figura 2.3 apresenta um exemplo da estrutura básica de um documento XML.

```
<?xml version=”1.0”?>
<curso tipo=”programacao”>
     <nome>Introducao do XML</nome>
     <descricao>Introducao a linguagem XML</descricao>
     <carga>60 horas</carga>
</curso>
```

Figura 2.3 – Estrutura básica de um documento XML.

No exemplo da figura 2.3, se o texto for editado em um bloco de notas e salvo com extensão .XML, o arquivo poderá ser aberto e visualizado em um *browser*, por exemplo, *Internet Explorer*. Os arquivos XML são aceitos por programas capazes de processar XML.

2.1.2.2 WSDL

A *Web Service Description Language*[3] (WSDL) é uma linguagem em formato XML para descrição de interface dos serviços, de forma que outros programas possam interagir com esses serviços. WSDL descreve um serviço como uma coleção de operações que podem ser acessadas através de mensagens. Para sua utilização com *Web Services* deve existir um arquivo WSDL, escrito em XML, para cada *Web Service*, cuja função é descrever as operações que o *Web Service* realiza [Hansen, 2003], [Newcomer, 2002].

A WSDL esta dividida em três elementos principais: definições de tipo de dados, operações abstratas e protocolos de ligação. Cada um desses elementos podem ser especificados em documentos XML diferentes e importados em diferentes combinações, para

[3] http://www.w3.org/TR/2004/WD-wsdl20-20040326/

criar a descrição final de um *Web Service*, ou definidos juntos em um único arquivo XML. A definição do tipo de dados determina a estrutura e o conteúdo das mensagens. As operações abstratas determinam as operações possíveis e o protocolo de ligação determina a forma de transmissão das mensagens pela rede até os destinatários [Newcomer, 2002].

A descrição de um serviço consiste de duas partes: definição da implementação do serviço e definição da interface do serviço [Hansen, 2003]. A figura 2.4 apresenta essas definições.

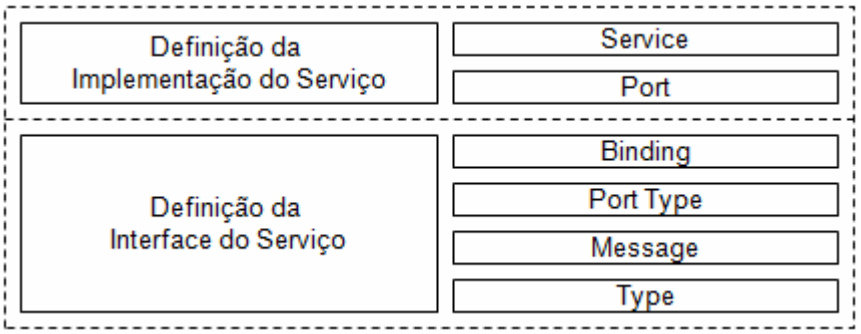

Figura 2.4 – Camada de descrição dos serviços [Hansen, 2003].

A camada de definição da interface do serviço contém a definição de serviço WSDL, permitindo que uma interface possa ser utilizada, instanciada e referenciada por múltiplas definições de implementação de serviços, incluindo diretivas:

- *wsdl:binding* - descreve protocolos, formato de dados, segurança e outros atributos para uma interface (*portType*) em particular;
- *wsdl:portType* - informa elementos de operações do *Web Service*;
- *wsdl:message* - define entrada e saída de dados referentes a operações. Pode assumir a forma de um documento inteiro ou de argumentos que devem ser mapeados para invocação de métodos;
- *wsdl:type* - define tipos de dados complexos em uma mensagem.

A camada de definição de implementação do serviço descreve como uma interface de serviço é implementada por um provedor, ou seja, onde o serviço está instalado e como pode ser acessado. A definição de um serviço (*WSDL:service*) contém uma coleção de elementos *WSDL:port* com um elemento *WSDL:binding*. Um arquivo de implementação descreve onde o *Web Service* está instalado e como é acessado e, além disso, a WSDL especifica extensões

para ligações com protocolos e formatos de mensagem como SOAP, HTTP GET/POST e *Multipurpose Internet Mail Extensions* (MIME).

O trecho a seguir define um exemplo simplificado de uma descrição WSDL.

```
<message name="getMethodSearchRequest">
     <part name="name" type="xsd:string"/>
</message>
<message name="getMethodSearchResponse">
     <part name="value" type="xsd:string"/>
</message>
<portType name="methodSearchs">
     <operation name="getMethodSearch">
          <input message="tns:getMethodSearchRequest"/>
          <output message="tns:getMethodSearchResponse"/>
     </operation>
</portType>
```

Figura 2.5 – Exemplo de documento WSDL.

Nos elementos *message* estão definidas as partes de uma mensagem e os tipos de dados associados, ou seja, os elementos de dados de uma operação. No exemplo, a definição de uma mensagem nomeada *getMethodSearchRequest* possui um elemento chamado *name*, do tipo *string*. Outra mensagem nomeada *getMethodSearchResponse*, possui um elemento chamado *value*, do tipo *string*. Comparando com a programação tradicional, é o mesmo que ter uma função *getMethodSearchRequest* com o parâmetro *name* e outra função *getMethodSearchResponse* com o parâmetro *value*.

O elemento *portType* define o *Web Service*, as operações que podem ser realizadas e as mensagens que estão envolvidas. No exemplo, *methodSearchs* é definido como um elemento de operação do *Web Service*, onde *getMethodSearch* é o nome da operação. A operação *getMethodSearch* tem uma mensagem de entrada chamada *getMethodSearchRequest* e uma mensagem de saída chamada *getMethodSearchResponse*. Fazendo uma comparação com a programação tradicional, *methodSearchs* seria uma biblioteca de funções, e a operação *getMethodSearch* seria uma função com *getMethodSearchRequest* como parâmetro de entrada e *getMethodSearchResponse* como parâmetro de retorno.

2.1.2.3 SOAP

O protocolo *Simple Object Access Protocol*[4] (SOAP) permite a comunicação entre diversas aplicações em um ambiente distribuído e descentralizado. A comunicação é realizada através de trocas de mensagens, transmitidas em formato XML, incluindo parâmetros usados na chamada, bem como os dados de resultados. Isto significa que as mensagens podem ser utilizadas e compreendidas por quase todas as plataformas de *hardware*, sistemas operacionais, linguagens de programação e equipamentos de rede [Hansen, 2003].

Um pacote SOAP consiste de quatro partes [Newcomer, 2002], [Hansen, 2003]:

- Envelope: guarda o conteúdo da mensagem, quem poderá tratá-la e se o tratamento é opcional ou obrigatório. A estrutura de mensagem SOAP encapsula os elementos sintáticos da mensagem. O envelope contém os seguintes subelementos: cabeçalho – contém os atributos opcionais das mensagens; corpo - contém os dados da mensagem em XML;
- Codificação: responsável por definir mecanismos de serialização que podem ser utilizados para trocar instâncias ou tipos de dados definidos por uma aplicação;
- *Remote Procedure Call*[5] (RPC): especifica como encapsular chamadas remotas de métodos e respostas dentro da mensagem;
- *Framework* de ligação e transporte: define um *framework* abstrato para troca de envelopes SOAP entre aplicações utilizando um protocolo de transporte simples.

A figura 2.6 apresenta as três partes principais das mensagens SOAP: envelope, cabeçalho e corpo. O envelope é obrigatório e marca o início e o fim das mensagens. O cabeçalho é opcional e pode conter um ou mais blocos com atributos da mensagem. O corpo também é obrigatório e contém um ou mais blocos contendo a mensagem propriamente dita [Newcomer, 2002].

[4] http://www.w3.org/TR/soap/

[5] http://www.faqs.org/rfcs/rfc1050.html

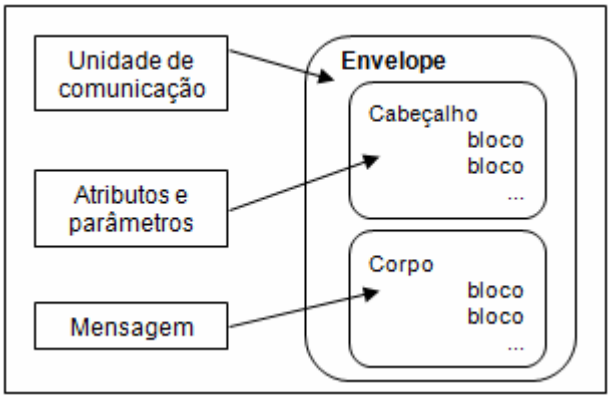

Figura 2.6 – Estrutura do envelope SOAP [Newcomer, 2002].

Quando um *Web Services* é implementado, os dados contidos nas mensagens SOAP devem ser interpretados, já que o SOAP não define o serviço, apenas contém dados suficientes para que o processador SOAP possa reconhecê-lo. O SOAP é definido com um nível de abstração suficiente para abranger tanto documentos como interações RPC [Newcomer, 2002].

A invocação do serviço ocorre conforme apresentado na figura 2.7.

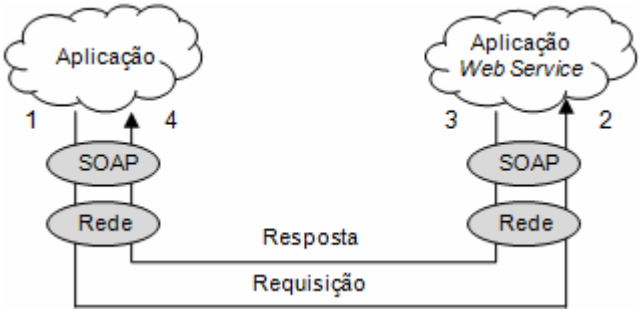

Figura 2.7 – Invocação do serviço utilizando SOAP [Kreger, 2001].

A aplicação (1) requisita uma mensagem SOAP e invoca a operação do serviço através de um provedor de *Web Services*. O solicitante de serviço apresenta a mensagem junto com o endereço de rede do provedor de *Web Service*. A infra-estrutura de rede (2) entrega a mensagem para um servidor SOAP. O servidor SOAP redireciona a mensagem requisitada para o provedor de serviço *Web Service*. O servidor Web (3) é responsável por processar uma mensagem de requisição e formular a resposta. Quando a mensagem XML chega ao nodo requisitante, é convertida para uma linguagem de programação, sendo entregue para a aplicação (4) [Kreger, 2001].

2.1.2.4 UDDI

A especificação *Universal Description, Discovery and Integration*[6] (UDDI) tem como objetivo criar um padrão para a descoberta de serviços. Com UDDI é possível localizar um serviço, constituindo de uma especificação técnica para descrever, descobrir e integrar *Web Services* [Newcomer, 2002].

O UDDI é constituído de duas partes: o UDDI *Project* que é uma especificação técnica utilizada para construir e distribuir *Web Services*, a qual permite que as informações sejam armazenadas em um formato XML especifico; e o UDDI *Business Registry*, que é uma implementação operacional completa da especificação UDDI [Hansen, 2003], [Rheinheimer, 2004].

O UDDI *Project* é considerado o componente central que manipula o registro global e público UDDI *Business Registry*. Toda informação mantida no UDDI *Business Registry* está disponível para consultas em geral.

A informação oferecida pelo UDDI *Business Registry* consiste de três componentes [Rheinheimer, 2004]:

- *white pages*: são incluídas informações gerais sobre a empresa específica, tais como: endereço, contato e identificadores conhecidos;
- *yellow pages*: são incluídos dados de classificação gerais da empresa ou serviço oferecido, tais como: categorização industrial;
- *green pages*: são incluídas informações técnicas sobre *Web Services*.

A implementação UDDI é um servidor de registro que fornece um mecanismo para publicar e descobrir *Web Services*. O registro UDDI contém informações categorizadas sobre empresas e serviços que elas oferecem, e associações destes serviços com especificações dos *Web Services* podem ser feitas em WSDL através do próprio registro [Hansen, 2003], [Rheinheimer, 2004].

---

[6] http://www.oasis-open.org/committees/uddi-spec/doc/tcspecs.htm

O acesso público do registro pode ser efetuado via Internet e o acesso privado pode ser efetuado em Intranets de empresas, por exemplo. Um registro UDDI pode também ser acessado por aplicações, via código ou através de alguma interface.

O modelo de informação principal utilizado pelo registro UDDI é definido através de *XML Schema*, definindo quatro tipos de informações: negócio, serviço, ligação e a especificação do serviço [Hansen, 2003], [Rheinheimer, 2004].
A informação referente ao registro de um serviço consiste de cinco tipos de estruturas de dados [Newcomer, 2002], [Hansen, 2003], [Rheinheimer, 2004]. Esta divisão por tipos de informação fornece partições simples para auxiliar na rápida localização e compreensão das diferentes informações que compõem o registro. A figura 2.8 apresenta as cinco estruturas.

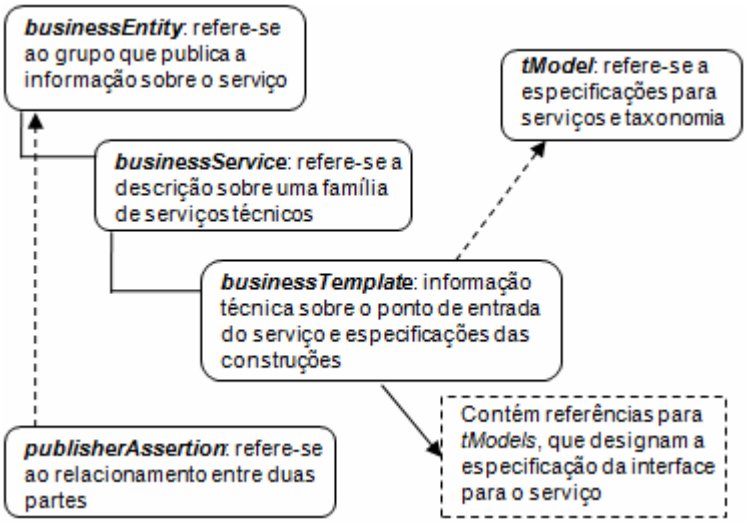

Figura 2.8 – Modelo de Estrutura UDDI [Hansen, 2003].

O *businessEntity* fornece informação sobre uma empresa tais como: nome, descrição e endereço, podendo conter um ou mais *businessServices*. Representa toda a informação conhecida sobre uma empresa específica ou informações descritivas sobre uma entidade, bem como os serviços que ela fornece.

O *businessService* define descrições técnicas e de negócios tais como: nome e descrição para um ou grupo de *Web Services* relacionados. Representa uma classificação lógica de serviço. Cada estrutura *businessService* pertence a uma única estrutura *businessEntity.*

O *bindingTemplate* define descrições técnicas de como e onde acessar um *Web Service* específico. Fornecem suporte para que se possam acessar os serviços remotamente, e definem o suporte a tecnologias, parâmetros específicos da aplicação e os arquivos de configuração.

A estrutura *tModel* é representada por meio de metadados (dados sobre dados). Seu propósito é fornecer um sistema de referência para os documentos WSDL.

O *publisherAssertion* permite que se possa associar estruturas *businessEntity,* de forma a obter uma melhor identificação. Ambas as estruturas devem publicar exatamente a mesma informação, para que o relacionamento torne-se visível.

Cabe salientar que as informações contidas em arquivos de descrição de serviço (WSDL) complementam aquelas que estão no registro. No entanto, UDDI fornece suporte a vários tipos de descrição de serviço, mas não suporta a criação de descrições WSDL de forma direta.

Uma descrição WSDL completa consiste da: combinação dos documentos de interface, que pode ser publicada no registro usando um *tModel*, o que deve ser feito antes da implementação ser publicada como *businessService;* e, da implementação do serviço, que é publicada no registro UDDI como um *businessService* ou dentro de um *bindingTemplate* (ou mais de um) [Hansen, 2003], [Rheinheimer, 2004].

2.2 *Service Oriented Architecture* (SOA)

Um caminho para constituir um sistema é considerá-lo uma composição de um conjunto de interações de serviços, assim, cada serviço fornece acesso a um conjunto de funcionalidades bem-definidas. O sistema como um todo é projetado e implementado, e interações entre os serviços são mantidas. Expor uma funcionalidade como um serviço é a chave para a flexibilidade [Brown et al., 2002].

Como os sistemas de informações têm crescido exponencialmente, as empresas têm construído arquiteturas de softwares cada vez mais complexas, onde precisam responder rapidamente as novas exigências de negócios, reduzirem continuamente os custos de TI, e ao mesmo tempo absorver e integrar novos negócios de parceiros e clientes. Arquiteturas

tradicionais alcançaram o limite de suas capacidades e assim, uma nova arquitetura se faz necessária para maximizar o reuso e flexibilidade nos negócios [IBM, 2005].

SOA relaciona serviços e seus consumidores, que representam um processo de negócio. Os serviços podem ser acessados pelo nome via uma interface, e os consumidores acessam os serviços disponíveis via interface de serviço, por exemplo, *Web Services*. Essa nova arquitetura está substituindo aos poucos as arquiteturas monolíticas, por exemplo, princípios de projeto, para novas aplicações de negócios [Gartner, 2003].

As definições apresentadas na próxima seção apresentam abordagens de diferentes autores sobre a arquitetura SOA.

2.2.1 Definindo SOA

Algumas definições de SOA são encontradas na literatura:

- "SOA é um novo paradigma de desenvolvimento de aplicações cujo objetivo é criar módulos funcionais chamados de serviços, com baixo acoplamento e permitindo a reutilização de código" [Sampaio, 2006].
- "SOA é uma forma de tecnologia arquitetural que adere aos princípios da orientação a serviços. Quando realizada através da plataforma de tecnologia de *Web Services*, SOA estabelece um potencial para suportar e promover estes princípios durante todo o processo de negócios e domínio de automação de uma empresa" [Erl, 2006].
- O W3C define SOA como: "uma forma de arquitetura de sistemas distribuídos" [W3C, 2007]. Estes sistemas, por sua vez, são caracterizados pelas seguintes propriedades:
    - Visão lógica: o serviço é uma visão abstrata e lógica dos programas reais, banco de dados, processos de negócio, etc., definidos em termos do que faz, realizando tipicamente uma operação em nível de negócios;
    - Orientação por mensagens: o serviço é formalmente definido em termos de trocas de mensagens entre provedor e solicitante, e não nas propriedades deles, já que são abstratas na implementação de SOA;

- Orientação por descrição: o serviço é descrito por um metadado. A descrição suporta a natureza pública de um SOA: somente aqueles detalhes que são expostos publicamente e importantes para o uso do serviço devem ser incluídos na descrição; A semântica de um serviço deve ser documentada, diretamente ou indiretamente, por sua descrição;
- Granularidade: serviços tendem a usar um pequeno número de operações como mensagens relativamente grandes e complexas;
- Orientação por rede: serviços tendem a ser orientados para uso em rede, embora isto não seja um requisito absoluto;
- Plataforma neutra: mensagens são enviadas em uma plataforma neutra, em formato padronizado, por exemplo, XML, e entregue através de interfaces.

A figura 2.9 apresenta um modelo de arquitetura do W3C que encapsula diferentes conceitos, como: política, mensagens, recursos e ações de forma a traduzir um modelo geral de arquitetura SOA [W3C, 2007].

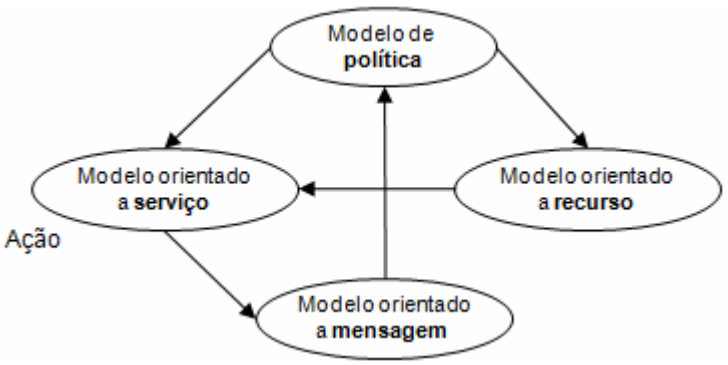

Figura 2.9 – Modelo W3C SOA [W3C, 2007].

Os quatros modelos da arquitetura são:

- Modelo orientado a mensagem: define a mensagem em termos de conteúdo (cabeçalho e corpo), transporte de entrega, solicitante e provedor do serviço;
- Modelo orientado a recurso: define recursos em termos de endereço (URI), representação e dono do recurso;
- Modelo de políticas: define a política em termos de recursos, aplicada também para descrições dos serviços;

- Modelo orientado a serviço: mais complexo de todos. Um serviço é oferecido e utilizado, sendo mediado por meio de trocas de mensagens.

2.2.2 Tecnologias que fundamentam a arquitetura SOA

Esta seção apresenta uma abordagem de como tecnologias como XML e *Web Services* estão sendo usadas para embasar a arquitetura SOA e fundamentá-la de tal forma que possa ser aplicada.

2.2.2.1 XML

Através do uso de XML, torna-se possível unir significado e contexto para alguma parte da informação transmitida por meio dos protocolos da Internet [Newcomer, 2002].

A arquitetura de representação dos dados em XML estabelece uma camada principal para construção de uma SOA. Nesta camada, o XML estabelece o formato e a estrutura das mensagens que navegam por todos os serviços. Qualquer movimentação dentro de SOA não pode ser feita sem envolver XML [Rogers and Hendrick, 2005].

2.2.2.2 *Web Services* e SOA

O conceito de criar um *framework* de comunicação padrão, baseado na Web com tecnologia distribuída, para servir de ponte a enorme disparidade que existe entre e dentro das organizações, recebeu o nome de *Web Services*. Para dar suporte ao uso de *Web Services* surgiram os conceitos de WSDL, SOAP e UDDI, que completaram a primeira geração da família padrão para *Web Services* [Newcomer, 2002].

Web Services foi a base da criação de uma plataforma arquitetural separada, que alavancou um conjunto de benefícios desta tecnologia para realizar o conceito de serviços nas empresas. Assim, SOA ganhou popularidade com o conceito de serviços.

Componentes SOA podem ser criados usando qualquer tecnologia. Um modelo, inspirado pelo conjunto de padrões de *Web Services*, definiu SOA como uma arquitetura modelada em torno de três componentes básicos: provedor, solicitante e registro de serviço. Este modelo serviu apenas para ilustrar o conceito inicialmente formado para SOA, de uma

perspectiva de arquitetura física, a variação de um SOA baseado em *Web Services* vai além desta primeira definição [Keen et al., 2004].

Com a evolução de XML e *Web Services*, SOA evoluiu também. Isto é resultado da relação entre numerosas iniciativas dirigidas por uma variedade de organizações de padrões e desenvolvimento de software. Os padrões de *Web Services* continuam a serem adotados em grande número e com isso fornecem suporte ao uso na arquitetura SOA [Rogers and Hendrick, 2005].

2.2.3 Fundamentos de SOA

O termo "orientado a serviços" existe há algum tempo e tem sido usado em diferentes contextos e para diferentes propósitos, por exemplo, separação de interesses, que significa decompor um sistema em um conjunto de partes menores, onde cada uma destas partes atenda um interesse ou uma parte específica de um problema [Sehmi, 2006].

Quando acoplado com "arquitetura", orientado a serviços leva uma conotação técnica. "SOA" é um termo que representa um modelo para organização e utilização de lógica de negócios distribuída que estão sob o controle de diferentes domínios proprietários. Os termos visibilidade, interação e efeito são chaves na arquitetura SOA. A visibilidade introduz a possibilidade de compartilhar necessidades, a interação é a atividade que usa a lógica de negócios, e o efeito é o que o uso da lógica de negócios promove no mundo real. O mecanismo pelo qual as necessidades e a lógica de negócios são colocadas juntas é chamado de serviço [OASIS, 2006].

Sampaio define um serviço como sendo um "componente que atende a uma função de negócio específica para os clientes. O serviço recebe requisições e as responde ocultando todo o detalhamento do seu processamento" [Sampaio, 2006].

Um serviço pode executar unidades completas de trabalho, não dependendo do estado de outros componentes externos, aumentando assim a sua reutilização, ou seja, um serviço executa uma função atômica (ou transação). Todas as etapas intermediárias devem ser gerenciadas apenas pelo serviço e não pelo solicitante. Serviços podem ser criados usando qualquer linguagem, tecnologia ou plataforma. Os itens a seguir apresentam exemplos de serviços [Sampaio, 2006]:

- Verificar a disponibilidade de vôos para uma determinada cidade;
- Efetuar a venda de um determinado produto;
- Reservar hotel para um cliente.

2.2.3.1 Princípios comuns da orientação a serviços

Um projeto de *Web Service* para SOA difere de outros *Web Services* criados para uso em outros ambientes de aplicações distribuídas porque seguem um conjunto de convenções. Existe um conjunto comum de princípios associados com orientação a serviços que estabelecem uma abordagem única de projeto para construir *Web Services* para SOA. Quando aplicados, estes princípios levam a uma padronização de *Web Services*, ao mesmo tempo que preservam o baixo acoplamento das relações entre eles. Erl [Erl 2006] apresenta uma lista destes princípios:

- Serviços são reusáveis: serviços são projetados para suportar potencial reuso;
- Serviços compartilham um contrato formal: para os serviços interagirem, não necessitam compartilhar algo, mas um contrato formal que descreve cada serviço e define os termos das informações trocadas, como operações e mensagens;
- Serviços são fracamente acoplados: serviços devem ser projetados para interagir de forma independente e resistente a mudanças;
- Serviços devem abstrair a lógica de negócios: somente o que está exposto via contrato de serviço é que está visível para o solicitante do serviço, a lógica de negócio é invisível e irrelevante;
- Serviços são passíveis de composição: serviços podem compor outros serviços. Esta lógica pode ser representada por diferentes níveis de granularidade e promover reusabilidade e criação de camadas de abstração;
- Serviços são autônomos: a lógica governada por um serviço reside dentro de um bloco explícito. O serviço tem controle dentro deste bloco e não está dependente de outros serviços para executar esta governança;
- Serviços são *stateless*: serviços não devem ser utilizados para gerenciar o estado das informações;

- Serviços são passíveis de descoberta: serviços levam suas descrições para serem descobertos e entendidos por solicitantes de serviços que podem estar habilitados para fazer uso da sua lógica de negócios.

A tecnologia de *Web Services* dá suporte a alguns destes princípios descritos anteriormente, porém, foram identificados quatro princípios que não são providos por esta tecnologia, que são: serviços são reusáveis; serviços são autônomos; serviços são *stateless*; serviços são passíveis de descoberta. A tabela 2.1 apresenta uma descrição que compara cada um destes princípios com a tecnologia de *Web Services* [Keen et al., 2004], [Erl, 2006].

Tabela 2.1 – Princípios da orientação a serviço suportados por *Web Services* [Erl, 2006].

Princípios da orientação a serviços	**Suporte *Web Services***
Serviços reusáveis	*Web Services* não são automaticamente reusáveis.
Serviços compartilham contrato formal	*Web Services* requerem o uso de descrições de serviços, tornando os contratos de serviço parte da sua comunicação.
Serviços são fracamente acoplados	*Web Services* são naturalmente fracamente acoplados através do uso das descrições dos serviços.
Serviços devem abstrair lógica de negócios	*Web Services* automaticamente emula o modelo de caixa preta dentro do framework de comunicação de *Web Services*, escondendo os detalhes da lógica de negócios.
Serviços são passíveis de composição	*Web Services* são naturalmente compostos.
Serviços são autônomos	*Web Services* não são autônomos. Isso requer muito esforço no projeto.
Serviços são *stateless*	*Stateless* é a condição preferida para *Web Services*.
Serviços são passíveis de descoberta	Não é suportado por *Web Services*. Deve ser implementada por uma arquitetura e pode ser considerada uma extensão para infra-estrutura de TI.

Abstração, possibilidade de composição, baixo acoplamento e a necessidade por contratos de serviços são características nativas de *Web Services* que estão em total alinhamento com os princípios correspondentes da orientação a serviços. Reusabilidade, autonomia, *stateless* e descoberta não são automaticamente providos por *Web Services*. Realizar estas tarefas requer esforço em relação ao modelo e ao projeto.

2.2.4 Componentes de uma arquitetura SOA

SOA é um ambiente padronizado para os princípios da orientação a serviço. Neste ambiente é definido um conjunto de serviços que completam os objetivos e processos de negócios de uma organização [Manolescu and Lublinsky, 2007a]. A seguir são descritos os principais componentes que fazem parte da arquitetura SOA:

- mensagens (unidades de comunicação): representa o dado solicitado para completar alguma ou toda parte de uma requisição;
- operações (unidades de trabalho): representa a lógica solicitada para processar mensagens a fim de completar uma requisição;
- serviços (unidades de processamento lógico): representa um conjunto de operações logicamente agrupadas capazes de executar as requisições relacionadas;
- processos (unidades de automação lógica): contém as regras de negócio que determinam quais operações de serviços são usadas para completar uma requisição.

Os componentes da arquitetura SOA se relacionam da seguinte forma [Manolescu and Lublinsky, 2007a]:

- Uma operação envia e recebe mensagens para executar um trabalho;
- Uma operação é geralmente definida pelas mensagens que ela processa;
- Um serviço agrupa uma coleção de operações relacionadas;
- Um serviço é geralmente definido pelas operações que se relacionam com ele;
- Uma instância de processo pode compor serviços;
- Uma instância de processo não é definida necessariamente por serviços porque pode somente solicitar um subconjunto de funcionalidades oferecidas pelos serviços;

- Uma instância de processo invoca uma única série de operações para completar a automação;
- Cada instância de processo está parcialmente definida pelas operações de serviço que utiliza.

A figura 2.10 apresenta estes relacionamentos. As instâncias de processo executam uma série de operações, podendo também defini-las, e compõem um conjunto de serviços. Os serviços agrupam logicamente um conjunto de operações, e essas definem os serviços. As operações enviam e recebem mensagens, e essas definem as operações relacionadas.

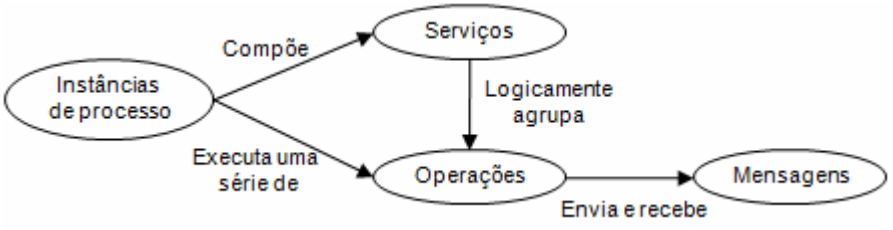

Figura 2.10 – Componentes de SOA relacionados [Manolescu and Lublinsky, 2007a].

2.2.5 Modelo de uma arquitetura SOA

Um modelo de software implementado na arquitetura SOA possui as seguintes características que são diferentes das arquiteturas de software tradicionais [Chen et al., 2006]:

- Interoperabilidade baseada em padrões: baseia-se em protocolos, comunicação, coordenação, *workflow*, descoberta, colaboração e publicação através de protocolos padrões tais como: XML, SOAP, WSDL, UDDI, HTTP, entre outros. Estes padrões permitem que os serviços desenvolvidos em diferentes plataformas possam interoperar uns com os outros conhecendo-se apenas a especificação do serviço;
- Composição dinâmica via descoberta: fornece uma nova forma de desenvolver aplicações baseadas em descoberta de serviços. Além disso, a composição e descoberta podem ser realizadas em tempo de execução;
- Governança e orquestração dinâmica: fornece mecanismos para controle da execução dos serviços. Um deles é um serviço de governança por políticas, ou seja, políticas podem ser especificadas, verificadas e reforçadas durante a fase de desenvolvimento e

em tempo de execução. Outro mecanismo, chamado de orquestração, coordena a execução de um processo e é responsável por programar a execução de um serviço.

Com a introdução do conceito de serviços é estabelecida uma forma de abstração entre os processos de negócios e as aplicações já desenvolvidas nas organizações. A figura 2.11 apresenta um modelo de camadas de uma arquitetura SOA, que atende as características mencionadas anteriormente, apresentando onde cada uma delas está representada.

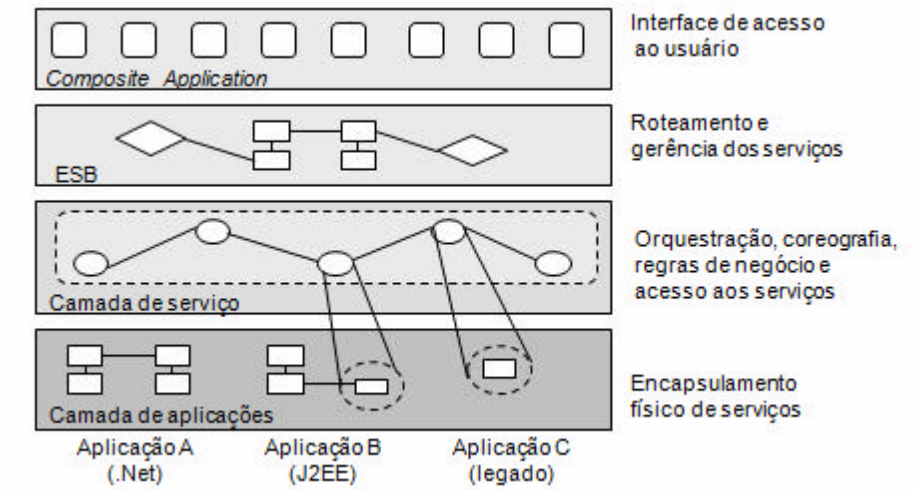

Figura 2.11 – Camadas da arquitetura SOA [Erl, 2006].

Na figura 2.11, as aplicações A, B e C, pertencentes a uma camada de aplicações, distribuem seus dados como serviço para a camada de serviços, assim, a conexão com essas aplicações deixa de existir, restando apenas os serviços gerados a partir delas. A camada de serviços é responsável por gerenciar os processos de negócios, orquestrar e disponibilizar os serviços para o uso. A camada ESB efetua o roteamento e a gerência dos serviços disponibilizados e a camada de *composite application* fornece uma interface de acesso ao usuário para utilização do serviço requisitado.

2.2.5.1 Camada de serviço

No modelo de camadas apresentados na figura 2.11 são identificados quatro abstrações que compõem a camada de serviços, que são [Erl, 2006], [Chen et al., 2006]:

- Acesso aos serviços: são responsáveis por representar a tecnologia e a lógica de uma aplicação;

- Regras de negócios: são responsáveis por expressar a lógica do negócio através da orientação a serviços e trazer a representação de modelos de negócios corporativos para *Web Services*;
- Coreografia: pode ser vista como um processo que age de forma a permitir a colaboração entre diferentes orquestrações, ou ainda, podendo consistir de múltiplos participantes que podem assumir diferentes papéis e que possuem relacionamentos diferentes;
- Orquestração: representa o processo pelos quais diferentes serviços são invocados. Os serviços podem ser organizados em diferentes formas ou apenas reagrupados em outros fluxos. A orquestração é composta por um fluxo de etapas, e um coordenador responsável pelo andamento no fluxo.

A figura 2.12 apresenta o processo de orquestração de serviços.

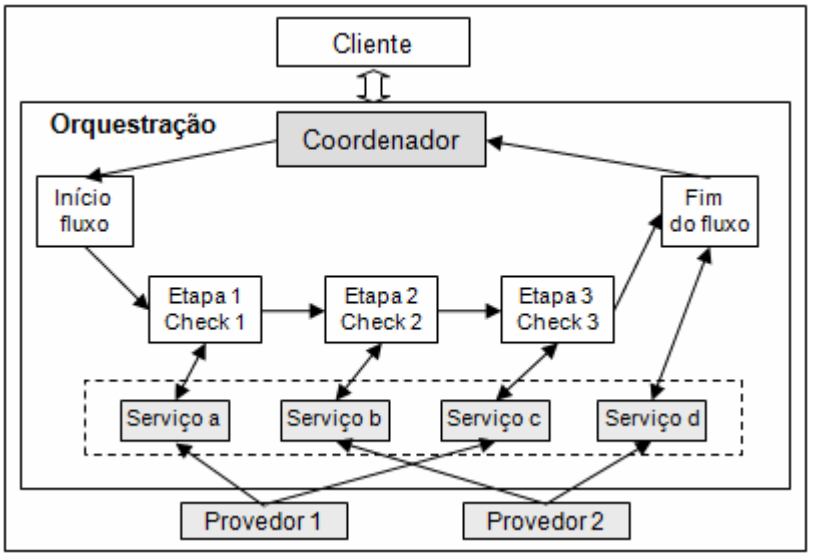

Figura 2.12 – Orquestração de serviços [Sampaio, 2006].

Na figura 2.12 é apresentado um esquema de orquestração de quatro serviços, fornecidos por dois provedores de serviços diferentes. Neste processo, o cliente se comunica com o coordenador e efetua uma solicitação. O coordenador inicia o fluxo, invocando e verificando todas as etapas necessárias. Cada etapa invoca um serviço, que é fornecido por um provedor de serviço. Desta maneira, é possível mudar a ordem das etapas, acrescentar outras, mudar os critérios de verificação ou criar outros fluxos sem alterar o código dos serviços.

A orquestração tem sido chamada de “o núcleo de SOA”, estabelecendo um significado de centralização e controle das lógicas de negócios através de um modelo de serviço padronizado. A orquestração comanda um processo baseado em serviços e pode acessar diversas aplicações, implementar regras de negócios complexas e interagir com diferentes processos de negócios [Chappell, 2004].

Com a orquestração diferentes processos podem ser conectados redesenvolvendo as soluções originais. A orquestração introduz novas lógicas de fluxos de dados, assim, seu uso pode reduzir a complexidade das soluções estabelecidas. A lógica de fluxo de dados é abstraída e facilmente mantida [Chappell, 2004].

A camada de orquestração de serviços consiste de um ou mais serviços que compõem processos de negócios de acordo com a lógica de negócios. A orquestração abstrai as regras de negócios e a seqüência lógica de execução dos serviços, promovendo agilidade e reusabilidade.

2.2.5.2 *Enterprise Service Bus* (ESB)

De acordo com a IBM, um ESB ajuda a maximizar a flexibilidade de SOA. Os participantes de uma interação entre serviços são conectados ao ESB, e não um diretamente ao outro. Quando o serviço solicitante conecta no ESB, este assume a responsabilidade de entregar suas requisições, através do uso de mensagens, para o provedor de serviço que oferece a função solicitada [IBM, 2006].

O ESB facilita a comunicação entre solicitantes e provedores de serviços, resolvendo diferenças de protocolos, padrões de interação ou capacidade dos serviços. Também pode prover ou melhorar o monitoramento e gerenciamento sobre os serviços. O ESB provê virtualização e funcionalidades de gerenciamento que implementam e estendem as capacidades principais do SOA [IBM, 2006]. O ESB virtualiza a:

- Localização e Identidade: participantes não precisam saber a localização ou identidade de outros participantes. Por exemplo, um serviço solicitante não precisa saber quais serviços podem atender sua requisição. Provedores de serviços podem ser adicionados ou removidos a qualquer momento sem problemas;

- Protocolo de Interação: participantes não precisam usar o mesmo protocolo de comunicação ou modo de interação. Uma requisição feita por SOAP sob HTTP pode ser atendida por um serviço que apenas opere em SOAP sob JMS (*Java Message Service*);
- Interface: solicitantes e provedores não precisam utilizar uma interface comum. Um ESB concilia as diferenças transformando as requisições e respostas na forma esperada pelo serviço que irá recebê-las;
- Qualidade de serviço (QoS): participantes ou administradores de sistema especificam os requisitos de QoS, incluindo requisições de autorização, encriptação e decriptação de dados, auditoria de serviços e modo de roteamento de serviços (otimizando para velocidade ou custo, por exemplo).

A figura 2.13 apresenta como o ESB provê conectividade para estender diferentes pontos de SOA. Ambos os provedores e solicitantes de serviços conectam-se ao ESB para que a comunicação aconteça, conforme suas requisições.

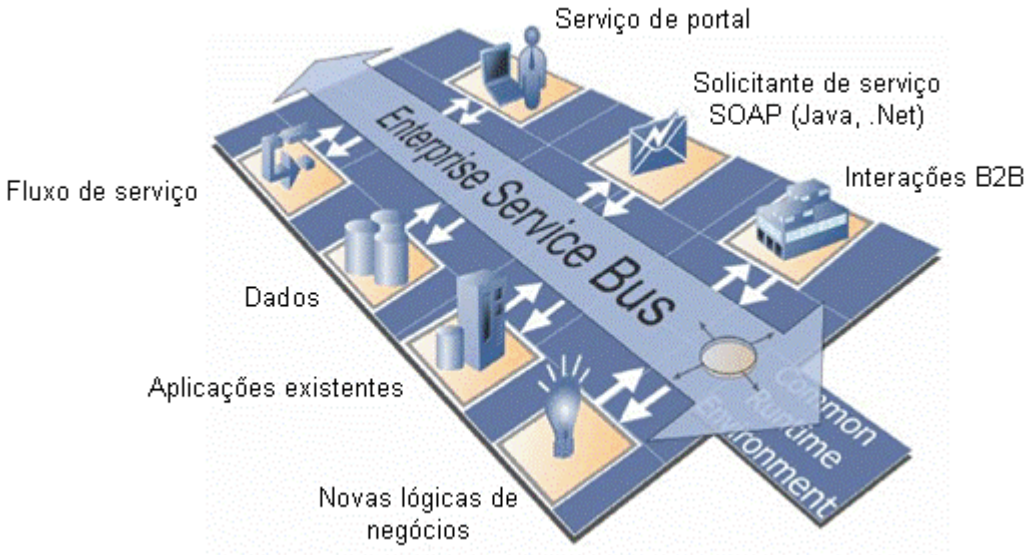

Figura 2.13 – Visão de infra-estrutura de um ESB [Keen et al., 2004].

2.2.5.3 *Composite Applications*

A globalização requer que as pessoas trabalhem de um modo mais colaborativo do que antes, tornando-se necessário, um intercâmbio entre as ferramentas utilizadas para adquirir

compreensão, colaboração e que ajude na tomada de decisões. Hoje em dia, a maioria das aplicações de negócios são eficazes na automatização de transações, mas não permitem uma colaboração entre os limites funcionais. Isto usualmente leva pessoas a utilizarem ferramentas de produtividade para uso pessoal, por exemplo, um editor de texto, para realizarem suas tarefas. Entretanto, isto também ocasiona uma perda de produtividade, pois são obrigados a mover-se de uma ferramenta para outra, trocando a informação de forma manual mediante os recursos como recortar e colar. Segundo Banerjee, estas diferenças entre as diversas aplicações de negócios e ferramentas de produtividade devem ser reduzidas para um modo mais fácil, sincronizado e seguro [Banerjee, 2007].

Uma composição refere-se à maneira de disponibilizar soluções na empresa, assemelhando-se a componentes pré-construídos. Isto inclui também habilidades de personalização e de customização, assim as pessoas podem facilmente e rapidamente modificar funcionalidades específicas da solução criada. Os benefícios são substanciais, porque a composição fornece meios para conseguir agilidade, adaptabilidade e alinhamento nos negócios da empresa [Banerjee, 2007].

Algumas definições de *composite applications* são encontradas na literatura:

- Uma *composite application* é: "uma coleção de serviços que foram montados para fornecer uma potencialidade ao negócio. Estes serviços são artefatos que podem ser desdobrados independentemente, permitindo a composição e utilização das capacidades de plataformas específicas" [Banerjee, 2007].
- Schimidt define que uma *composite application* "é criada por um conjunto de serviços interconectados e parametrizados fornecidos por componentes através de, por exemplo, *Web Services*" [Schimidt, 2003].
- Segundo Crespo, uma *composite application* "é definida através da composição de vários serviços, onde o relacionamento entre eles será realizado por *roles* que fazem o papel de uma interface para a colaboração dos serviços" [Crespo, 2000].

A figura 2.14 apresenta um modelo simplificado do conceito de *composite application*, onde os usuários se conectam as *composite applications* disponibilizadas por meio da orquestração de diversos serviços.

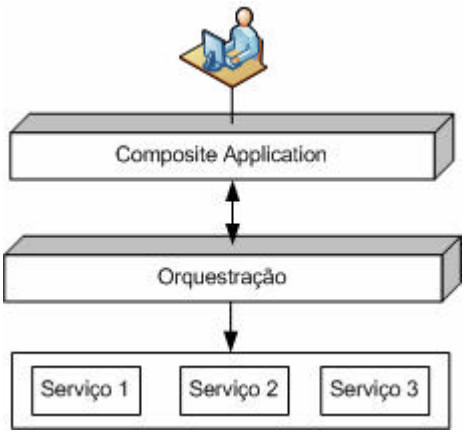

Figura 2.14 – Modelo de *composite application* [Banerjee, 2007].

Em SOA, *composite application* é o produto final. Estas representam o valor de negócio de uma empresa derivada da sua aplicação de SOA. Independente de a *composite application* ter sido planejada para uso interno ou externo, ela representa como uma empresa pode mapear suas necessidades e processos de negócios para que sejam disponibilizados através dos princípios de SOA.

Composite applications têm um grande potencial de mudar a maneira como as aplicações são construídas, entregues e utilizadas pelos usuários finais. Em alguns níveis, entretanto, isto complica o trabalho dos desenvolvedores de aplicações, pois se torna necessário considerar a experiência dos usuários no desenvolvimento. Quando um processo de negócio for disponibilizado como serviço, deve-se considerar com cuidado seus limites para que possa ser utilizado em ambientes compostos [Banerjee, 2007].

2.2.6 Arquitetura orientada a funcionalidades e arquitetura orientada a serviços

As diferentes arquiteturas existentes tornaram-se alvo de interessantes comparações com a proposta de SOA. Um estudo realizado por Erl identifica como SOA derivou de muitas características existentes em arquiteturas como: cliente-servidor, distribuída e *Web Services* [Erl, 2006].

SOA emprega tecnologias originalmente utilizadas para construir aplicações cliente-servidor, por exemplo, XML e *Web Services*. Entretanto, SOA e a arquitetura cliente-servidor diferem muito entre si. SOA elimina qualquer dependência com uma estação de trabalho de

usuário, delegando todo o processamento para dentro do servidor, contrário a arquitetura cliente-servidor.

A arquitetura de Internet distribuída tem muito mais em comum com SOA, incluindo as tecnologias de XML e *Web Services*. Entretanto, SOA tem características distintas para ambas as tecnologia e princípios de projeto. Por exemplo, SOA introduz requisitos de processamento e segurança que diferem da arquitetura de Internet distribuída, e a administração de SOA é tipicamente mais complexa no que se refere à comunicação baseada em mensagens.

Em relação à arquitetura orientada a objetos, não existe competição entre ambas, apenas comparação, visto que o projeto da arquitetura orientada a serviços está baseado no projeto de serviços e a arquitetura orientada a objetos está centrada na criação de objetos.

Muitos princípios de SOA são relacionados e derivados dos princípios da arquitetura orientada a objetos. O princípio de herança da orientação a objetos não faz parte dos princípios de SOA. Assim como fraco acoplamento e autonomia não são promovidos nos princípios da orientação a objetos. A seguir é apresentada uma lista de aspectos comparativos entre ambas as arquiteturas [Erl, 2006].

Tabela 2.2 – Comparação entre orientação a serviços e orientação a objetos [Erl, 2006].

Princípios da orientação a serviços	**Princípios da arquitetura orientada a objetos relacionados**
Serviços reusáveis	Abstração e encapsulamento suportam reuso, porém requerem uma distinta separação entre interface e implementação lógica.
Serviços compartilham contrato formal	O requisito de contratos de serviço é muito comparável para o uso de interfaces na construção de aplicações, mas ainda é considerada um dos itens de melhores práticas.
Serviços são fracamente acoplados	O uso de herança e outros princípios tornam os relacionamentos entre os objetos altamente acoplados.
Serviços devem abstrair lógica de negócios	Uma classe fornece uma interface para o mundo externo tornando-se acessível via interface.
Serviços são passíveis de composição	Suporte a conceitos de associação, tais como: agregação e composição.
Serviços são autônomos	Referências entre objetos e dependências relacionadas com herança suportam um pequeno grau de autonomia.
Serviços são *stateless*	Objetos combinam classes e dados sendo naturalmente *statefull.*
Serviços são passíveis de descoberta	Projetar classes de interfaces para serem consistentes e descritivas é um dos itens de melhores práticas.

2.3 Aplicação das tecnologias estudadas

O SINS consiste em um ambiente para geração de aplicações baseadas em serviços, gerenciando os serviços e facilitando o uso destes, permitindo que usuários criem aplicativos de forma rápida sem a necessidade de um profissional de software.

O SINS faz uso de *Web Services* no papel de serviços, mapeados para atender as necessidades de negócios já implementadas nos sistemas legados (podendo estar em diferentes máquinas, servidores de aplicação e plataforma) ou de novos sistemas, ambientados num repositório e comunicando-se via rede. Segundo os conceitos apresentados por Graham [Graham et al., 2002], pela W3C [W3C, 2007] e por Erl [Erl, 2006], optou-se por consumir *Web Services* sob o conceito de serviços, de forma que uma regra de negócio possa ser definida, descrita e descoberta. O capítulo 4 descreve o ambiente desenvolvido, apresentando as características incorporadas ao uso de *Web Services*.

Os *Web Services* utilizam XML como base de sua comunicação, seguindo o conceito já apresentado por Erl [Erl, 2006] como requisito essencial para comunicação em arquitetura SOA. O estudo realizado, incluindo a análise de decomposição de serviços, serviu como base para a composição de *composite applications* a partir de serviços (independentemente de terem sido gerados a partir de sistemas legados ou não). A necessidade de disponibilizar estes serviços levou à definição do uso de *Web Services*, que seguirão a definição dos princípios da orientação a serviços descritos em Erl [Erl, 2006]. O capítulo a seguir apresenta trabalhos relacionados com o trabalho aqui apresentado.

3 TRABALHOS RELACIONADOS

Em tempo de pesquisa e confecção deste trabalho foram encontrados trabalhos acadêmicos com foco em SOA, porém sem intersecções diretas com o trabalho aqui apresentado de modo a permitir comparações integrais. Este capítulo busca posicionar o SINS perante os principais trabalhos relacionados a esse tema e que trazem contribuições para a geração das *composite applications*, estando na seção 3.1 o detalhamento de um trabalho acadêmico sobre o assunto e na seção 3.2 a descrição de soluções comerciais.

3.1 Camadas de apresentação e SOA

O trabalho desenvolvido por Link [Link et al, 2006] tem como foco principal a camada de apresentação de SOA e a interface para o usuário final, não estando no escopo do trabalho a conexão com a camada de processos. Os autores propõem um modelo de migração de sistemas legados para camada de apresentação de SOA. A idéia central está em fornecer componentes de apresentação aplicáveis em ambas, nova e antiga, arquiteturas de software. Além disso, é utilizado WSRP (*Web Service Remote Portlets*) na camada de apresentação de SOA, que torna possível a integração de *portlets* de diferentes provedores de serviços sem interesse na sua implementação. WSRP é um protocolo para agregação de índices e aplicações Web interativas de pesquisas remotas.

A tarefa principal do processo de migração está em estender a arquitetura da aplicação legada para que esta possa integrar e invocar *portlets* e com isso fornecer os componentes de apresentação existentes como *portlets*. Esta provisão é feita por um processo iterativo consistido por três passos:

1. Identificar os componentes de apresentação reutilizáveis na aplicação legada;
2. Duplicar o componente de apresentação usando tecnologia de *portlet*;
3. Integrar este *portlet* novamente para o cliente legado.

Os componentes de apresentação devem ser identificados e examinados verificando se estes podem usados para construir *Web Services* orientados a apresentação para SOA. Um outro ponto importante apontado pelos autores está em verificar se o componente pode ser tecnicamente implementado utilizando tecnologia de *portlet*.

Com o *portlet* criado, ele poderá ser usado em qualquer portal que utilize WSRP, mas reusá-lo na aplicação legada necessita de algum trabalho inicial. A aplicação legada necessita ser estendida para mostrar *portlets* junto com a interface do usuário atual. Esta extensão pode ser dividida em três componentes:

- *Portal Controller*: age como um consumidor WSRP responsável por estabelecer a comunicação com o provedor WSRP;
- *Portal Pluggin*: necessário para indicar os fragmentos de marcação agregados dos *portlets* invocados ao usuário;
- *Client Controller*: permite o cliente legado comunicar-se com os *portlets* adicionados a aplicação. Esta extensão pode beneficiar o cliente legado de todos os *portlets* publicados no SOA.

A figura a seguir apresenta o cenário de migração proposto pelos autores. O ponto central da arquitetura é o *portlet container*. Cada componente de interface do usuário que pode ser implementado como um *portlet* pode ser integrado no portal e na aplicação legada modificada. Segundo os autores, a vantagem desta abordagem baseia-se no fato de que após cada etapa da migração processar, um sistema plenamente funcional está disponível.

A utilização do padrão WSRP neste trabalho mostrou que existem meios de fornecer os componentes de apresentação como serviços. No entanto, isto não é suficiente para deixar de compor serviços de apresentação manualmente. Algumas restrições, tais como, usabilidade e conveniência das interfaces de usuário devem ser levadas em consideração. Os autores, com sua abordagem apresentada, permitiram migrar componentes de apresentação existentes para *portltes* e utilizá-los dentro de ambas, arquitetura legada e orientada a serviços.

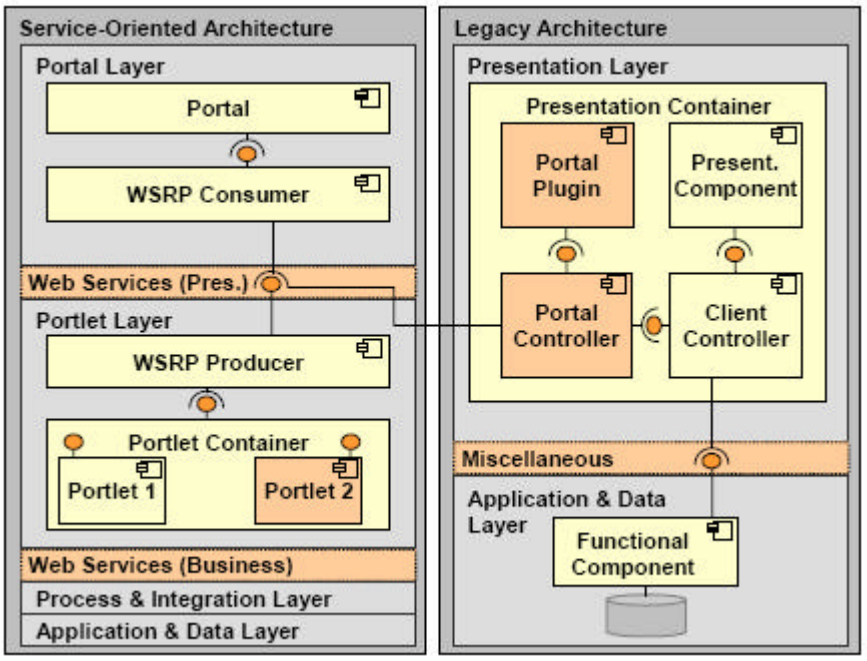

Figura 3.1 – Cenário de Migração [Link et al, 2006].

3.2 Geração de aplicações com suporte de ferramentas: Soluções comerciais

Foram encontradas ferramentas comerciais de desenvolvimento para SOA, oferecidas e disponibilizadas comercialmente pelas empresas IBM, Microsoft, Oracle e SAP. Estas ferramentas, IBM *SOA Foundation*[7], Microsoft *BizTalk Server* 2006[8], Oracle *SOA Suite*[9], SAP *NetWeaver*[10], possuem diferentes mecanismos para desenvolvimento de SOA e fornecem soluções deste a integração com sistemas legados até a disponibilidade dos serviços para uso por outras aplicações.

3.2.1 IBM *SOA Foundation*

Segundo a IBM, SOA fornece flexibilidade e reuso dos processos de negócios de uma empresa. SOA propõe uma arquitetura que combina conexões adaptáveis com relações bem

[7] IBM SOA Foundation – http://www.ibm.com

[8] Microsoft BizTalk – http://www.microsoft.com

[9] Oracle SOA Suite – http://www.oracle.com

[10] SAP NetWeaver – http://www.sap.com

definidas, baseadas em tecnologias padrão para ajudar a flexibilizar infra-estruturas existentes [IBM, 2005].

Os serviços de SOA são extensíveis, baseados na implementação de novos serviços ou recursos de TI existentes. Assim, o desenvolvimento de uma arquitetura SOA começa com uma infra-estrutura flexível, robusta que pode ser utilizada em conjunto com outra infra-estrutura existente e recursos de TI para proporcionar mais valor ao negócio.

A IBM definiu juntamente com seus clientes o ciclo de vida de SOA, apresentado na figura 3.2 a seguir [IBM, 2005]:

Figura 3.2 – Ciclo de vida de SOA proposto pela IBM [IBM, 2005].

- Modelo: levantamento dos requisitos e projeto dos processos de negócios;
- Construção: para os processos otimizados, são construídos componentes que são combinados aos serviços existentes para dar forma aos processos de negócios;
- Implantação: os processos otimizados são implantados num ambiente de serviços altamente seguro e integrado;
- Controle: a informação levantada durante a fase de controle é realimentada para o ciclo de vida para permitir melhoria contínua no processo;
- Governança e processos: fornecem orientação e visão gerencial para o projeto SOA.

A plataforma IBM *SOA Foundation* foi desenvolvida para atender todas as camadas da arquitetura de referência SOA da IBM, na qual define serviços de TI requeridos para fornecer suporte a cada um dos estágios do ciclo de vida de SOA apresentados anteriormente.

A arquitetura de referência SOA inclui um ambiente de desenvolvimento, gerenciamento de serviços, integração de aplicações e processo de serviços em tempo de execução [IBM, 2005].

A figura 3.3 apresenta a arquitetura de referência SOA da IBM.

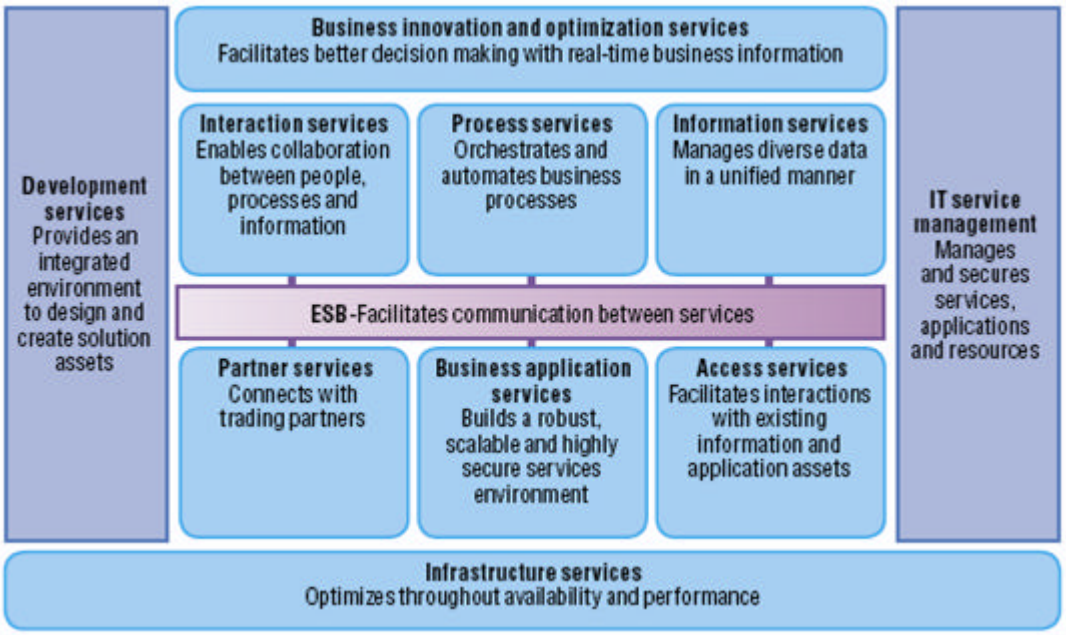

Figura 3.3 – Arquitetura de referência SOA da IBM [IBM, 2005].

Para conseguir abranger toda a arquitetura de referência, a plataforma IBM SOA *Foundation* está dividida em diversas ferramentas, cada uma atendendo uma função específica na arquitetura [IBM, 2006]. Dentre elas algumas se destacam na criação de *composite application* e orquestração de serviços.

IBM *Rational Application Developer*: ferramenta utilizada por desenvolvedores. Com esta ferramenta é possível criar código J2EE[11], por exemplo *servlets* e EJBs, testar o código e implantar no servidor em tempo de execução. Com as ferramentas de *Web Services* é possível criar *Web Services clients*, *Web Services* fornecedores e mapear aplicações existentes como serviços. É um ambiente de desenvolvimento que auxilia na programação de interfaces a partir dos serviços existentes. Estas interfaces são disponibilizadas para os usuários.

[11] http://www.sun.com

IBM *WebSphere Integration Developer*: ferramenta utilizada por arquitetos. Contém vários adaptadores para diferentes bancos de dados ou sistemas de gestão, com o objetivo de permitir a conexão dos serviços com qualquer fonte de dados. É possível criar um processo de negócios com *Business Process Execution Language* (BPEL), configurar adaptadores, criar mediadores, testar o código e implantar no servidor em tempo de execução.

IBM *WebSphere ESB* e IBM *WebSphere Message Broker*: ferramenta de desenvolvimento de ESB. Tem como função garantir o nível dos serviços, fazer transformação de dados entre serviços se necessário, catalogar os serviços e rotear dados entre eles, da forma mais eficaz possível.

IBM *WebSphere Portal*: ferramenta de portal corporativo e colaboração da IBM. O papel na arquitetura SOA é ser a interface do usuário e abrigar as *composite applications.*

IBM *WebSphere Application Server*: trata-se do servidor de aplicação da IBM. É um servidor de web e um servidor de aplicação Java integrado, de forma a executar as *composite applications* e o portal da plataforma.

3.2.2 Microsoft

Segundo a Microsoft, a orientação a serviços é uma abordagem para organizar recursos distribuídos de TI em uma solução integrada que desmembra grupos de informação e maximiza a organização na agilidade dos negócios. Os serviços se comunicam entre si por meio de formatos de mensagens bem-definidos; isso significa que a confiabilidade do aplicativo de sistemas conectados sofrerá uma grande influência da confiabilidade da infra-estrutura de mensagens que ele usa para fazer a comunicação entre os serviços. A orientação a serviços une fontes de informações autônomas construindo uma ponte em grande escala de sistemas operacionais, tecnologias, e protocolos de comunicação [Microsoft, 2006].

Um processo em SOA ocorre de forma iterativa iniciando pela exposição de novos serviços, composição destes serviços em *composite application*, e disponibilização para consumo por usuários ou outras aplicações de negócios. A figura 3.4 apresenta esta visão do ciclo de vida de SOA.

Figura 3.4 – Ciclo de vida de SOA proposto pela Microsoft [Microsoft, 2006].

- Expor: foco nos serviços a serem criados das aplicações e dados em questão;
- Compor: os serviços criados são combinados com outros serviços, aplicações, ou outros processos de negócios;
- Consumir: os serviços criados são disponibilizados para uso por outras aplicações.

Entre alguns aplicativos da Microsoft que ganham destaque na colaboração para desenvolvimento de aplicações voltadas para SOA estão:

- Microsoft *BizTalk Server* 2006: pode-se dizer que no ambiente Microsoft, o *BizTalk Server* é a plataforma para desenvolver SOA. É um servidor da Microsoft que cria soluções para a integração de processos de negócios e incorpora os recursos de integração e automação das tecnologias XML e *Web Services*. O *BizTalk Server* funciona como um mecanismo de execução de processo e como um hub transparente para transformações de documentos e geração de mensagens. Em relação a SOA, estão contempladas as atividades de integração, desenho dos processos pelo arquiteto, *Business Activity Monitor* (BAM) e acesso a fontes de dados diversas para disponibilização dos seus dados como serviço, geralmente sob a forma de *Web Services* [Microsoft, 2005a];
- Microsoft *Sharepoint*: o Microsoft *Sharepoint* é a plataforma para geração, utilização e disponibilização das *composite applications*. É uma ferramenta de portal, colaboração e interface para *composite application*s da Microsoft. Na visão da empresa, todas as aplicações desenvolvidas no Microsoft *Visual Studio* com serviços oriundos do *BizTalk Server* teriam sua interface publicada no portal, no caso, o *Sharepoint*. Além disso, todas as atividades que envolvem o usuário final, são feitas

através de ambiente Web no S*harepoint*, como é o caso da funcionalidade de BAM do *BizTalk Server*. [Microsoft, 2005b];

- Microsoft *Visual Studio*: é o ambiente de desenvolvimento integrado da Microsoft. Além de permitir o desenvolvimento de aplicações web e cliente/servidor, permite o desenvolvimento de *composite applications*, seja para serem usados no *Sharepoint* ou não, a partir de serviços já criados [Microsoft, 2005b]. Possui um módulo *Orchestration Designer* que é uma ferramenta de desenvolvimento visual para construir processos e fluxos de trabalho, os quais incorporam regras comerciais, eventos, transações e exceções, e para vincular estes elementos a objetos de implementação e eventos geradores de mensagens. O processo montado gera um *script* de tempo de processamento baseado em XML (BPEL) do processo que é executado no *BizTalk Server*. A figura 3.5 apresenta uma interface dessa ferramenta.

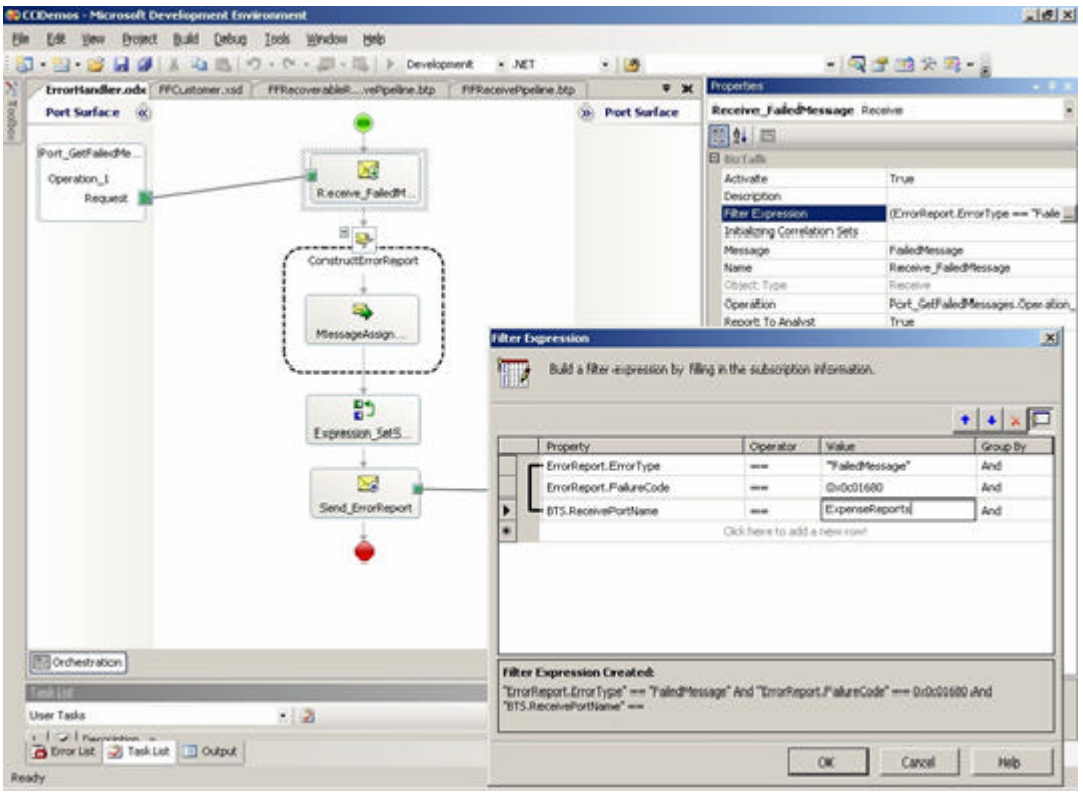

Figura 3.5 – Desenvolvimento de *composite application* a partir do processo criado no *BizTalk* [Microsoft, 2005b].

A figura apresenta um exemplo de orquestração de processos, a fim de gerar *composite applications*.

3.2.3 Oracle *SOA Suite*

Segundo a Oracle, muitas empresas estão direcionando a complexidade de suas aplicações e ambientes de TI para SOA. SOA fornece uma arquitetura que suporta a construção de aplicações conectadas entre empresas, facilitando o desenvolvimento dessas aplicações como *Web Services* que podem ser facilmente integrados e reutilizados, criando uma infra-estrutura de TI flexível e adaptável [Oracle, 2006].

A Oracle publicou o ciclo de vida de SOA. O ciclo apresenta como está organizado o processo de implementação de SOA e é composto de uma seqüência de passos, incluindo desde o desenvolvimento inicial dos serviços, seqüenciamento, gerenciamento até sua utilização [Oracle, 2005]. A figura 3.6 apresenta os sete estágios do ciclo de vida de SOA:

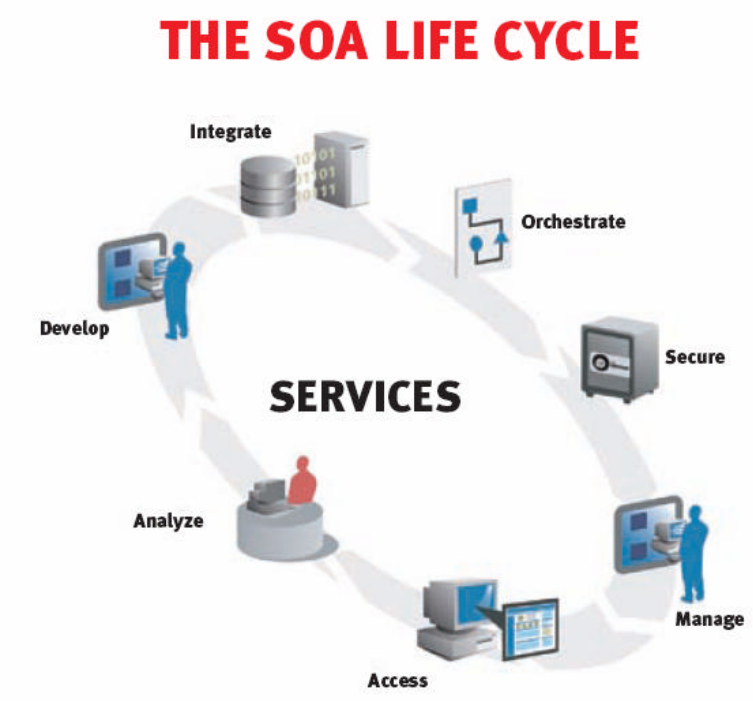

Figura 3.6 – Ciclo de vida de SOA [Oracle, 2005].

- Desenvolver: projeto e construção de serviços que correspondem às etapas específicas dentro de um processo de negócio;
- Integrar: integração dos serviços construídos com outros serviços, sistemas, banco de dados, que requer transformações de dados para mapeamento entre diferentes

esquemas de dados, assim como rotinas dinâmicas para conectar os serviços apropriados em tempo de execução;

- Orquestrar: com os serviços desenvolvidos, a orquestração fornece o seqüenciamento dos serviços a fim de combinar tarefas ou processos de negócios;
- Assegurar: antes dos serviços serem implantados, o acesso para eles deve ser definido;
- Controlar: definir e reforçar acordos em nível de serviço, e políticas operacionais para auditar e faturar (se necessário) o uso do serviço;
- Acessar: os serviços são expostos através de um portal ou uma *composite application*;
- Analisar: a análise dos serviços, eventos e processos de negócios envolvidas em operações de negócios geralmente necessita ocorrer em tempo real, tornando possível o monitoramento, análise e resposta pelos gerenciadores.

Assim como a IBM, a Oracle oferece uma plataforma completa, chamada Oracle *SOA Suite*, para a adoção de SOA, desde a especificação até a implantação, além de ferramentas para manutenção, melhoria e gerenciamento da arquitetura. Todas essas ferramentas trabalham com padrões abertos e são integráveis com ferramentas de outros fornecedores [Oracle, 2006].

O Oracle *SOA Suite* é o conjunto de ferramentas da Oracle para criação de aplicativos sob o conceito da arquitetura SOA assim como adaptação e transformação de aplicativos existentes em serviços [Oracle, 2006]. A figura 3.7 apresenta um diagrama com os principais processos da plataforma.

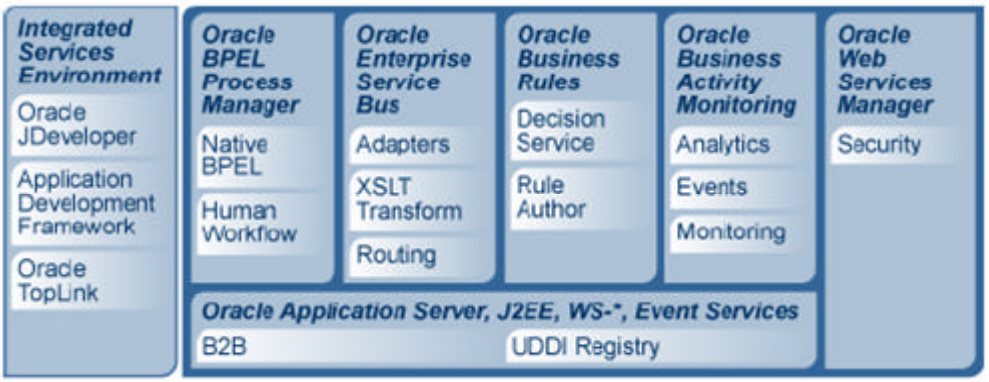

Figura 3.7 – Diagrama dos processos do Oracle SOA *Suite* [Oracle, 2006].

Dentre os componentes que fazem parte do Oracle *SOA Suite*, um deles será descrito a seguir, pois está relacionado com a criação de *composite application*.

Oracle *JDeveloper* é um ambiente de desenvolvimento da Oracle que serve para desenhar os fluxos de BPEL a serem usados pelo BPEL *Process Manager*, programar *Web Services* para serem inclusos como serviços no catálogo ou desenvolver em Java *composite application* que farão uso dos serviços disponíveis no ambiente.

Oracle *Application Server*: serve como servidor Web e servidor de aplicação Java, além de hospedar a ferramenta de portal da Oracle chamada Oracle *Portal* e ser o servidor de aplicação da plataforma. É responsável por ser o servidor de aplicação das *composite applications* criadas no *JDeveloper*.

O processo de criação e disponibilização de uma *composite application* no Oracle *JDeveloper* funciona da seguinte forma:

1. Primeiramente deve-se ter instalado o software na máquina cliente e após isso, um novo projeto deve ser criado. A figura a seguir apresenta a interface do Oracle *JDeveloper*.

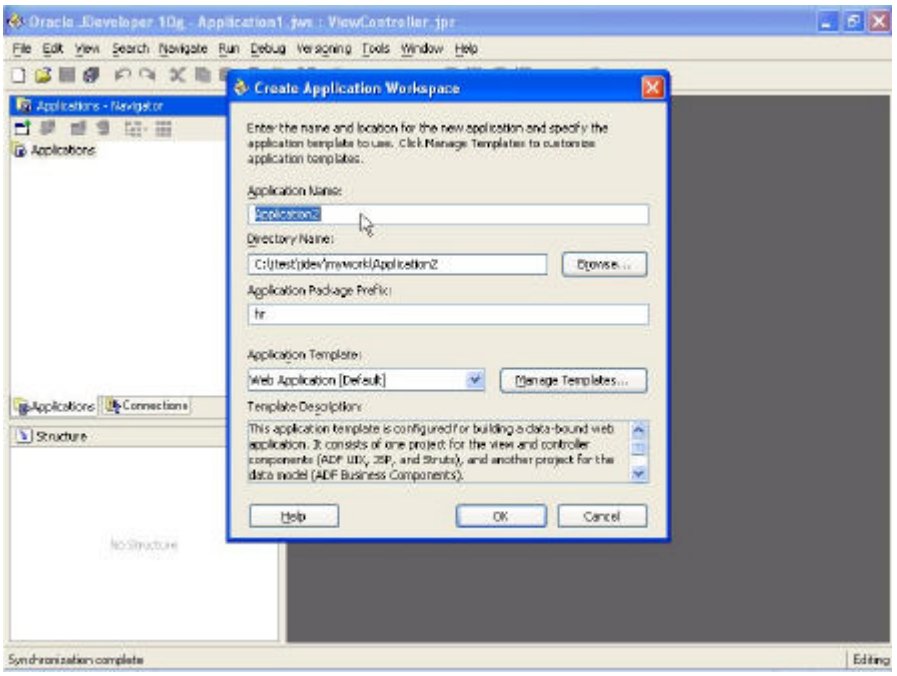

Figura 3.8 – Interface do Oracle *JDeveloper*.

2. Após criado o projeto, deve ser efetuado o mapeamento do *Web Service* via UDDI e gerado um código chamado de esqueleto que servirá de acesso ao *Web Service*. A figura a seguir apresenta as interfaces do aplicativo.

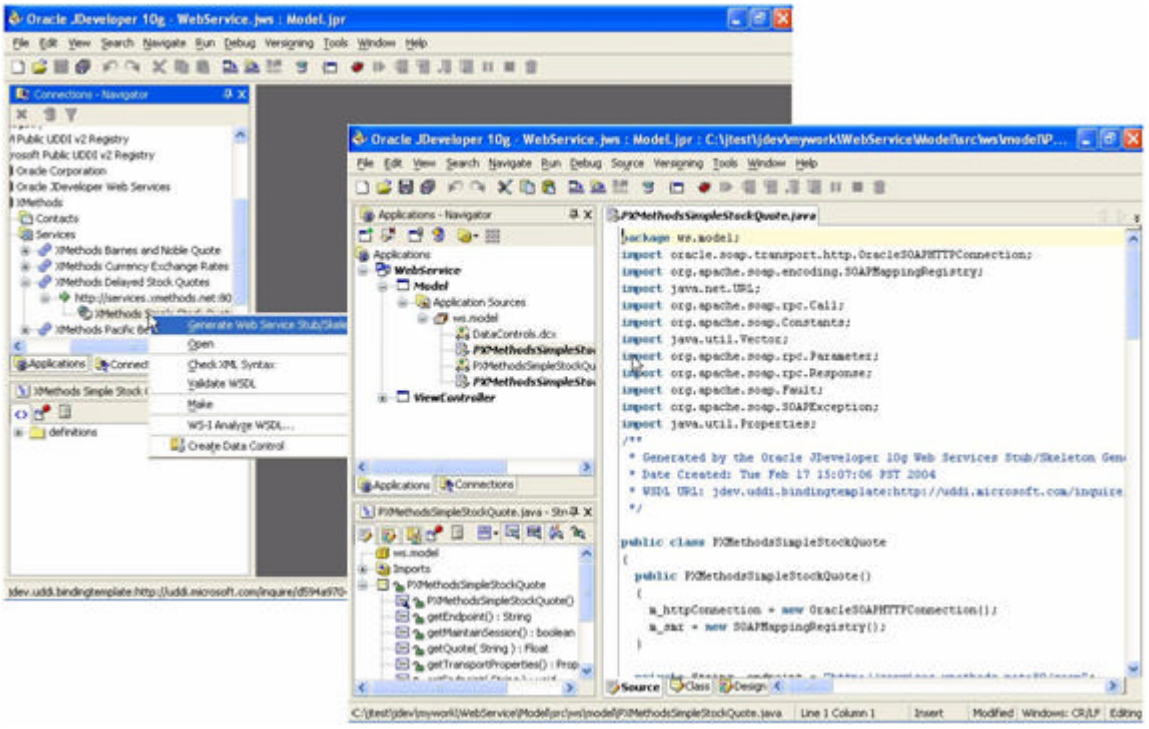

Figura 3.9 – Interfaces de criação do *Web Service*.

3. O próximo passo é montar uma estrutura de funcionamento da aplicação e o desenvolvimento da interface.

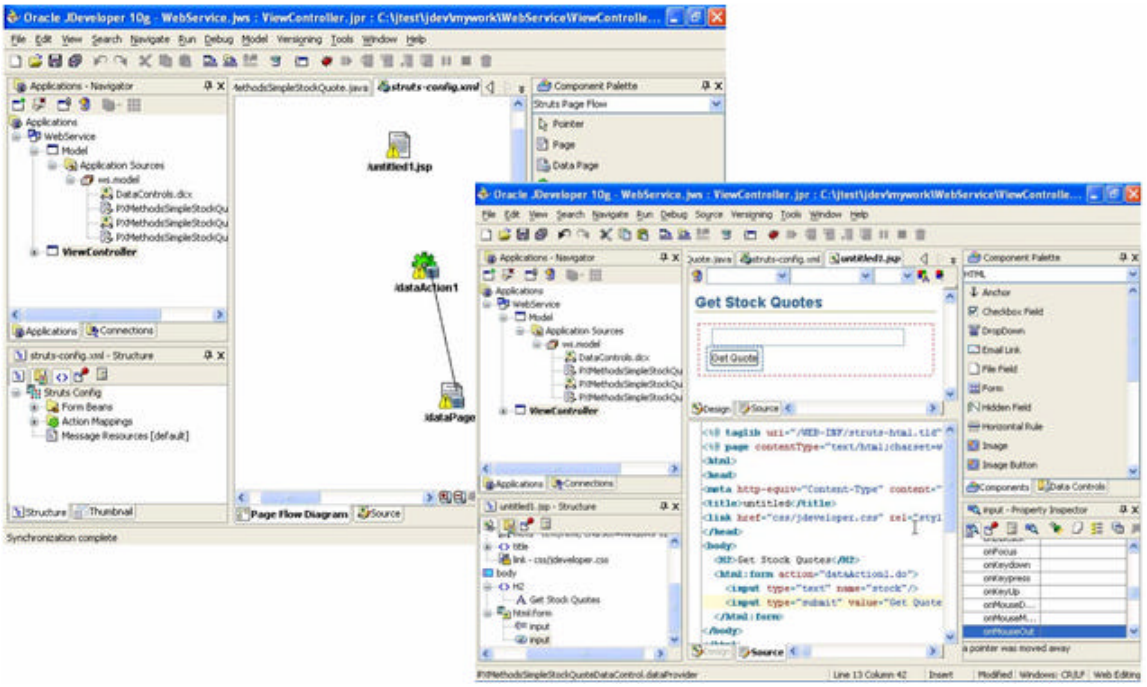

Figura 3.10 – Interfaces de criação da aplicação.

4. Com a interface criada, é necessário efetuar a relação entre os campos da aplicação e o *Web Service*. Após efetuada a correspondência entre os campos, o *deploy* da aplicação pode ser realizado.

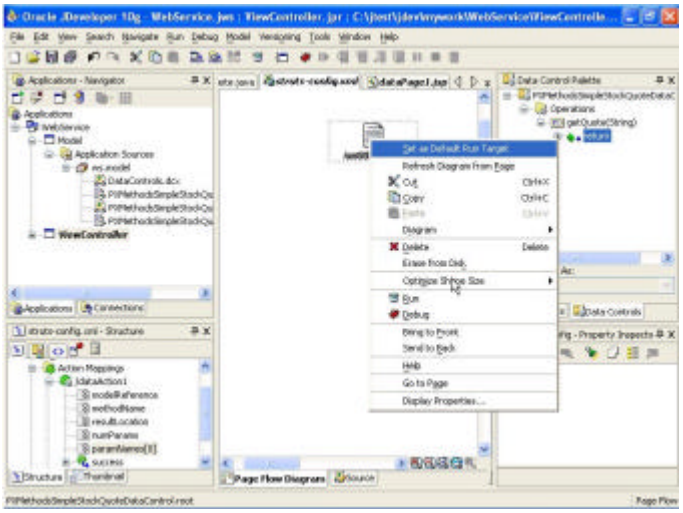

Figura 3.11 – Interface de ligação entre a aplicação e o *Web Service*.

3.2.4 Solução da SAP para SOA

A SAP é uma empresa mundial consagrada na área de *Enterprise Resource Planning* (ERP). Sua estratégia para SOA esta em encapsular os conceitos desta arquitetura sob a sua plataforma SAP *NetWeaver*, ou seja, ao contrário das outras empresas que oferecem soluções parciais e integráveis que juntas formam uma plataforma para desenvolvimento, uso e gerenciamento do SOA, a SAP criou sua própria solução, denominada de *Enterprise Services Architecture* (ESA), com o intuito de que os sistemas legados se convertam a essa arquitetura [SAP, 2006]. Na prática pode-se dizer que a arquitetura ESA, em si, não é mais do que um nome comercial da SAP para a arquitetura SOA. Sendo assim, a diferença prática entre ESA e SOA resume-se ao uso ou não da plataforma SAP.

O SAP *NetWeaver* é uma plataforma voltada para integração, colaboração e interação entre pessoas, informações, processos e aplicações [Heuser, 2004]. A figura a seguir apresenta um overview do SAP *NetWeaver*.

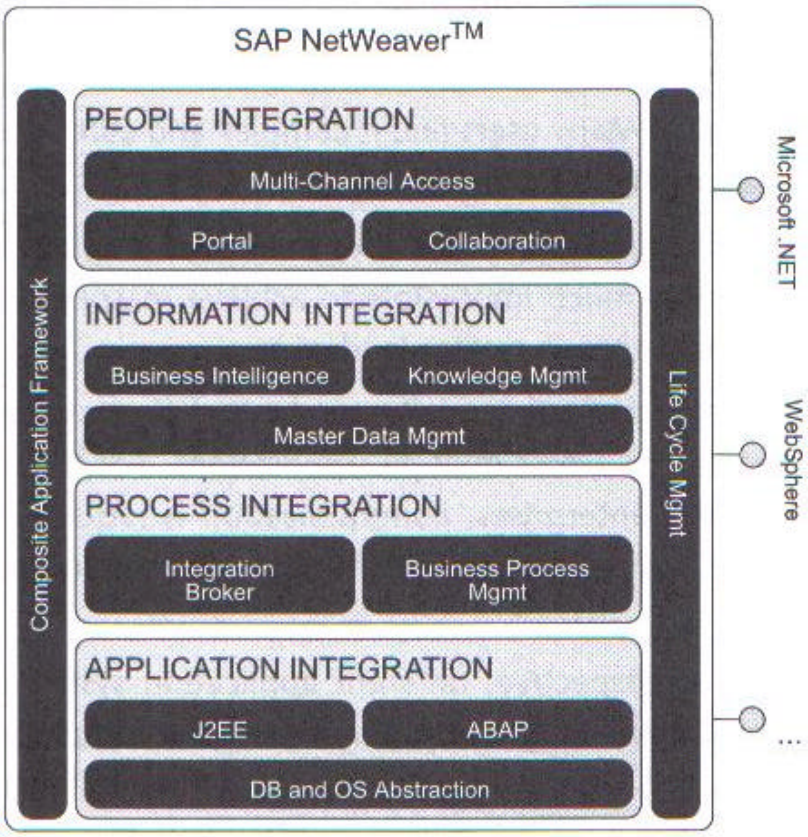

Figura 3.12 – Overview do SAP *NetWeaver*.

O SAP *NetWeaver* busca dados de aplicações ou componentes e disponibiliza serviços para as *composite applications* [SAP, 2007].

O *Composite Application Framework* (CAF) dentro da arquitetura SAP *NetWeaver* é responsável pela criação de novas aplicações. Estas aplicações, também chamadas de xApps, descrevem uma aplicação criada usando os serviços disponibilizados. As funções de interface do usuário e gerenciamento de acesso ficam sob a responsabilidade do SAP *Portal*. Essa ferramenta concentra todas as interfaces do sistema que não sejam de desenvolvimento, ou seja, ela concentra as interfaces de gerenciamento assim como acesso as *composite applications* (chamadas de *iViews* no Portal) já criadas no ambiente [Heuser, 2004].

O SAP XI (de *eXchange Infraestructure*) é o centro da arquitetura ESA. Concentra as funções de acesso a dados e aplicações através de diversos adaptadores, gerenciamento dos serviços do ambiente, monitoramento do status dos serviços e, por fim, gerenciamento dos fluxos de trabalho, fazendo o papel de uma ferramenta de *Business Process Management* (BPM). Sua implementação faz uso de XML e múltiplos adaptadores para conectar diferentes sistemas.

A SAP possui o SAP *NetWeaver Developer Studio* como ferramenta de desenvolvimento, que está baseada no *framework* Eclipse *Open-Source.* Para o desenvolvimento das *iViews* (as *composite applications* do SAP *Portal*), deve-se utilizar uma versão customizada do Eclipse[12] que agrega e padroniza diversas funcionalidades ao *Developer Studio*.

3.3 Ferramentas apresentadas versus solução proposta

Com base nos cinco estudos apresentados, um comparativo se faz necessário para melhor identificar os pontos que a proposta deste trabalho irá focar. Alguns critérios foram definidos para a comparação, e serão explicados a seguir:

- Necessidade de codificação para consumir serviços: esse critério avalia a capacidade da ferramenta de consumir serviços (sob a forma de *Web Services*) sem a necessidade de codificação;

[12] http://www.eclipse.org

- Catálogo de serviços: é o repositório que centraliza dados dos serviços disponíveis em um ambiente facilitando seu uso;
- Geração de *composite applications*: avalia a capacidade das ferramentas gerarem aplicativos (*composite applications*) a partir de serviços, sem necessidade de recodificar toda a interface.

A tabela 3.1 apresenta estes critérios comparativos entre as ferramentas utilizadas no estudo.

Tabela 3.1 – Tabela comparativa.

	Trabalho baseado em Portlets	*IBM WebSphere*	*SAP XI*	*Microsoft Biztalk*	*Oracle SOA Suite*
Necessidade de codificação para consumir serviços	Sim	Sim	Apenas se forem serviços externos ao SAP	Sim	Sim
Catálogo de serviços	Não	UDDI	UDDI ou através de diretório próprio	UDDI ou através de diretório próprio	UDDI
Geração de *composite applications*	Não	Não	Não	Não	Sim, através de uma IDE de programação.

4 MODELO DO AMBIENTE SINS

Neste capítulo será explanado o modelo do ambiente SINS, assim como os meios para se alcançar os objetivos almejados, que são:

- a organização de serviços de forma a facilitar seu uso – diferenciando-se das soluções mostradas no capítulo 3 pelo fato de ter, além de UDDI, os serviços cadastrados em um banco de dados, permitindo a gestão sobre eles através de uma interface Web;

- a capacidade de geração de aplicações (*composite applications*) por um usuário[13], de acordo com suas necessidades, apenas combinando serviços existentes sem a necessidade de codificação ou envolvimento de profissionais da área de software[14] - diferenciando-se das soluções mostradas no capítulo 3 por não necessitar codificação tanto para o consumo de serviços quanto para a geração das *composite applications*.

4.1 Arquitetura do Modelo

O ambiente SINS é composto por uma interface de configuração, usada pelo usuário que irá configurar os serviços e as *composite application* e por uma camada de orquestração, responsável pelo acesso aos *Web Services*, sua indexação, armazenamento de seus dados e geração das *composite applications*. Sua arquitetura é melhor explicada pela figura 4.1.

[13] Entenda-se como usuário uma pessoa com conhecimento avançado no que diz respeito à operação do computador (por exemplo, ferramentas Office) e com conhecimento suficiente do negócio em questão para seleção dos serviços existentes.

[14] Entenda-se como profissionais da área de software como analistas de sistemas, programadores e afins.

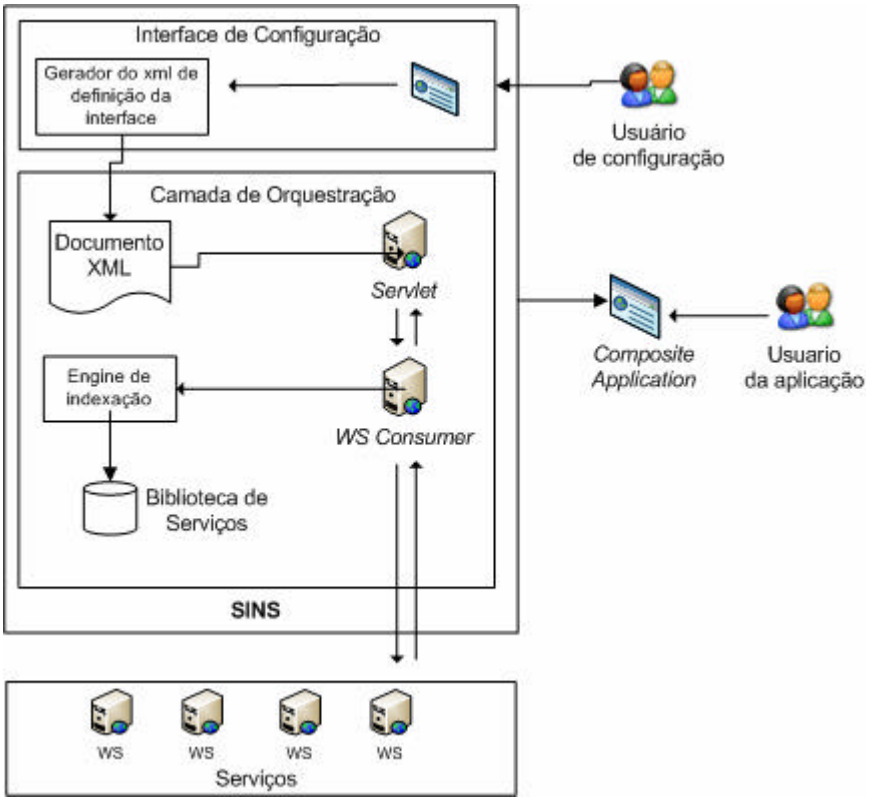

Figura 4.1 – Arquitetura do SINS.

Os componentes da arquitetura apresentada na figura 4.1 são os seguintes:

- Serviços : São os *Web Services* que serão utilizados pelo ambiente.
- Biblioteca de serviços: constitui o conjunto de serviços (*Web Services*) conhecidos pelo ambiente (entenda-se cadastrados em seu banco de dados) ou passíveis de descoberta através de um diretório UDDI acessível pelo ambiente;
- *Engine* de Indexação: constitui de um motor de indexação que percorre o banco de dados, onde estão armazenadas as informações sobre os serviços, e relaciona os serviços existentes, de forma que possa auxiliar o usuário na posterior confecção de uma *composite application*;
- *WS Consumer*: é a parte da *composite application* responsável por acessar os *Web Services*.
- Servlet: é responsável por processar as requisições oriundas de um *browser* e gerar as interfaces das *composite applications*;

- Documento XML: é o documento que armazena os dados que servem como parâmetros para geração das *composite applications*;
- Gerador XML com definição de interface: esse módulo gera um XML a partir das configurações feitas pelo usuário que contém os dados necessários para geração das *composite applications*;
- Interface de configuração: trata-se de uma interface disponibilizada pelo ambiente que fornece as operações necessárias para criação, configuração e alteração de *composite applications*. Esta interface deve ser utilizada pelo usuário de configuração (sendo esse um profissional com conhecimento de operação de computadores e, principalmente, das regras de negócio envolvidas na aplicação);
- *Composite Application*: é a aplicação em si. Trata-se de uma interface que disponibilizará o uso de serviços mapeados na sua configuração (através da interface de configuração). Os serviços são gerenciados pela camada de orquestração do ambiente SINS.

A partir da arquitetura apresentada, os principais componentes do ambiente SINS serão melhor detalhado nas próximas seções.

4.2 Biblioteca de Serviços

A biblioteca de serviços tem por função armazenar as informações pertinentes a todos os serviços conhecidos pelo ambiente, são elas:

- Sistema a que se refere;
- Descrição do serviço;
- Parâmetros de entrada;
- Parâmetros de saída;

Essas informações serão usadas em três momentos:

1) Pelo usuário, na interface de configuração, para construir a sua aplicação;
2) Pela *engine* de indexação, para criar referências entre os serviços;
3) Pela camada de orquestração, no momento de mapear uma aplicação ao seu respectivo serviço.

Quanto à entrada de dados na biblioteca de serviços, ela pode ocorrer de três maneiras:

1) Pelo usuário, na interface de configuração, para construir a sua aplicação;
2) Por uma consulta UDDI, no momento que o usuário utilizar um *Web Service* não cadastrado anteriormente na biblioteca.
3) Por alguma ferramenta que faça a inserção no banco de dados.

Tal repositório será hospedado em um banco de dados relacional, conforme apresentado na figura 4.2. Este modelo é responsável por armazenar todas as informações que serão fornecidas para a geração dos serviços.

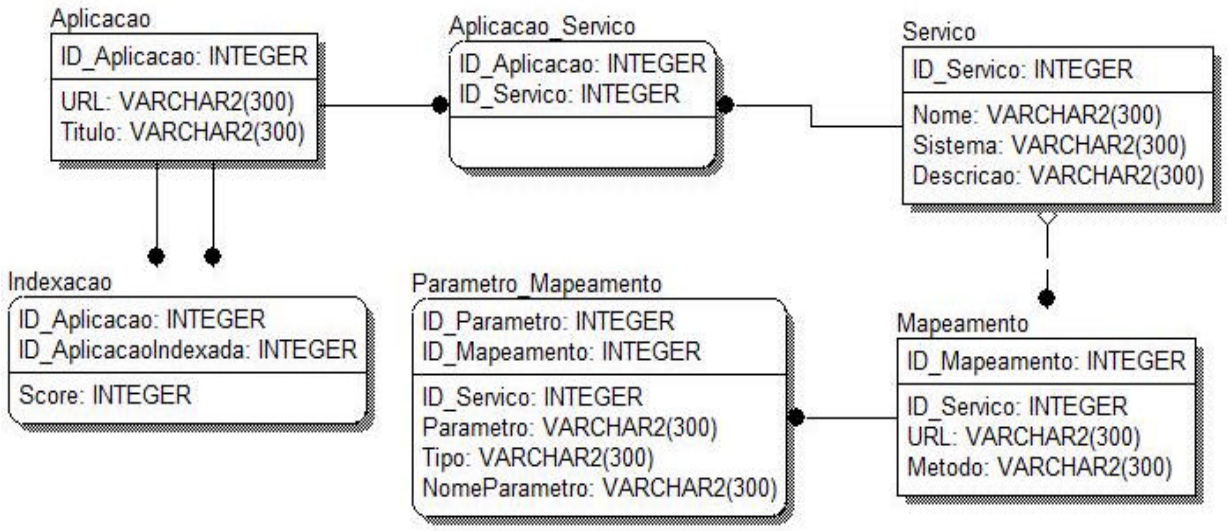

Figura 4.2 – Modelo de dados da biblioteca de serviços.

Na tabela **aplicacao** encontra-se o seu título e sua URL de acesso. Ela, por sua vez, faz uma relação 1-n com a tabela **aplicacao_servico** que também se relaciona 1-n com a tabela **servico**. A tabela **servico** contém os dados do serviço que serão usados pelo usuário na interface, ou seja, que servirão como facilitadores no momento de busca por um serviço. Por fim, o serviço é composto por um **mapeamento** (relacionamento 1-1) que contém a URL de acesso ao *Web Service* do serviço em questão e, através da tabela **parametro_mapeamento** (relacionamento 1-n) define todos os parâmetros e tipos de dados daquele *Web Service.* A tabela **Indexacao** relaciona um conjunto de aplicações juntamente com um *score* que define seu grau de afinidade.

4.3 *Engine* de indexação

A *engine* de indexação trabalha com os *Web Services* conhecidos pelo ambiente, que estão inseridos no repositório de dados. Entende-se por "conhecidos pelo ambiente" os *Web Services* que já tenham sido inseridos no repositório de serviços. Todos os *Web Services* utilizados pela *engine* de indexação devem estar devidamente cadastrados no repositório de dados do ambiente.

O código apresentado na figura 4.3, representa o arquivo WSDL de descrição de um serviço no ambiente, tendo como dados usados pela *engine de indexação* os contidos na *tag* <*wsdl:definitions*>, que indica qual é o *Web Service* em questão, e os dados contidos nas *tags* <*wsdl:message*> que contém os métodos do *Web Service* assim como seus parâmetros de entrada e o seu retorno.

```
<?xml version="1.0" encoding="UTF-8" ?>
<wsdl:definitions targetNamespace="http://192.168.1.102/ws/somador.jws"
   xmlns:apachesoap="http://xml.apache.org/xml-soap"
   xmlns:impl="http://192.168.1.102/ws/somador.jws"
   xmlns:intf="http://192.168.1.102/ws/somador.jws"
   xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
   xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
   xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<wsdl:message name="adicionaRequest">
  <wsdl:part name="campo1" type="xsd:int" />
  <wsdl:part name="campo2" type="xsd:int" />
  </wsdl:message>
<wsdl:message name="adicionaResponse">
  <wsdl:part name="adicionaReturn" type="xsd:string" />
  </wsdl:message>
<wsdl:portType name="somador">
<wsdl:operation name="adiciona" parameterOrder="campo1 campo2">
  <wsdl:input message="impl:adicionaRequest" name="adicionaRequest" />
  <wsdl:output message="impl:adicionaResponse" name="adicionaResponse" />
  </wsdl:operation>
  </wsdl:portType>
<wsdl:binding name="somadorSoapBinding" type="impl:somador">
  <wsdlsoap:binding                                          style="rpc"
transport="http://schemas.xmlsoap.org/soap/http" />
<wsdl:operation name="adiciona">
  <wsdlsoap:operation soapAction="" />
<wsdl:input name="adicionaRequest">
  <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
     namespace="http://DefaultNamespace" use="encoded" />
  </wsdl:input>
<wsdl:output name="adicionaResponse">
  <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
     namespace="http://192.168.1.102/ws/somador.jws" use="encoded" />
  </wsdl:output>
  </wsdl:operation>
  </wsdl:binding>
<wsdl:service name="somadorService">
<wsdl:port binding="impl:somadorSoapBinding" name="somador">
<wsdlsoap:address location="http://server/ws/somador.jws" />
</wsdl:port>
</wsdl:service>
</wsdl:definitions>
```

Figura 4.3 – Modelo do arquivo WSDL.

A função de *engine* de indexação é buscar dentre os serviços conhecidos pelo ambiente aqueles que executam tarefas iguais ou muito similares com o objetivo de mapear serviços alternativos no caso de indisponibilidade de um deles assim com oferecer ao usuário alternativas similares no momento da configuração de uma *composite application*. Isso se fará através dos seguintes critérios:

- Comparação dos parâmetros de entrada e saída do serviço (quantidade, nome e tipo de dado);

- Comparação sintática da descrição do serviço;
- Comparação sintática do nome do sistema a que aquele serviço se refere.

No momento da inserção ou alteração de algum serviço no ambiente, a *engine* buscará entre todos os serviços cadastrados no ambiente aqueles que tenham o mesmo tipo de retorno assim como os mesmos parâmetros de entrada (retorna um *void* e passa como parâmetros *string*, int, *string*, por exemplo), os que obedecerem este critério receberão um "ponto" para cada palavra coincidente do que estiver inserido nos campos "nome", "descrição" e "sistema". O total dessa pontuação será gravado no campo "score" da tabela "indexação", conforme mostrado na seção 4.2.

Uma vez mapeados esses serviços, os que tiverem o maior valor gravado no campo "score" serão oferecidos ao usuário como uma alternativa no momento da criação de uma nova *composite application*, elencados na execução da aplicação quando da indisponibilidade de um deles ou então apenas mostrados na interface de configuração de forma que o responsável pelos sistemas possa evitar redundância de serviços e código.

Importante destacar que todas as comparações realizadas são apenas sintáticas e que esta *engine* existe com o intuito de auxiliar o usuário e proporcionar algum tipo de redundância. Caso existam muitos serviços com parâmetros e/ou nomes iguais, os serviços elencados pela *engine* como alternativos tendem a ser incorretos.

4.4 *Servlet*

O principal produto da interface de configuração é um XML que define uma *composite application*. No momento em que o usuário fizer uma requisição ao endereço configurado na interface de configuração como referente àquela aplicação, um *servlet* padrão irá processar a requisição e, através dos parâmetros do XML, diagramar a página com os campos e/ou informações configuradas, assim como fazer acesso aos serviços pertinentes aquela *composite application.*

O entendimento do interpretador é facilitado através do fluxo apresentado na figura 4.4:

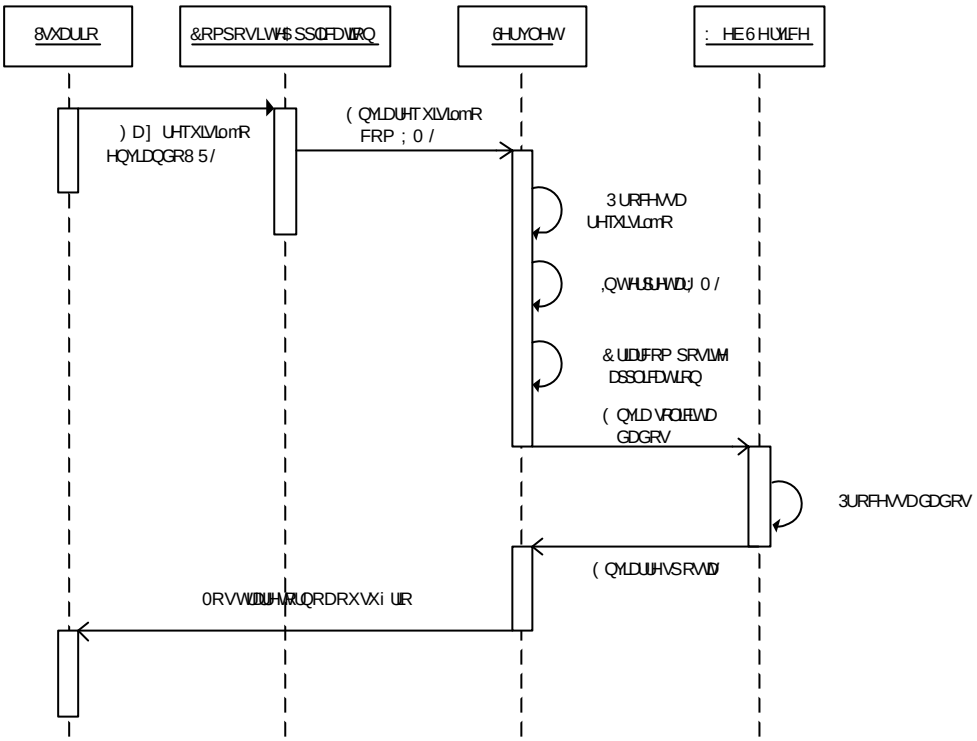

Figura 4.4 – Fluxo do interpretador XML.

O fluxo acima é detalhado a seguir:

1) O usuário faz a requisição a URL da aplicação;
2) A URL direciona para o *servlet* que criará a *composite application* em tempo real, de acordo com o XML de configuração que é apontado pelo URL da aplicação;
3) Os dados oriundos e destinados ao usuário são trocados diretamente com a aplicação, não sendo mais usada a URL inicial até o fim da comunicação;
4) O *servlet* faz todos os envios e recebimentos de dados com o serviço de modo transparente, sendo a *composite application* a única interface conhecida pelo usuário.

```
<?xml version=“1.0”>
<xml>
<dadosAplicacao>
      <id></id>
      <url></url>
      <titulo></titulo>
</dadosAplicacao>
<bindDefinition>
  <idServico>3</idServico>
  <metodo>adiciona</metodo>
  <url>http://localhost/ws/somador.jws</url>
  <nomeParametro>campo1</nomeParametro>
  <tipoParametro>int</tipoParametro>
  <valorParametro />
  <nomeParametro>campo2</nomeParametro>
  <tipoParametro>int</tipoParametro>
  <valorParametro />
</bindDefinition>
</xml>
```

Figura 4.5 – Modelo do XML do interpretador.

Na figura 4.5 mostra-se um exemplo do XML que o interpretador lerá no momento de gerar uma *composite application*. As *tags* trazem definições dos dados para geração da *composite application*, são elas:

- <dadosAplicacao> - corresponde aos dados da *composite application*;
- <id> - identifica o id da *composite application* gerada;
- <url> - identifica o caminho onde está gerada a *composite application*;
- <titulo> - identifica o nome da *composite application* gerada;
- <*bindDefinition*> corresponde a informações de um serviço da *composite application*, podem existir mais de um;
- <idServico> - identifica o id do serviço;
- <metodo> - identifica o nome do método que será invocado;
- <url> - identifica a url do *Web Service*;
- <nomeParametro> - identifica o nome do parâmetro do método;
- <valorParametro> - corresponde a informação fixa do parâmetro, caso necessário;
- <tipoParametro> - identifica o tipo de dado do parâmetro.

4.5 Gerador do XML de definição da interface

De forma transparente para o usuário, inerente a interface de configuração, existe um gerador responsável por gravar, em XML, um arquivo de definição da interface e da aplicação construída pelo usuário.

A função dele é traduzir os dados de uma aplicação (título e URL), serviços atrelados, e campos que devem aparecer na tela, em um arquivo XML que será interpretado pelo interpretador (contido no *servlet*) no momento de gerar uma *composite application*, que ocorre no momento em que usuário fizer o acesso.

O fluxo detalhado do funcionamento, assim como o modelo de arquivo XML são mostrados nas figuras 4.4 e 4.5, respectivamente.

4.6 *Composite Application*

As *composite applications* geradas pelo ambiente possuem interfaces praticamente idênticas, sendo possível apenas pequenas customizações no momento da criação da aplicação através da interface de configuração. É importante destacar as características das *composite application* gerada pelo SINS:

- A *composite application* não existe, de fato, ela é criada em tempo real através da leitura de um arquivo XML que contém suas definições;
- Todas as *composite applications* possuem o mesmo padrão visual;
- Todas as *composite applications* são geradas pelo mesmo *servlet*, o que é favorável no ponto de vista de reuso;
- As regras de negócio, validação e dados são inerentes aos serviços, sendo as *composite application* apenas uma interface para o usuário.

4.7 Restrições e pré-requisitos ao uso do ambiente

Além da necessidade da instalação do ambiente SINS no *application server* e banco de dados, os seguinte pré-requisitos devem ser observados:

- Para seu funcionamento, o SINS necessita de um *application server* Java (por exemplo, Tomcat) assim como da JVM 1.4 ou superior;
- Para seu funcionamento, o SINS necessita de um banco de dados Oracle versão 8 ou superior.
- Somente os serviços devidamente cadastrados no banco de dados ou disponíveis em um diretório UDDI acessível pelo ambiente poderão ser utilizados. Importante salientar que o cadastro dos parâmetros, assim como dos seus tipos de dados devem estar corretos. (erros no cadastro tendem a impedir o uso do serviço);
- Os serviços do sistema deve ser atômicos e *stateless*. A arquitetura não é compatível com sessão assim como pré ou pós execuções no momento da chamada de um serviço;
- Os *Web Services* utilizados devem respeitar os tipos de dados do *set* XSD, tipos de dados customizados não são suportados pelo SINS;
- Os *Web Services* utilizados devem conter em sua codificação a validação dos dados. O SINS faz uma validação simples por preenchimento e tipo de dados, mas não valida máscara (por exemplo, CPF) ou valida os dados digitados.

4.8 Conclusão

Este capítulo teve como objetivo apresentar o modelo do ambiente proposto, detalhando seus componentes e suas funcionalidades, assim como explanar o uso feito das tecnologias apresentadas e a diferença da abordagem em relação aos trabalhos relacionados.

O próximo capítulo apresentará como cada um destes componentes foi implementado bem como as tecnologias utilizadas neste desenvolvimento.

5 IMPLEMENTAÇÃO DA ARQUITETURA

Neste capítulo serão mostrados os detalhes inerentes a implementação da arquitetura, bem como as tecnologias que foram utilizadas no seu desenvolvimento, a estrutura e componentes resultantes. Além disso, o capítulo apresenta as telas do sistema, arquivos WSDL e XML reais gerados pelo SINS e um passo a passo mostrando o processo de criação de uma *composite application* usando o ambiente.

5.1 Modelo de Componentes da Arquitetura

O ambiente SINS foi implementado utilizando a linguagem de programação Java 2 Enterprise Edition 5[15] (J2EE), servidor de aplicações Tomcat 6.0[16] e banco de dados Oracle 10g Express Edition[17]. O protótipo da interface de configuração foi implementado em Java Server Pages (JSP). Além da própria linguagem Java, foi utilizado o Axis[18] para parte de consumo de *Web Services*, o jUDDI[19] para parte de consulta a catálogo de serviços, e o NetBeans 5.5[20] como IDE de desenvolvimento.

A plataforma Java foi escolhida pela sua facilidade em lidar com *Web Services*, pela ampla utilização para o desenvolvimento Web e por ser multiplataforma. A figura 5.1 ilustra o modelo de componentes identificados para a construção do ambiente.

[15] Sun Java – http://java.sun.com

[16] Apache Tomcat – http://tomcat.apache.org

[17] Oracle – http://www.oracle.com

[18] Apache Axis – http://ws.apache.org/axis/

[19] Apache jUDDI - http://ws.apache.org/juddi/

[20] NetBeans – http://www.netbeans.org

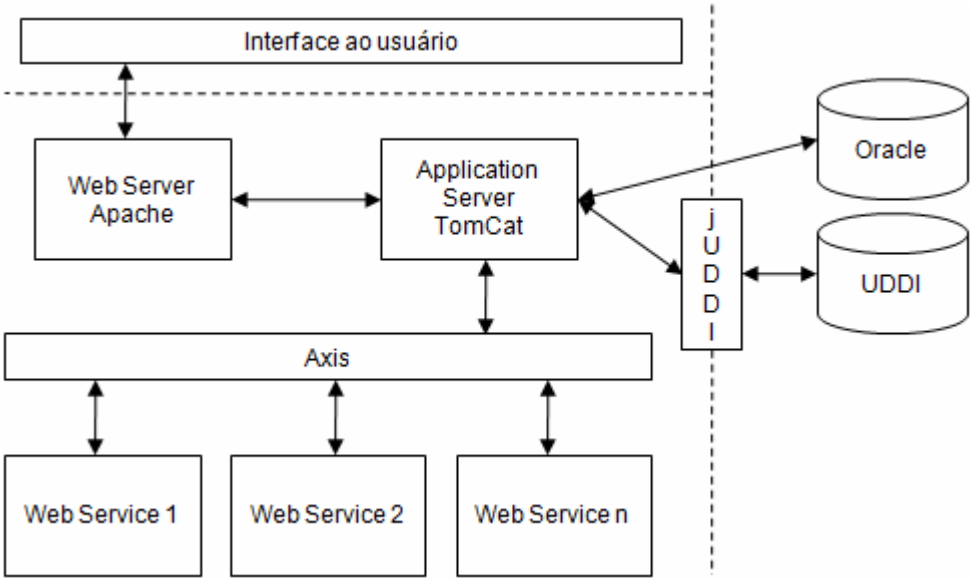

Figura 5.1 – Modelo de componentes do ambiente.

Segue uma descrição detalhada de cada componente e suas principais características:

1. *Web Server*: servidor responsável por atender as requisições de clientes via HTTP. O *Web Server* utilizado na arquitetura foi o Apache[21], por ser uma plataforma *Open-Source*, de fácil instalação e utilização. Além disso, fornece suporte ao uso de *Web Services*;

2. *Application Server*: servidor de aplicação para páginas Web. O *application server* utilizado na arquitetura esta incluído no pacote do *Web Server* Apache, o Tomcat. Ele trabalha como um servidor de aplicações Java para Web.

3. Axis: utilizado para desenvolvimento da camada de *Web Services*, é um *framework* desenvolvido pela Apache para construir software que possa processar a troca de mensagens SOAP (cliente, servidores, *gateways*, etc), provendo mecanismo para a disponibilização e consumo dos *Web Services* em um servidor de aplicações.

4. jUDDI : utilizado para consulta a repositórios UDDI, é um *framework* para implementação, consulta e cadastro de *Web Services* em um repositório UDDI.

5. Oracle: estrutura que dá suporte a organização e armazenamento das informações dos serviços na arquitetura.

[21] Apache – http://www.apache.org/

6. UDDI : serviço de diretório padrão para uso com *Web Services.*

5.2 Modelo de Classes

A figura 5.2 representa a estrutura das principais classes do ambiente SINS.

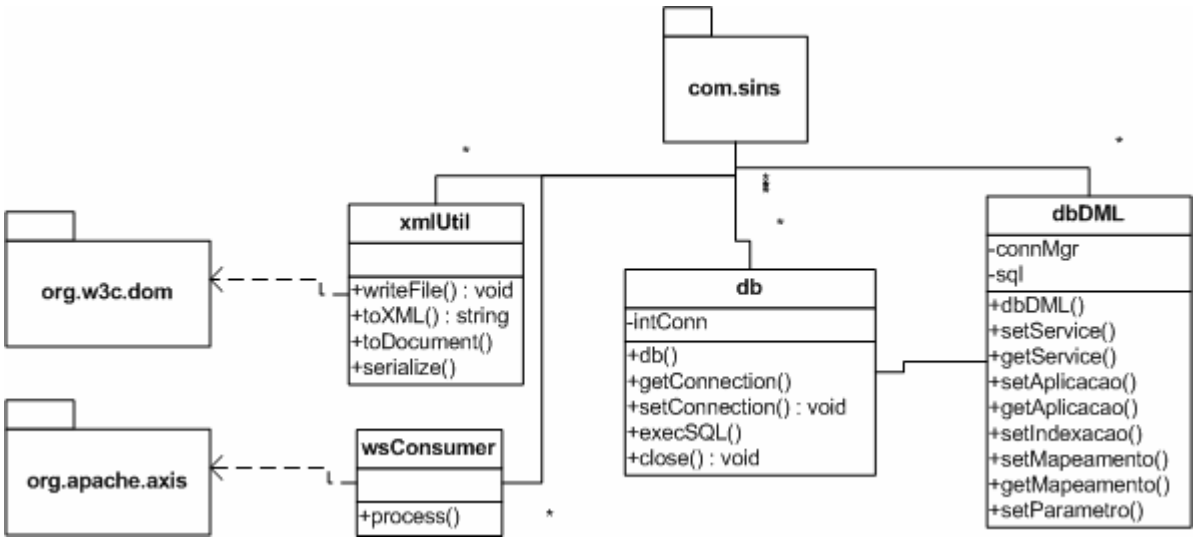

Figura 5.2 – Modelo de classes da arquitetura.

A classe **db** é responsável por executar as operações de conexão no banco de dados, a classe **dbDML** faz uso da classe **db** e é responsável pelas operações de manipulação de dados, a classe **wsConsumer** é responsável pelo consumo (*bind*) dos *Web Services* usados pelo SINS e a classe **xmlUtil** executa operações referentes ao tratamento dos arquivos XML.

A tabela 5.1 apresenta o detalhamento dos métodos da classe db.

Tabela 5.1 – Descrição dos métodos da classe db.

Método	**Função**
getConnection	efetua a conexão com o banco de dados e retorna um objeto do tipo connection
setConnection	configura a conexão com o banco de dados para um objeto de classe
execSql	executa uma query no banco de dados, enviada por parâmetro
Close	fecha a conexão com o banco de dados

A tabela 5.2 apresenta o detalhamento dos métodos da classe dbDML.

Tabela 5.2 – Descrição dos métodos da classe dbDML.

Método	**Função**
setService	efetua a inclusão ou alteração dos dados de um serviço
getService	obtém os dados de um serviço
setAplicacao	efetua a inclusão ou alteração dos dados de uma aplicação
getAplicacao	obtém os dados de uma aplicação
setIndexacao	efetua a inclusão ou alteração dos dados da indexação
setMapeamento	efetua a inclusão ou alteração dos dados do mapeamento
getMapeamento	obtém os dados do mapeamento
setParametro	efetua a inclusão ou alteração dos dados do parâmetro

A tabela 5.3 apresenta o detalhamento dos métodos da classe *wsConsumer*.

Tabela 5.3 – Descrição dos métodos da classe *wsConsumer*.

Método	**Função**
Process	efetua o consumo de um *Web Service*

A classe wsConsumer é apresentada no trecho de código da figura 5.3. O método "process" da classe "wsConsumer" funciona como um consumidor genérico de *Web Services*. Para tal, ele precisa apenas receber o endereço em que o *bind* será feito (endpoint), o método a ser chamado do *Web Service* (method) e os parâmetros (parameters). Tendo em vista que os parâmetros de um *Web Service* podem ser de qualquer um dos tipos XSD existentes, é feito um *loop* pelo *array* de objetos "parametros" inserindo na chamada um parâmetro com seu respectivo tipo de dados a cada iteração do *loop*. Feito isso, a chamada é feita através do método *invoke* (no caso, no objeto *call*, disponibilizado pelo org.apache.axis.client.Call do pacote Axis) e no seu retorno padrão, do tipo *Object*, é feito um *cast* para que este possa ser facilmente interpretado posteriormente.

```
package mestrado;

import org.apache.axis.client.Call;
import org.apache.axis.client.Service;
import org.apache.axis.encoding.XMLType;
import org.apache.axis.utils.Options;

import javax.xml.rpc.ParameterMode;

public class wsConsumer {

public wsConsumer() {}

public static String process(String endpoint,String method, Object[]
parametros) throws Exception {

String retorno = "";

Service  service = new Service();
Call     call    = (Call) service.createCall();

call.setTargetEndpointAddress( new java.net.URL(endpoint) );
call.setOperationName( method );

for (int i=0;i<parametros.length;i++) {
 if (parametros[i].getClass().toString().toLowerCase().indexOf("int") > 0)
call.addParameter("param"+Integer.toString(i),XMLType.XSD_INT,ParameterMod
e.IN);
 else
call.addParameter("param"+Integer.toString(i),XMLType.XSD_STRING,Parameter
Mode.IN);
  }
  call.setReturnType( XMLType.XSD_STRING );
  retorno = (String) call.invoke(parametros);
      return (retorno);
   }
}
```

Figura 5.3 – Trecho do código do *wsConsumer*.

A tabela 5.4 mostra o mapeamento feito entre os tipos de dados suportados pelos *Web Services*, XSD, e seus equivalentes em Java. A tabela também elucida quais são os tipos de dados suportados pelos SINS.

Tabela 5.4 – Mapeamento entre os tipos XSD e Java

XML Schema Type	Java Data Type
xsd:string	java.lang.String
xsd:integer	java.math.BigInteger
xsd:int	int
xsd.long	long
xsd:short	short
xsd:decimal	java.math.BigDecimal
xsd:float	float
xsd:double	double
xsd:boolean	boolean
xsd:byte	byte
xsd:QName	javax.xml.namespace.QName
xsd:dateTime	javax.xml.datatype.XMLGregorianCalendar
xsd:base64Binary	byte[]
xsd:hexBinary	byte[]
xsd:unsignedInt	long
xsd:unsignedShort	int
xsd:unsignedByte	short
xsd:time	javax.xml.datatype.XMLGregorianCalendar
xsd:date	javax.xml.datatype.XMLGregorianCalendar
xsd:g	javax.xml.datatype.XMLGregorianCalendar
xsd:anySimpleType	java.lang.Object
xsd:anySimpleType	java.lang.String
xsd:duration	javax.xml.datatype.Duration
xsd:NOTATION	javax.xml.namespace.QName

A tabela 5.5 detalha os métodos e suas respectivas funções da classe xmlUtil.

Tabela 5.5 – Descrição dos métodos da classe xmlUtil

Método	**Função**
writeFile	efetua a geração de um arquivo xml a partir de uma string
toXML	transforma o resultado de um resultSet em XML
toDocument	cria um objeto Document a partir de um resultSet com elementos definidos para o arquivo
Serialize	obtém os dados de um serviço

5.3 Geração de *Composite Applications*

O fluxo de atividades executadas pelo usuário para geração de uma *composite application* pode ser melhor entendido a partir da imagem apresentada pela figura 5.4:

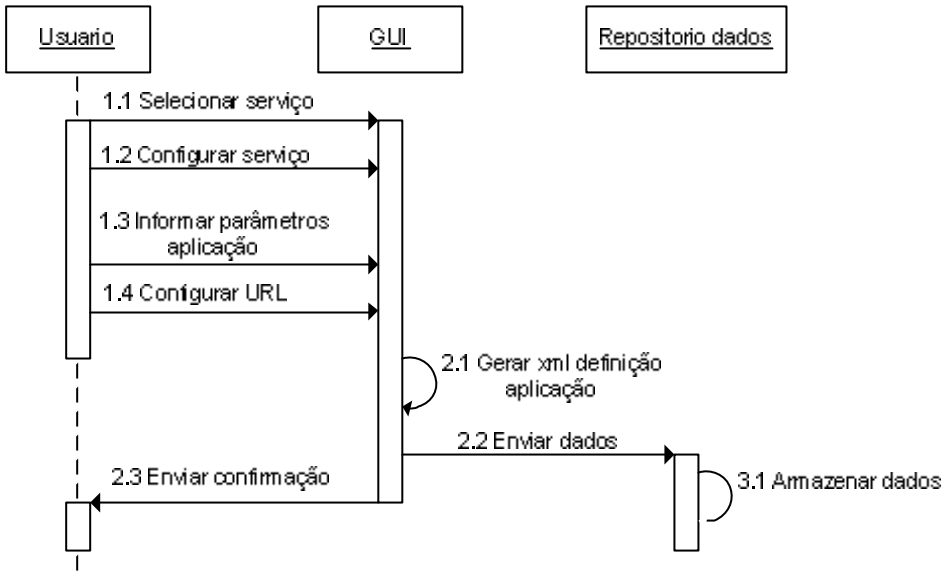

Figura 5.4 – Processo de geração de aplicações.

O usuário, ao acessar a interface, irá navegar pelos serviços de acordo com sua categoria e propósito, selecionando o conjunto de serviços que lhe interessa para criação da aplicação em questão. Feito isso, ele deverá realizar configurações no que diz respeito à disposição dos campos ou informação da aplicação na tela e, por fim, será necessária a configuração da URL de acesso da aplicação antes de gerar, de fato, a *composite application*. O resultado final será uma *composite application* com interface WEB, acessível a partir de um endereço configurável no servidor que hospeda o ambiente proposto por esse trabalho.

A figura 5.5 apresenta a tela inicial do ambiente SINS. Esta interface disponibiliza ao usuário funcionalidades, por exemplo, manutenção dos dados de uma aplicação e manutenção dos dados dos serviços utilizados. Para utilizar uma das funcionalidades disponíveis no ambiente, basta o usuário escolher uma delas e pressionar o respectivo botão.

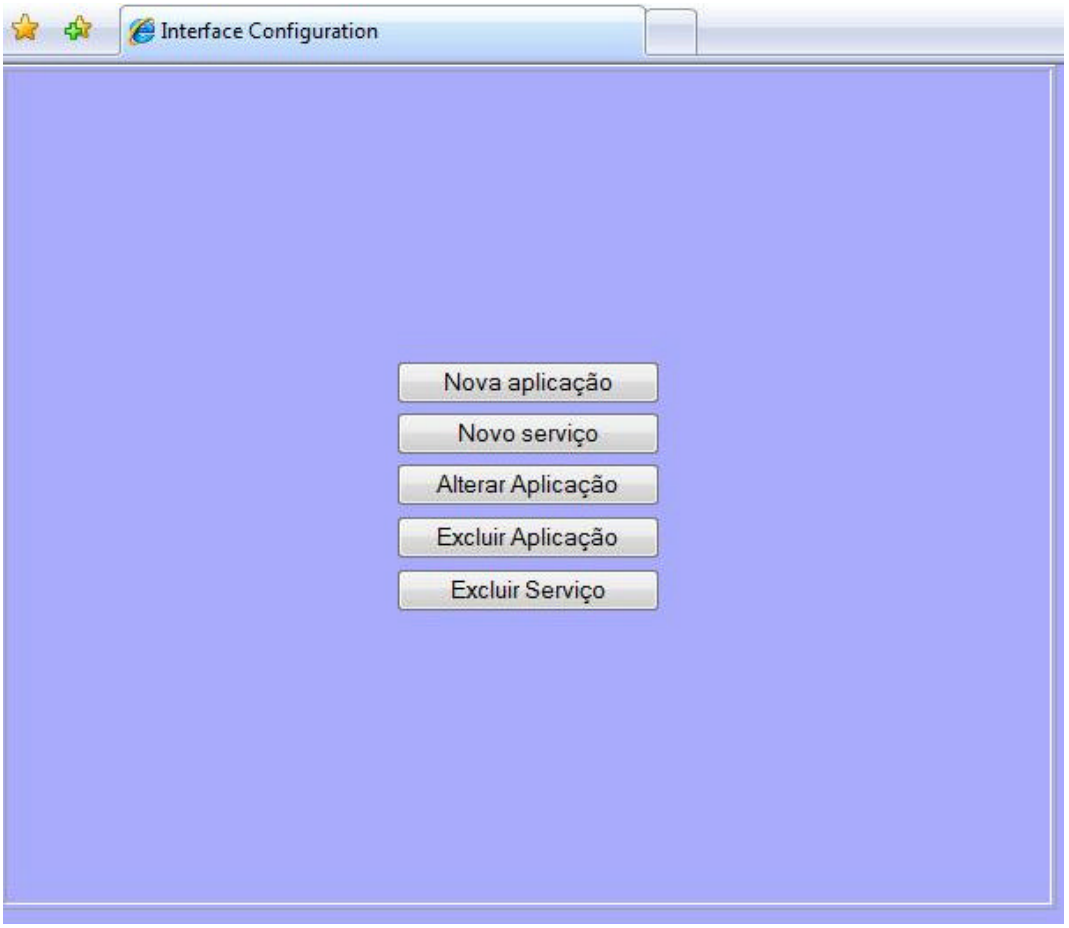

Figura 5.5 – Tela inicial do ambiente SINS.

Para efetuar o mapeamento de um serviço a tela de inserção de serviço é apresentada na figura 5.6. O usuário informa os dados referentes ao serviço que será utilizado pela *composite application*, ficando no "Mapeamento de Parâmetros" a entrada de cada um dos parâmetros do método indicado. Estas informações são armazenadas no banco de dados do ambiente para posterior utilização e são necessárias apenas caso o serviço não esteja cadastrado em um repositório UDDI ou este não estiver disponível.

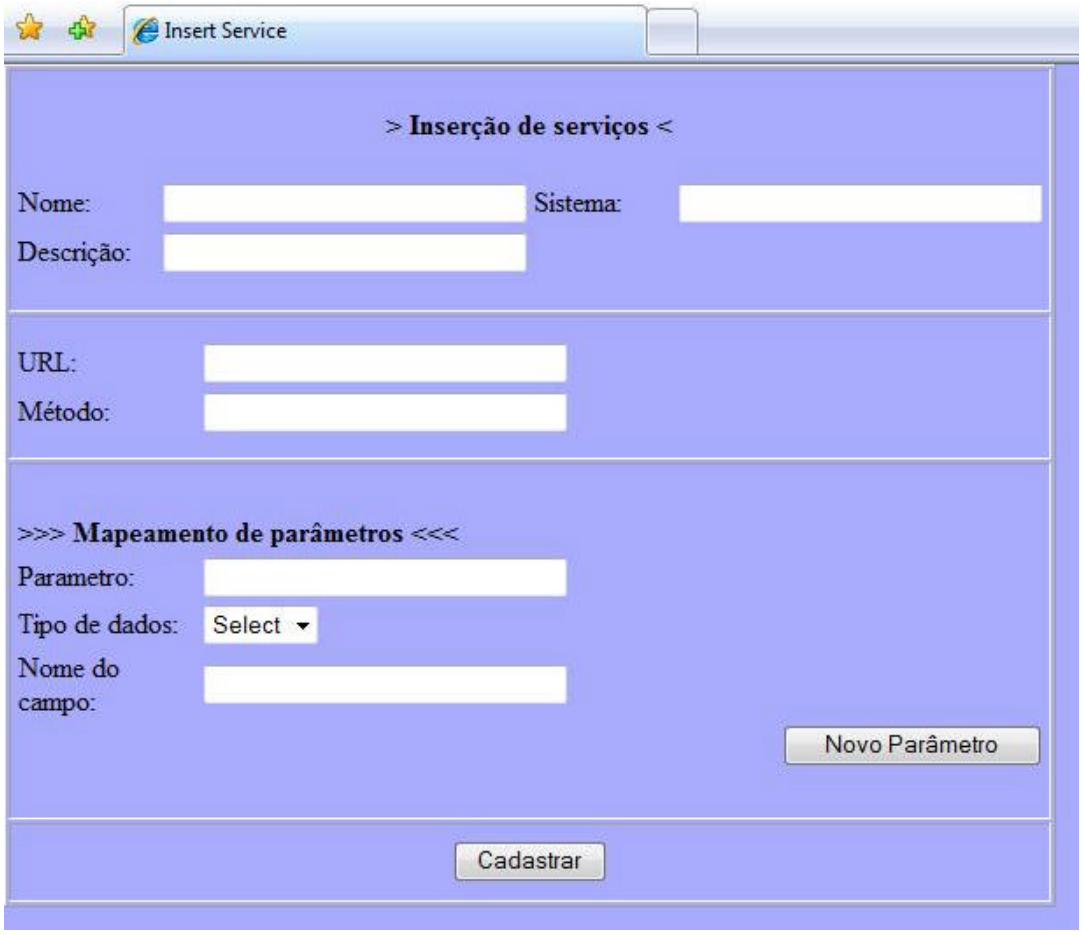

Figura 5.6 – Tela de mapeamento de serviço.

A figura 5.7 apresenta a interface de geração de uma *composite application*. Os dados da *composite application* como *titulo* e *url* devem ser informados para que o ambiente consiga identificar e armazenar a *composite application* gerada. A url contém a informação do diretório por onde a *composite application* será acessada. Este diretório será criado fisicamente pelo ambiente no final do processamento. O trecho de código que gera o diretório é apresentado na figura 5.8.

Os dados dos serviços que serão mapeados são apresentados na sessão “Inserção de serviços”. Estes serviços são pesquisados na biblioteca de serviços assim como em um diretório UDDI conhecido pelo ambiente, bastando selecionar um serviço para que as informações inerentes ao sistema sejam apresentadas. O trecho de código que faz a consulta UDDI junto à consulta ao banco de dados é apresentado na figura 5.9.

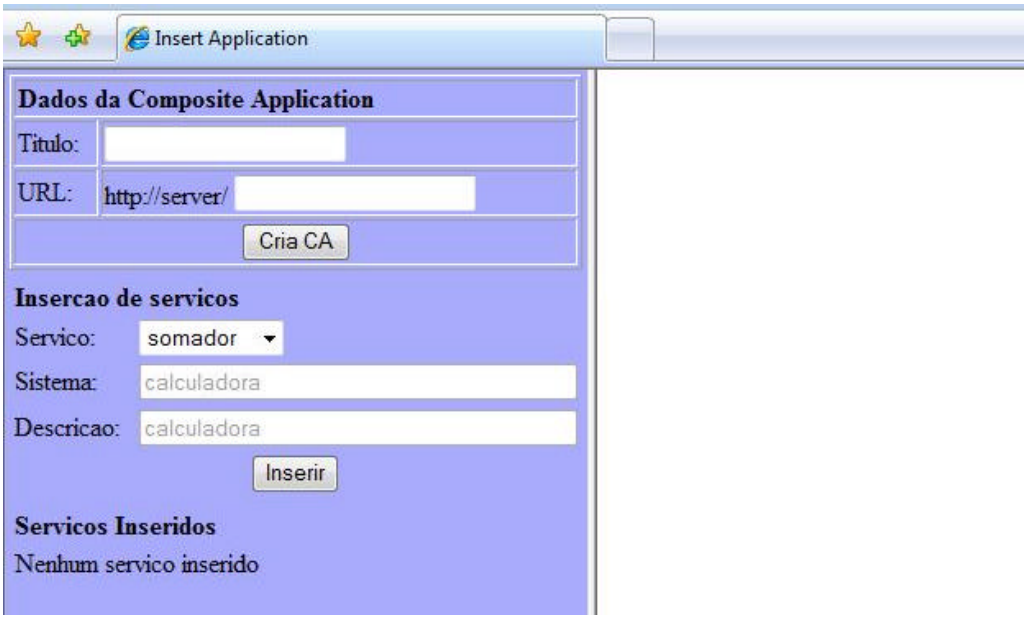

Figura 5.7 – Tela de geração de *composite application*.

Apesar da *composite application* ser gerada dinamicamente a cada acesso, no momento de sua criação é criado um diretório de forma a tornar essa geração transparente para o usuário, além de permitir ao criador da aplicação designar uma URL específica para cada aplicação. No trecho de código da figura 5.8 é mostrado a criação física do diretório digitado pelo usuário (figura 5.8, linha 3), assim como do seu respectivo WEB-INF (figura 5.8, linha 4) e seu arquivo web.xml (figura 5.8, linha 6 até 12), necessários para a identificação e funcionamento de aplicações na grande maioria dos *Application Servers* de plataforma java (incluindo o Tomcat usado neste implementação).

```
1# String dir  = path + request.getParameter("url");
2# String dir1 = dir + "/WEB-INF";
3# (new File(dir)).mkdir();
4# (new File(dir1)).mkdir();
5# //Copia o web.xml
6# FileInputStream fis  = new FileInputStream(new
File(pathXML+"web.xml"));
7# FileOutputStream fos = new FileOutputStream(new File(dir1+"/web.xml"));
8# byte[] buf = new byte[1024];
9# int i = 0;
10# while((i=fis.read(buf))!=-1) {
11#   fos.write(buf, 0, i);
12# }
13# fis.close();
14# fos.close();
```

Figura 5.8 – Trecho de código que cria o diretório de armazenamento da *composite application*.

Na figura 5.9 é mostrado como é feita a consulta contra um serviço UDDI. As linhas 1 à 7 configuram as propriedades para consulta. As linhas 9 à 15 fazem a carga dos dados enquanto nas linhas 19 à 28 são feitos loops que carregam no vetor arrSistema (linha 26) os dados de um novo serviço a cada iteração.

```
1#  props.setProperty
(RegistryProxy.ADMIN_ENDPOINT_PROPERTY_NAME,"http://localhost/juddi/admin");
2#  props.setProperty
(RegistryProxy.INQUIRY_ENDPOINT_PROPERTY_NAME,"http://localhost/juddi/inquiry");
3#  props.setProperty
(RegistryProxy.PUBLISH_ENDPOINT_PROPERTY_NAME,"http://localhost/juddi/publish");
4#  props.setProperty
(RegistryProxy.TRANSPORT_CLASS_PROPERTY_NAME,"org.apache.juddi.proxy.AxisTransport
");
5#  props.setProperty
(RegistryProxy.SECURITY_PROVIDER_PROPERTY_NAME,"com.sun.net.ssl.internal.ssl.Provi
der");
6#  props.setProperty
(RegistryProxy.PROTOCOL_HANDLER_PROPERTY_NAME,"com.sun.net.ssl.internal.www.protoc
ol");
7#  IRegistry registry = new RegistryProxy(props);
8#
9#  CategoryBag catBag = new CategoryBag();
10# catBag.addKeyedReference(new KeyedReference("nome","descricao"));
11#
12# BusinessList list =
registry.findBusiness(null,null,null,catBag,null,null,1000);
13# BusinessInfos infos = list.getBusinessInfos();
14#
15# Vector businesses = infos.getBusinessInfoVector();
16#
17# if (businesses != null)
18# {
19#    for (int i=0; i<businesses.size(); i++)
20#         {
21#           BusinessInfo info = (BusinessInfo)businesses.elementAt(i);
22#           Vector outNames = info.getNameVector();
23#           for (int j=0; j<outNames.size(); j++)
24#           {
25#             Name name = (Name)outNames.elementAt(j);
26#             arrSistema[j] = name.getValue();
27#           }
28#         }
29# }
```

Figura 5.9 – Trecho de código que busca serviços em um repositório UDDI

Pressionando o botão inserir, os métodos são apresentados na tela ao lado, conforme apresenta a figura 5.10.

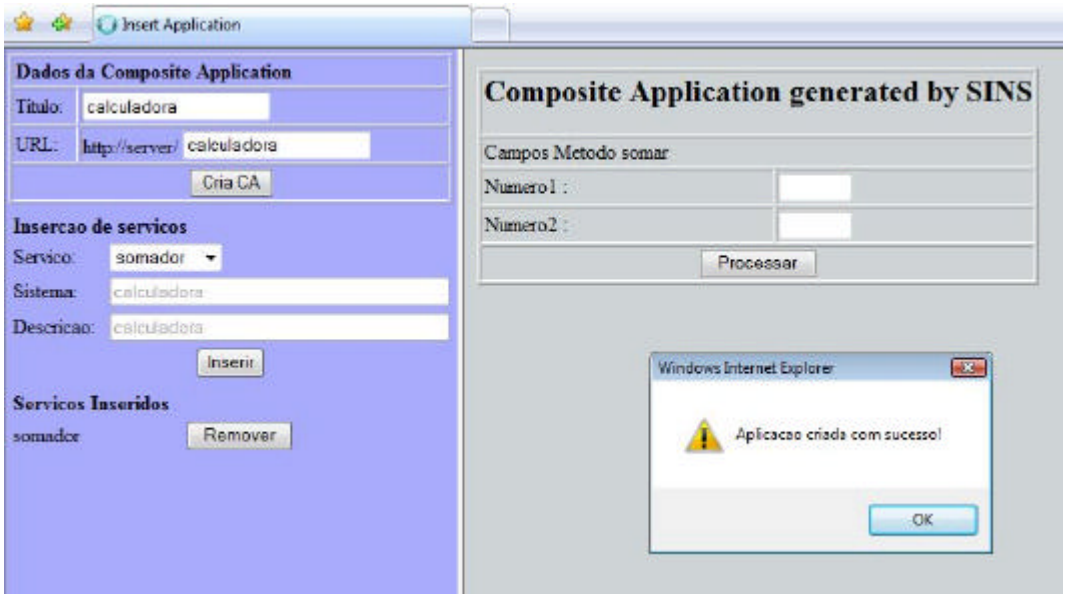

Figura 5.10 – Tela de geração com preview da composite application e confirmação.

Os serviços adicionados são apresentados na tela e podem ser excluídos quando necessário. A figura 5.10 apresenta o botão “Remover” ao lado do serviço adicionado. Após confirmados os serviços que serão utilizados pela *composite application*, ao pressionar o botão “Cria CA”, uma mensagem é apresentada na tela confirmando a geração da *composite application*.

Após confirmar a geração da *composite application*, ela pode ser acessada através do endereço mapeado na url e os serviços selecionados que sejam oriundos de uma consulta UDDI são cadastrados no banco de dados. A figura 5.11 apresenta a *composite application* gerada acessada através da url mapeada.

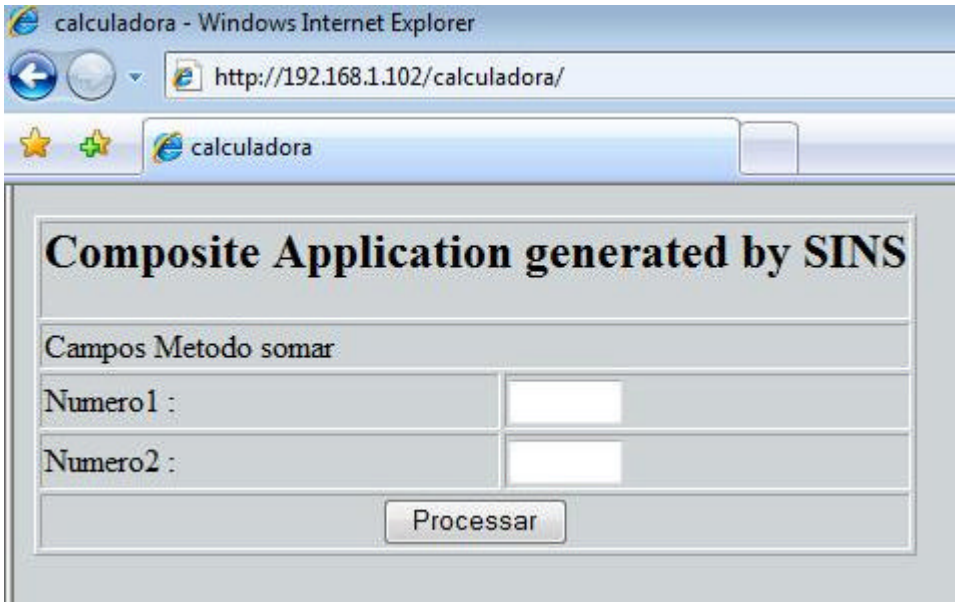

Figura 5.11 – *Composite application* acessada através da url.

O serviço mapeado é apresentado na interface da *composite application*, bastando preencher os campos solicitados e pressionar "Processar" para que o retorno seja apresentado, conforme pode-se visualizar na figura 5.12 e 5.13.

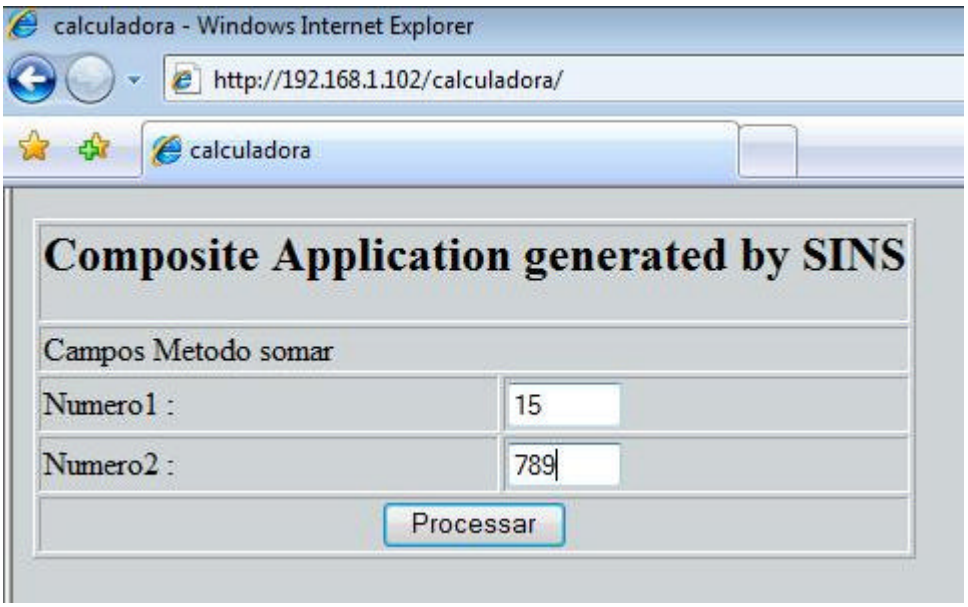

Figura 5.12 – Exemplo de uso da *composite application*.

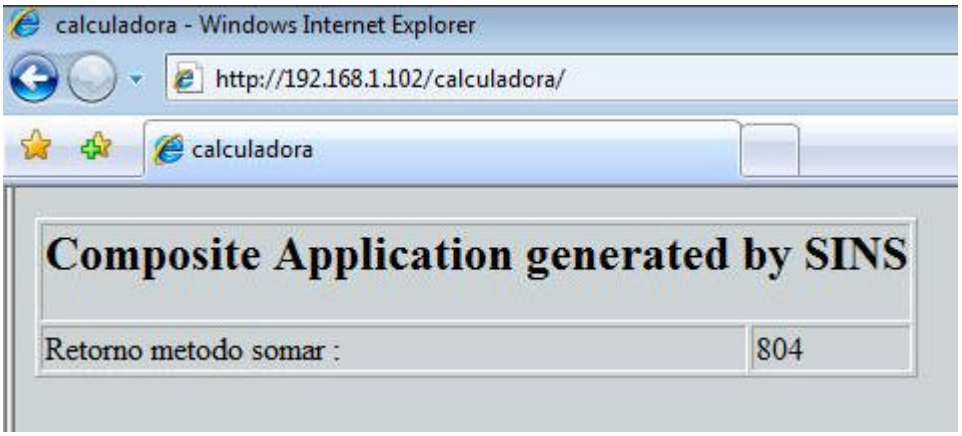

Figura 5.13 – Retorno do serviço “somador” mostrado na *composite application*.

A figura 5.14 apresenta o código HTML da *composite application* gerada. Quando da geração da *composite application*, todos os dados necessários para acesso posterior ao *Web Service* são embutidos no próprio HTML, de modo a evitar novos acessos a banco ou novo processamento do XML. O campo *hidden* "doBind" (figura 5.14, linha 6) sinaliza ao *servlet* que está sendo feito um *submit*, assim o *Web Service* já deve ser acessado passando os dados da *composite application* como parâmetro.

Quando do processamento do *submit*, um *loop* carregará todos os campos de todos os métodos existentes nessa página. Isso é organizado pelos campos "contMetodos" (figura 5.14, linha 12) que diz o total de métodos dessa página e pelo campo contParametros (figura 5.14, linha 18), que diz quantidade de cada parâmetros em cada método. Partindo desse principio, os nomes dos campos são formados dinamicamente a cada iteração, de modo a torná-los identificáveis a um "*request*" e de modo a saber qual campo pertence a qual método. Assim sendo, os campos de métodos são identificados por <campo> seguido do seu índice (figura 5.14, linhas 16,17,18) e os campos de parâmetros são identificados por <campo> seguido do índice do método seguindo pelo índice do campo (figura 5.14, linhas 22,23,26 e 27).

```
1# <title>calculadora</title>
2# <html>
3# <body bgcolor='#CED3D5'>
4# <table border=1>
5# <form method="post">
6#     <input type="hidden" name="doBind" value="true">
7#     <tr>
8#           <td colspan=2>
9#                 <h2>Composite Application generated by SINS</h2>
10#          </td>
11#    </tr>
12#    <input type="hidden" name="contMetodos" value="1">
13#    <tr>
14#          <td colspan=2> Campos Metodo somar </td>
15#    </tr>
16#     <input type="hidden" name="metodo0" value="somar">
17# <input type="hidden" name="urlmetodo0"
value="http://localhost/ws/somador.jws">
18#     <input type="hidden" name="contParametros0" value="2">
19#    <tr>
20#          <td>Numero1 : </td>
21#          <td>
22#                <input type="hidden" name="tipocampo0-0" value="int">
23#                <input type="text" name="campo0-0" size="5">
24#          </td>
25#    </tr>
26#    <input type="hidden" name="tipocampo0-0" value="int">
27#    <input type="text" name="campo0-0" size="5">
28#    <tr>
29#          <td>Numero2 : </td>
30#          <td>
31#                <input type="hidden" name="tipocampo0-1" value="int">
32#                <input type="text" name="campo0-1" size="5">
33#          </td>
34#    </tr>
35#    <tr>
36#    <td colspan=2 align=center><input type=submit value=Processar></td>
37#    </tr>
38# </form>
39# </table>
40# </bodv>
```

Figura 5.14 – Código do HTML da *composite application* gerada.

No HTML gerado dinamicamente pelo *servlet*, existem campos que identificam a quantidade de serviços usados pela *composite application* em questão assim como a quantidade de parâmetros de cada um. Assim sendo, são necessários dois *loops* para leitura dos dados no momento do "*submit*", o primeiro (figura 5.15, linhas 1 e 19) carregando os métodos e o segundo carregando os parâmetros de cada método (figura 5.15, linhas 9 e 14).

A cada iteração por parâmetros de um método, é carregado uma posição em um *array* de objetos (figura 5.15, linhas 10 até 13) de acordo com seu tipo de dado.

Ao fim da carga do *array* de parametros correspondente a cada método, ele é passado como parâmetro para o wsConsumer, que retornará na variável "retorno" o resultado deste *bind* (figura 5.15, linha 17).

Um trecho de código do servlet é apresentado na figura 5.15.

```
1# for (int i=0;i<contMetodos;i++){
2#  nomeMetodo = request.getParameter("metodo"+i);
3#  contParametros = Integer.parseInt(request.getParameter("contParametros"+i));
4#
5#  //carrega os parametros em um array de objetos para passar para o consumer
6#  Object[] parametros;
7#  parametros = new Object[contParametros];
8#
9#  for (int j=0;j<contParametros;j++){
10#    if (request.getParameter("tipocampo"+i+"-"+j).equals("int"))
11#    parametros[j] = new Integer(request.getParameter("campo"+i+"-"+j));
12#    else
13#    parametros[j] = request.getParameter("campo"+i+"-"+j);
14# }// fim do for pelos parametros
15#
16#     //Consome     os     webservices     e     dá     retorno     ao     usuário
17#retorno=wsConsumer.process(request.getParameter("urlmetodo"+i),
       nomeMetodo,parametros);
18#
19# }//fim do for pelos metodos
```

Figura 5.15 – Trecho de código do *servlet* que efetua o *bind* no *Web Service*.

6 APLICAÇÃO DO AMBIENTE SINS (ESTUDO DE CASO)

O estudo de caso apresentado neste capítulo trata de um projeto real de implementação de arquitetura SOA realizado por uma empresa de software do estado do Rio Grande do Sul para uma empresa multinacional da área farmacêutica, tendo sido ele o escolhido por ser um projeto real, por fazer uso de *Web Services* sob o conceito de serviços e por ter diversas interfaces geradas a partir deles.

O capítulo apresenta a descrição do sistema utilizado, a estrutura dos *Web Services* utilizados para geração de *composite applications* assim como uma comparação entre os resultados obtidos com e sem o uso do SINS.

Para validação do ambiente SINS será apresentado um comparativo entre uma WBS[22] do projeto original do sistema com a WBS referente a mesma implementação utilizando o ambiente SINS. Nesta WBS, os valores referentes às etapas onde o SINS não apresenta nenhuma contribuição (por exemplo, especificação e desenvolvimento dos *Web Services*) serão mantidos e os resultados destas etapas no projeto original serão aproveitados.

Ao final do capítulo, é apresentado o resultado dos testes de custo/tempo, bem como uma discussão sobre os resultados obtidos.

6.1 Estudo de Caso: Sistema de Distribuição

6.1.1 Análise do Sistema

A empresa em questão não faz venda direta ao consumidor em alguns países (incluindo o Brasil) e por isso faz uso de distribuidores. Devido aos altos custos recorrentes em manter sistemas de distribuição isolados e com alcance limitado (geralmente de apenas um país), essa empresa optou por criar um novo sistema em arquitetura SOA, de modo a reutilizar ao máximo os seus sistemas legados, padronizar todos os sistemas em uma mesma arquitetura e

[22] *Work Breakdown Structure* – Representa a organização de tarefas ao longo do tempo, com seu custo e duração de execução.

interface e poder ter as customizações inerentes as diferentes regiões passíveis de serem feitas de forma rápida e com o menor custo possível.

O sistema em questão é chamado de “Sistema de Distribuição” e possui quatro funções:

- Registrar pedidos de produtos dos distribuidores;
- Executar um processo externo ao sistema para emissão de faturas aos distribuidores;
- Compatibilizar as unidades do estoque da empresa com as dos distribuidores, por exemplo: em alguns países usam-se litros e gramas, em outros, galões e onças. No Brasil vendem-se alguns produtos em embalagens de 20 comprimidos, em outros de 18;
- Compatibilizar os nomes de produtos entre a empresa e os distribuidores (produtos não necessariamente têm o mesmo nome em todos os países).

O diagrama de casos de uso mostrado na figura 6.1 descreve o fluxo das informações entre o sistema de distribuição e o cliente.

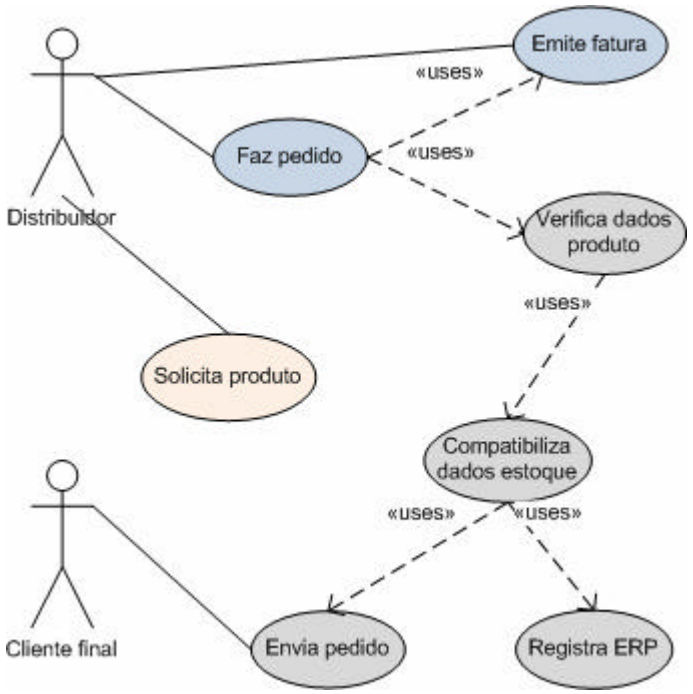

Figura 6.1 – Diagrama de casos de uso.

A arquitetura do Sistema de Distribuição é bastante distribuída, sendo:

- o sistema de pedido dos distribuidores está implementado em Oracle *Forms* situado em outro país;
- o sistema de emissão de faturas está contemplado no sistema ERP nacional da empresa;
- o sistema de conversão de unidades não existia na empresa e foi desenvolvido no escopo do projeto original executado pela empresa gaúcha;
- o banco de dados, com os dados de todos os produtos, está localizado em outro país, e é dito centralizado.

O projeto original contempla a atividade de desenvolvimento em que foram implementadas classes que faziam integração com os sistemas legados de pedido e fatura e gerados os *Web Services* a partir destas classes.

6.1.2 Definição dos papéis envolvidos

Para que seja possível mensurar diferenças e tirar conclusões a partir do projeto desenvolvido originalmente e o projeto feito com o uso do SINS, os perfis dos profissionais envolvidos serão listados, descritos[23] e precificados[24] na tabela 6.1.

[23] De acordo com os papéis existentes na grande maioria das empresas de desenvolvimento de software, empresas com processos "CMMi *Like"* e/ou MSF (*Microsoft Solution Framework)*

[24] Valor médio entre região sul e sudeste do profissional referente a salário acrescido de impostos e *overhead* da empresa. Valores oriundos de pesquisa realizada por uma empresa de RH em caráter privado entre Janeiro e Abril de 2007.

Tabela 6.1 – Descrição e valor/hora dos profissionais.

Função	**Breve Descrição**	**Valor/Hora**
Analista de Sistemas	Responsável por fazer o levantamento das regras funcionais e premissas do sistema	R$ 65,92
Programador	Responsável pela codificação do sistema	R$ 50,00
Gerente de Projeto	Responsável pela condução do projeto como um todo.	R$ 69,52
Programador HTML	Codificador da parte referente a interface do sistema	R$ 38,90
Analista de Testes	Responsável pelo entendimento e definição dos casos de teste do sistema	R$ 44,51
Testador	Responsável pela execução dos casos de teste definidos	R$ 33,66
Projetista de *Software*	Responsável por garantir, junto aos programadores, o andamento do desenvolvimento assim com o uso de melhores práticas	R$ 71,56
Arquiteto de *Software*	Responsável pela definição técnica do sistema	R$ 81,16

6.1.3 Geração das *composite applications* com SINS

O Sistema de Distribuição está desenvolvido na linguagem de programação Java. O diagrama de *Web Services* apresentado na figura 6.2, descreve os serviços desenvolvidos do projeto original e que foram cadastrados no ambiente para geração de *composite applications*.

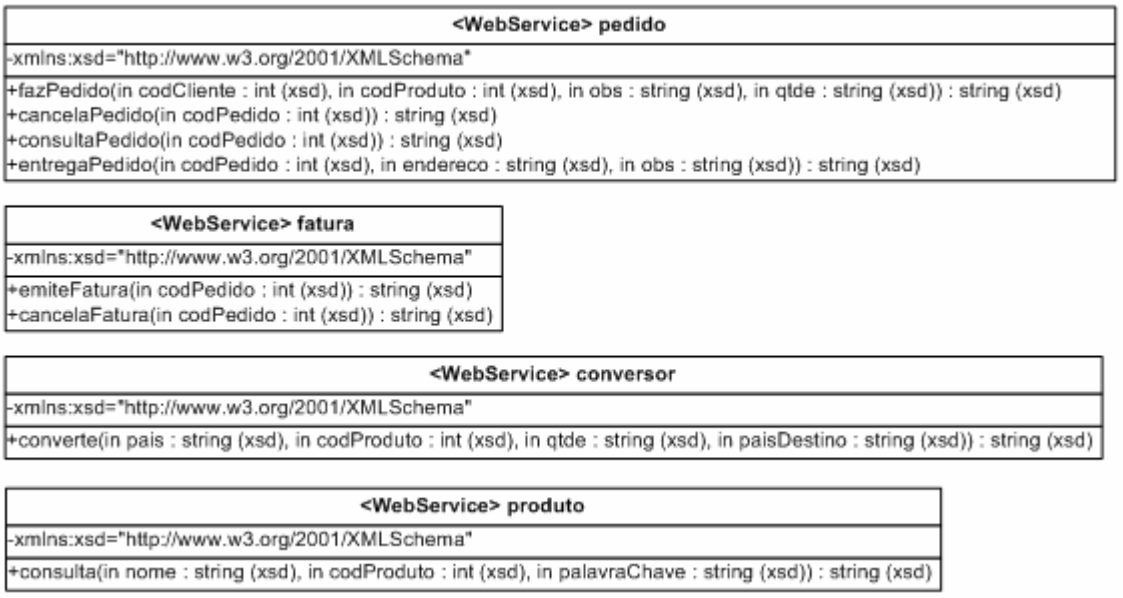

Figura 6.2 – Diagrama de *Web Services*.

O primeiro passo foi acessar a interface de geração de *composite application* e selecionar os serviços que farão parte da *composite application* gerada. A figura 6.3 apresenta a tela de geração de *composite application* com os serviços já previamente cadastrados.

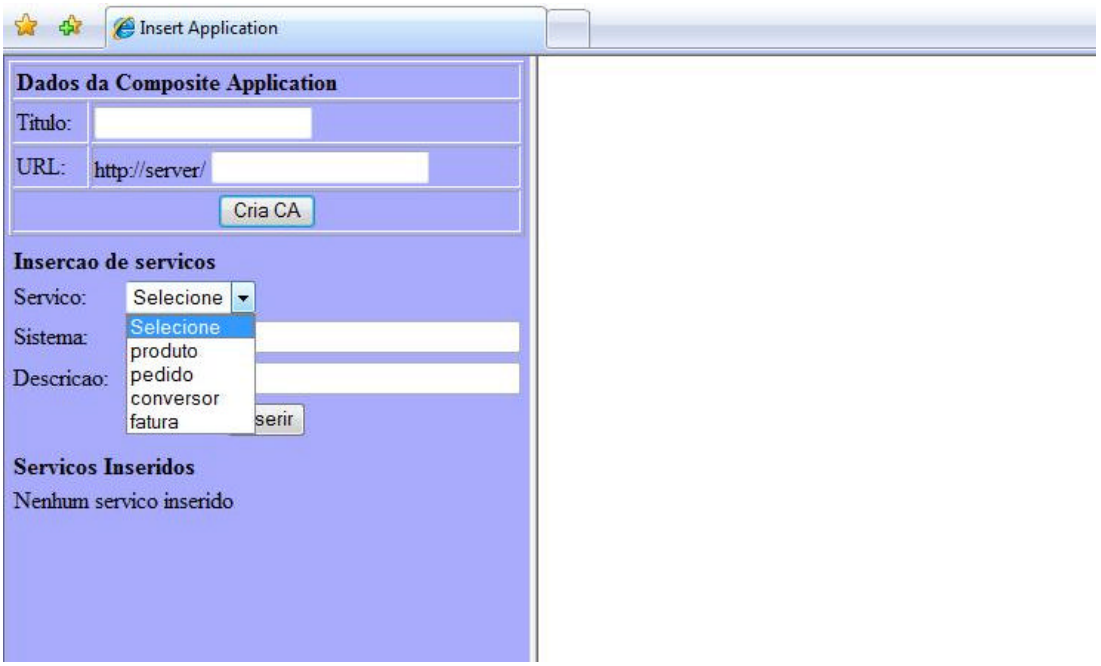

Figura 6.3 – Tela de geração de *composite application* com os serviços cadastrados no ambiente.

A figura 6.4 apresenta os serviços selecionados e que foram adicionados na *composite application*. Os serviços disponíveis de produto, conversor de unidades e pedido foram adicionados para que sejam testados posteriormente.

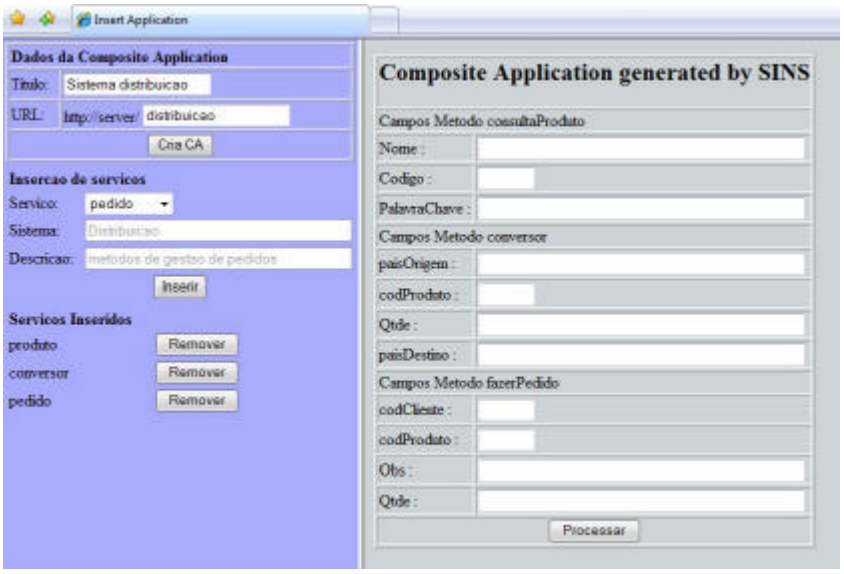

Figura 6.4 – Tela de geração de *composite application* com os serviços cadastrados no ambiente.

Após clicar no botão "Criar CA", a mensagem de confirmação foi apresentada na tela, conforme apresentado na figura 6.5.

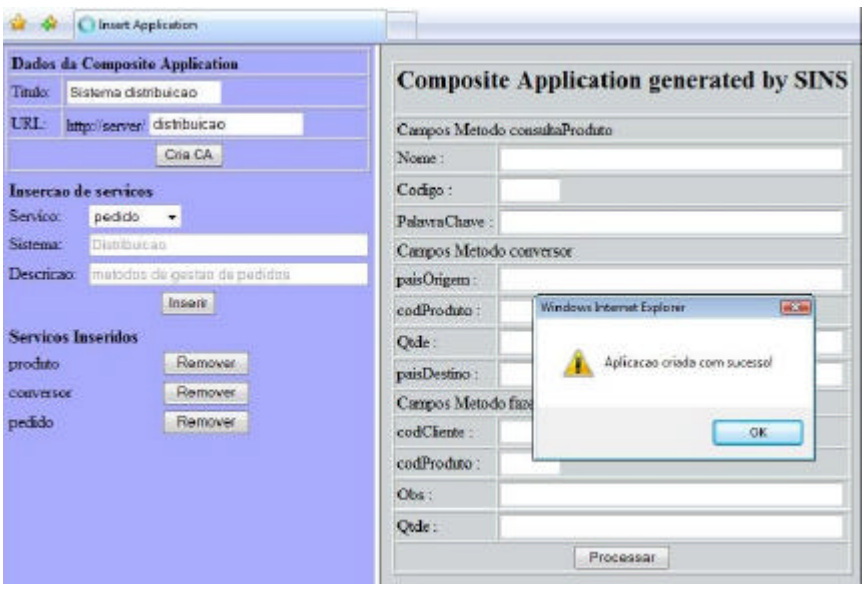

Figura 6.5 – Tela de confirmação dos serviços cadastrados para geração da *composite application*.

A figura 6.6 apresenta a *composite application* acessada através do endereço da url informado no momento da geração.

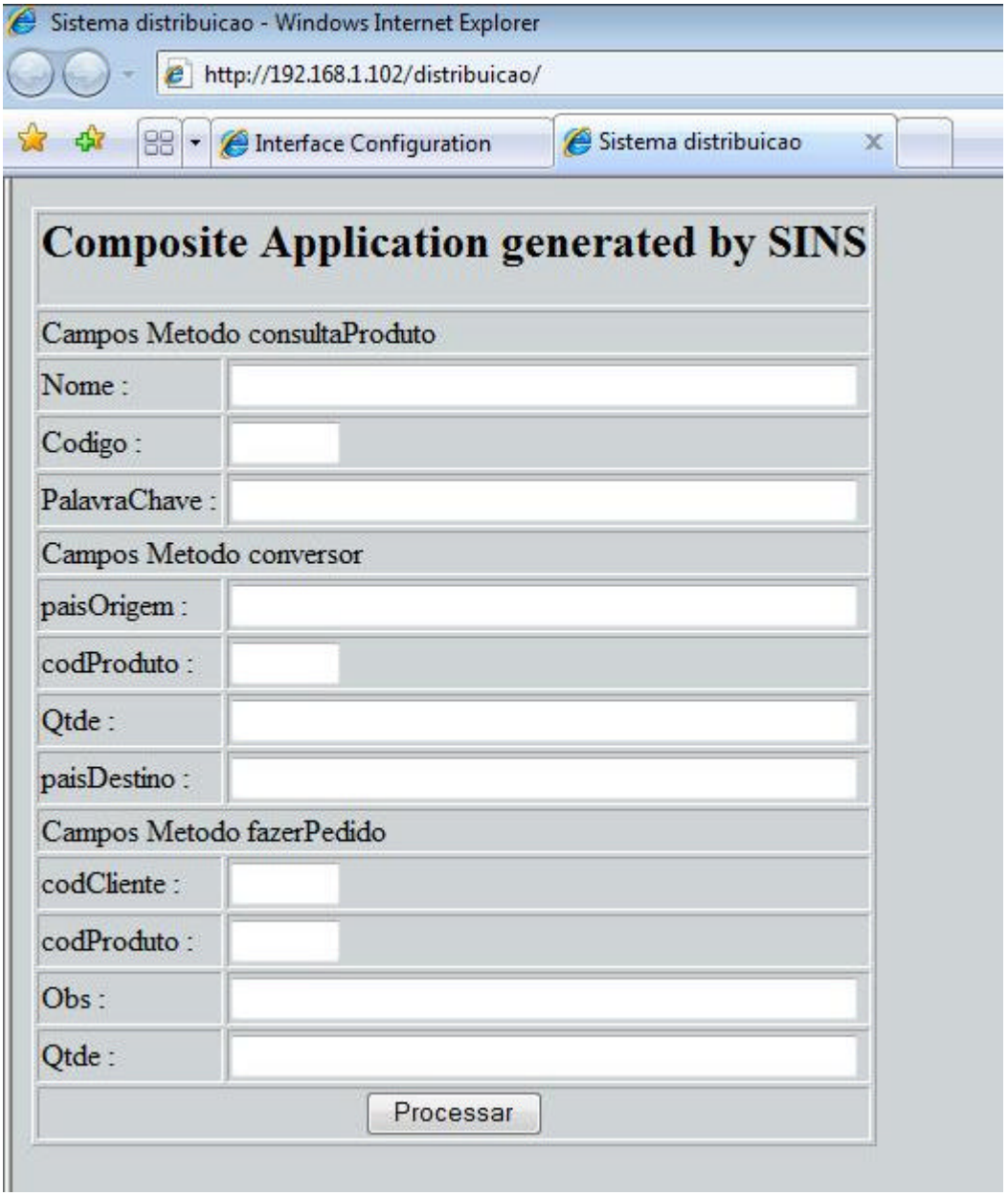

Figura 6.6 – Acesso a composite application gerada.

O arquivo XML de definição da interface gerado pelo ambiente é apresentado na figura 6.7. Nele constam as informações referentes à *composite application* e os métodos que estarão disponíveis no momento da chamada.

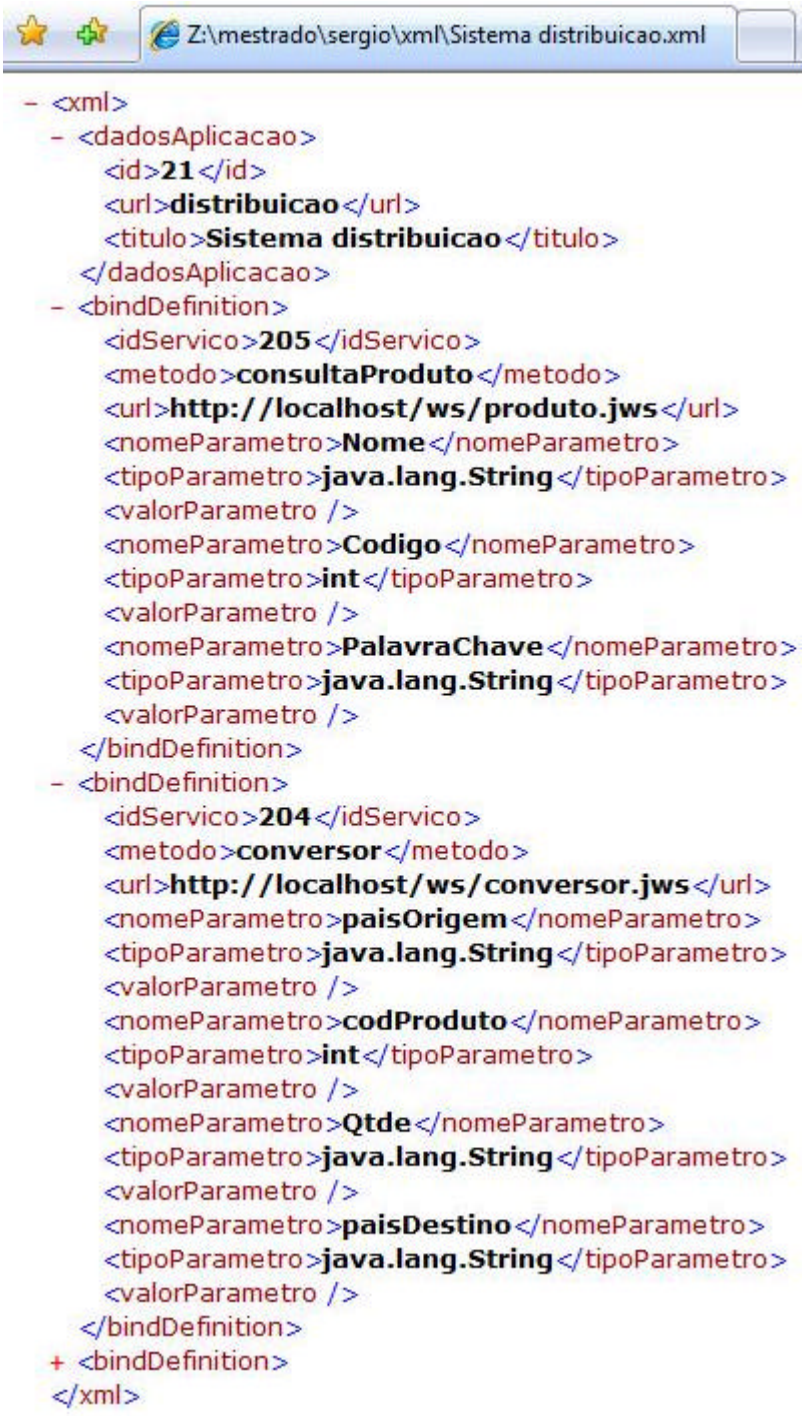

Figura 6.7 – Arquivo XML de definição da interface.

Após confirmar a inclusão dos valores nos campos referentes aos métodos do serviço é gerado um retorno com os dados referentes ao acesso aos serviços, conforme apresentado na figura 6.8.

Figura 6.8 – Retorno dos serviços disponíveis na *composite application*.

6.1.4 Comparativo: Implementação SOA x SINS

Para visualização das diferenças entre a implementação efetuada em SOA e a implementação utilizando o ambiente SINS, as WBSs referentes aos dois projetos são apresentadas nas figuras a 6.9 e 6.10:

Nome da tarefa	Duration	Resource Names	Cost	Work
Projeto	**129,91 days**		**R$ 98.505,57**	**2.377,22 hrs**
Pré-Planejamento	**0,5 days**		**R$ 710,90**	**12 hrs**
Planejamento	**6,88 days**		**R$ 4.387,77**	**95 hrs**
Especificação	**55,57 days**	**Analista de Sistemas**	**R$ 35.368,17**	**1.059,57 hrs**
Desenvolvimento	**30 days**	**Gerente de Projeto -**	**R$ 16.404,00**	**312 hrs**
Desenvolvimento conversor de unidades	14 days	Programador[75%]	R$ 4.200,00	84 hrs
Desenvolvimento dos Web Services	**14 days**		**R$ 4.200,00**	**84 hrs**
Sistema de Pedidos	6 days	Programador[75%]	R$ 1.800,00	36 hrs
Emissão de faturas	3 days	Programador[75%]	R$ 900,00	18 hrs
Conversor de unidades	2 days	Programador[75%]	R$ 600,00	12 hrs
Consulta Produto	3 days	Programador[75%]	R$ 900,00	18 hrs
Desenvolvimento das interfaces	**16 days**		**R$ 4.800,00**	**96 hrs**
Sistema de Pedidos	8 days	Programador[75%]	R$ 2.400,00	48 hrs
Emissão de faturas	3 days	Programador[75%]	R$ 900,00	18 hrs
Conversor de unidades	2 days	Programador[75%]	R$ 600,00	12 hrs
Consulta Produto	3 days	Programador[75%]	R$ 900,00	18 hrs
Testes	**36,72 days**	**Analista de Testes[75**	**R$ 25.323,46**	**562,22 hrs**
Entrega	**12,32 days**	**Gerente de Projeto -**	**R$ 8.459,17**	**206,92 hrs**
Encerramento	**5,5 days**	**Gerente de Projeto -**	**R$ 7.852,11**	**129,5 hrs**

Figura 6.9 – Project contendo a WBS do projeto original.

Nome da tarefa	Duration	Resource Names	Cost	Work
Projeto	**116,5 days**		**R$ 93.900,11**	**2.287,25 hrs**
Pré-Planejamento	**0,5 days**		**R$ 710,90**	**12 hrs**
Planejamento	**6,88 days**		**R$ 4.387,77**	**95 hrs**
Especificação	**55,57 days**	**Analista de Sistemas**	**R$ 35.368,17**	**1.059,57 hrs**
Desenvolvimento	**16,58 days**	**Gerente de Projeto -**	**R$ 11.798,54**	**222,03 hrs**
Deploy do SINS + mapeamento dos Web Services no ambiente	2 days	Arquiteto de Software -	R$ 902,16	12 hrs
Desenvolvimento conversor de unidades	14 days	Programador[75%]	R$ 4.200,00	84 hrs
Desenvolvimento dos Web Services	**15,83 days**		**R$ 4.750,00**	**95 hrs**
Sistema de Pedidos	7 days	Programador[75%]	R$ 2.100,00	42 hrs
Emissão de faturas	3,33 days	Programador[75%]	R$ 1.000,00	20 hrs
Conversor de unidades	2,17 days	Programador[75%]	R$ 650,00	13 hrs
Consulta Produto	3,33 days	Programador[75%]	R$ 1.000,00	20 hrs
Desenvolvimento das interfaces	**0,75 days**		**R$ 175,28**	**4,5 hrs**
Sistema de Pedidos	0,33 days	Programador HTML[75%	R$ 77,90	2 hrs
Emissão de faturas	0,17 days	Programador HTML[75%	R$ 38,95	1 hr
Conversor de unidades	0,08 days	Programador HTML[75%	R$ 19,48	0,5 hrs
Consulta Produto	0,17 days	Programador HTML[75%	R$ 38,95	1 hr
Testes	**36,72 days**	**Analista de Testes[75**	**R$ 25.323,46**	**562,22 hrs**
Entrega	**12,32 days**	**Gerente de Projeto -**	**R$ 8.459,17**	**206,92 hrs**
Encerramento	**5,5 days**	**Gerente de Projeto -**	**R$ 7.852,11**	**129,5 hrs**

Figura 6.10 – Project contendo a WBS do projeto usando SINS.

Em ambas as WBSs mostradas, a coluna *Work* representa o tempo total, em horas, levado para se executar aquela tarefa. A coluna *cost* representa o custo para execução daquela tarefa, sendo obtido pelo resultado da multiplicação do número de horas com o custo do profissional envolvido (vide tabela 6.1).

No grupo de tarefas "Desenvolvimento" da figura 6.10, foi incluído uma nova tarefa chamada "*Deploy* do SINS + mapeamento dos *Web Services* do ambiente", essa tarefa contém o número de horas que foram necessários para instalação e configuração do ambiente para uso com essa aplicação. Importante explanar que nesta tarefa estão inclusas as instalações de todos os componentes necessários pelo ambiente, tendo sido 1 hora para o Tomcat, 2 horas para o Oracle 10g XE, 2 horas para o AXIS e 3 horas para o jUDDI, restando então 1 hora para a instalação do SINS propriamente dito e 3 horas para o mapeamento dos *Web Services* do ambiente.

A tarefa "desenvolvimento conversor de unidades" consistiu na programação de um método que fizesse a conversão de unidades e medidas entre países e produtos. O método em questão foi usado neste estudo de caso sem nenhuma alteração.

As tarefas pertencentes ao grupo de tarefas chamado "Desenvolvimento dos *Web Services*" referem-se aos tempos de desenvolvimento para os serviços em questão. Para o

estudo de caso utilizou-se os serviços desenvolvidos no projeto original, fazendo pequenas alterações de validação de dados quando foi necessário.

As tarefas pertencentes ao grupo de tarefas chamado “Desenvolvimento das Interfaces” referem-se aos tempos de desenvolvimento das *composite applications* que consumem os *Web Services* desenvolvidos. Em todas estas tarefas foi usado o SINS, não se utilizando nada do projeto original, sendo estes itens o principal objeto de comparação deste estudo de caso.

Assim como as WBSs, a tabela 6.2 apresenta medições de custo e tempo efetuadas em tempo de desenvolvimento dos projetos original e com uso do SINS, porém com as informações consolidadas por tipo de tarefa.

Tabela 6.2 – Informações do projeto original x projeto com SINS.

	Tempo total (horas)	**Tempo Interface (horas)**	**Custo Total**	**Custo Interface**
Modo Usual	312	96	R$ 16.404,00	R$ 4.800,00
Com SINS	222	4,5	R$ 11.798,00	R$ 175,28

Para salientar a diferença na comparação do tempo total do desenvolvimento, a figura 6.11 mostra, em formato de gráficos, essa comparação.

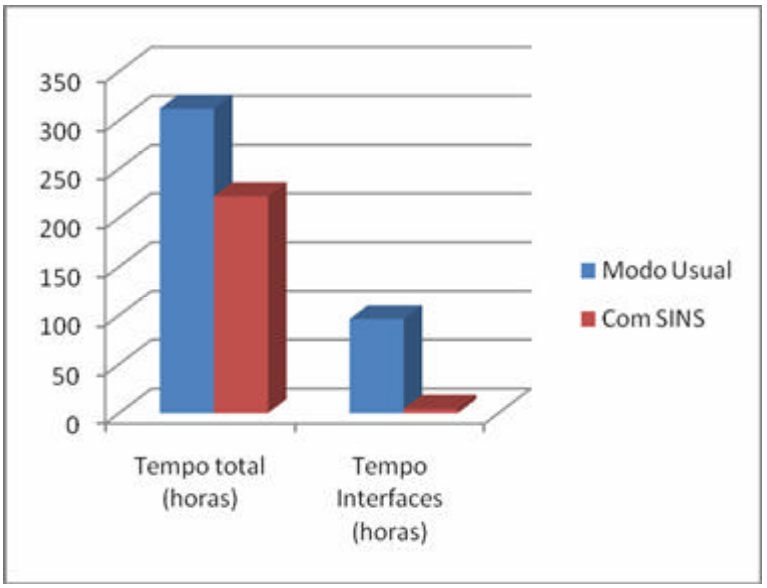

Figura 6.11 – Comparativo de tempo.

Para salientar a diferença na comparação do custo total do desenvolvimento, a figura 6.12 mostra, em formato de gráficos, essa comparação.

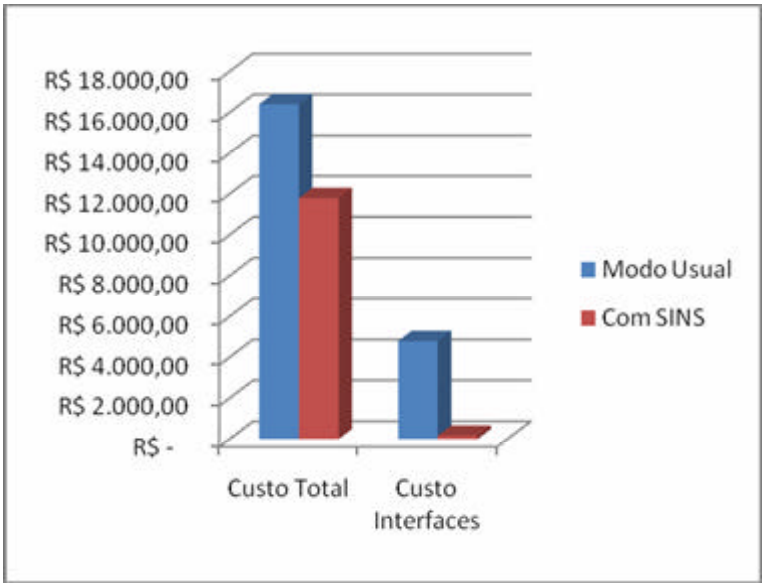

Figura 6.12 – Comparativo de custo.

Comparando a implementação original do projeto e o uso do SINS, pode-se constatar que:

- a quantidade de horas necessárias para desenvolvimento das interfaces cai na proporção de 21 para 1 (horas), valor esse obtido pela divisão das 96 horas necessárias no projeto original pelas 4,5 horas necessárias com o uso do SINS;
- considerando as 11 horas adicionais de customizações necessárias nos *Web Services* para uso com o SINS (vide figura 6.9) ante as 84 horas necessárias para desenvolvimento dos *Web Services* no projeto original (vide figura 6.10), conclui-se que a quantidade de horas necessárias no desenvolvimento da camada de negócio subiu 14%, o que ocorreu pela necessidade do desenvolvimento de um maior numero de validações no *Web Service*, uma vez que o SINS não as faz;
- uma vez que os serviços estejam cadastrados no ambiente ou disponíveis via UDDI, novas aplicações podem ser geradas por um usuário treinado na ferramenta, sem necessidade de codificação, o que não seria possível sem o uso do SINS;

- o número consolidado de horas de desenvolvimento foi reduzido em aproximadamente 27%, valor esse obtido comparando-se as 312 horas necessárias no projeto original com as 222 horas necessárias com o uso do SINS;
- o custo financeiro consolidado de desenvolvimento foi reduzido em aproximadamente 28%, custo obtido comparando-se os R$ 16.404,00 necessários para o desenvolvimento no projeto original ante os R$ 11.798,00 necessários com o uso do SINS (considerando os custos mostrados na tabela 6.1);
- as interfaces geradas pelo SINS ganham em agilidade de desenvolvimento, mas perde-se a possibilidade de customizações visuais;

  o SINS não tende a ter sua aplicação vantajosa em todas as situações. Em casos onde os *Web Services* tiverem métodos cujos parâmetros necessitaram uma validação complexa de dados e esta validação não estiver implementada nos serviços, o tempo necessário de customização nestes *Web Services* tende a tornar a utilização do SINS proibitiva;
- quanto maior for a confiabilidade da validação de dados contida nos serviços que estiverem sendo usados em conjunto com o SINS, maior tende a ser o ganho obtido com seu uso pois customizações tendem a ser menos necessárias e as interfaces geradas pelo SINS poderão ser usadas sem restrições quanto a isso.

7 CONSIDERAÇÕES FINAIS

Para elucidar qual a contribuição deste trabalho sob o contexto da arquitetura SOA, principalmente sob o viés das ferramentas já existentes, a tabela 3.1 será repetida abaixo, agora incluindo o SINS no comparativo.

Tabela 7.1 – Tabela comparativa.

	Trabalho baseado em Portlets	*IBM WebSphere*	*SAP XI*	*Microsoft Biztalk*	*Oracle SOA Suite*	*SINS*
Necessidade de codificação para consumir serviços	Sim	Sim	Apenas se forem serviços externos ao SAP	Sim	Sim	Não
Catálogo de serviços	Não	UDDI	UDDI ou diretório de serviços próprio.	UDDI	UDDI ou diretório de serviços próprio.	UDDI e banco de dados
Geração de *composite applications*	Não	Não	Não	Não	Sim, através de uma IDE de programação.	Sim, através de uma interface web.

Analisando as ferramentas da tabela 7.1, nota-se que todas necessitam de alguma codificação para fazer consumo de um *Web Service*. Salvo a ferramenta SAP XI que é capaz de fazer isso sem codificação caso o *Web Service* seja um serviço do próprio SAP.

Quanto ao critério "Catálogo de Serviços" há um trabalho acadêmico onde ele não existe e há ferramentas de empresas privadas fazendo uso de uma biblioteca de *Web Services* padrão (UDDI), sem que haja alguma interface ou facilitador para o usuário cadastrar ou encontrar um serviço específico. O SINS, por sua vez, disponibiliza uma interface ao usuário

que permite o cadastro e busca de serviços através de uma interface web, sendo possível o uso de UDDI sempre que houver um serviço de diretório deste tipo disponível.

Apesar do possível benefício proporcionado pelos critérios citados, acredita-se que a maior contribuição seja no que diz respeito à geração de *"Composite Applications"*. Enquanto todas as ferramentas disponíveis necessitam de codificação para criação de aplicações que consumam serviços (com uma pequena melhora para a ferramenta da Oracle, que possui um *Wizard* para essa tarefa, porém ainda exigindo codificação), independente da linguagem, o SINS faz isso de forma dinâmica e através de uma interface que permite a execução desta tarefa sem a necessidade de escrita de nenhuma linha de código e de forma acessível a uma pessoa que não seja da área de software. Inerente a isso pode-se notar no estudo de caso o evidente potencial de redução de custo e prazo em um projeto SOA.

Além dos critérios citados na comparação, o uso do SINS pode trazer outras vantagens com seu uso, como :

- Auxílio na identificação de serviços redundantes no ambiente, devido à indexação;
- Possibilidade de redundância entre serviços, caso um serviço esteja indisponível, o ambiente tentará utilizar o ambiente relacionado com o maior "*score*" no intuito de manter o funcionamento da *composite application*;
- No caso de alterações nos serviços (por exemplo, inclusão de um campo), a alteração do cadastro deste no SINS automaticamente atualizará todas as *composite applications* relacionadas;
- Tempo de desenvolvimento de *composite applications* reduzido se comparado ao tempo despendido por um programador.

7.1 Limitações do SINS

O SINS é um ambiente focado na geração de *composite applications* sem necessidade de codificação. Devido à arquitetura e objetivo da solução, nota-se as seguintes limitações:

- o SINS não possui nenhum tipo de interpretador de linguagens de script, não sendo possível nenhum tipo de customização na *composite application* gerada pelo ambiente;

- o SINS não possui nenhum mecanismo para otimização de carga ou *pooling*. Tendo em vista que o *handler* de todas as requisições para geração de *composite applications* é um único *servlet*, pode ser necessário configurar otimizações a nível de servidores ou *application server* quando as aplicações puderem ser submetidas a um grande número de acessos. (NLB, *pooling*, *clustering*, etc);
- o SINS não possui nenhum dispositivo de autenticação nas *composite applications*, assim como também não é capaz de autenticar-se contra *Web Services* que façam uso de algum aparato de segurança ou usem *WS-Security*;
- por basear-se no conceito de SOA, o SINS somente é compatível com *Web Services* que tenham sido desenvolvidos sob os conceitos de serviço (principalmente que sejam *stateless* e atômicos), pois ainda que serviços possam ser combinados em uma única interface, eles serão executados de forma individual e sem nenhum mecanismo para controle de sessão.

7.2 Trabalhos futuros

Como trabalhos futuros sugerem-se o uso do SINS em outros cenários para validação dos seus benefícios além do estudo de caso apresentado, assim como a complementação desse ambiente de forma que ele acompanhe as melhorias constantes da arquitetura SOA e do modelo de *Web Services* além do desenvolvimento de melhorias que cubram os pontos não abordados em tempo de desenvolvimento para o estudo de caso, sendo:

- capacidade de gerar interfaces baseadas em um modelo gráfico, o que permitiria organizar melhor as informações na tela além de gerar interfaces de melhor qualidade;
- mecanismos de autenticação e uso de *WS-Security;*
- mecanismos de otimização de desempenho, como o uso de *threading* no *servlet* gerador de *composite applications*;
- capacidade de exportar as *composite applications* respeitando os padrões de *portlets*, o que permitira se uso diretamente dentro de grande parte das ferramentas de portal que existem hoje;
- melhoria da parte de busca, catalogação e otimização de serviços. Seja aumentando e melhorando a taxonomia do banco de dados, seja integrando o ambiente a serviços UDDI distribuídos, permitindo a consulta a mais de um repositório simultaneamente.

8 REFERÊNCIAS BIBLIOGRÁFICAS

[Banerjee, 2007] Banerjee, A. Building Office Business Applications. The architecture journal. Input for Better Outcomes. Journal 10. Microsoft. ARC 098-107185.

[Brown et al., 2002] Brown, A.; Johnston, S.; Kelly, K.. Using Service-Oriented Architecture and Component-Based Development to Build Web Service Applications. A Rational Software White Paper. 2002.

[Chappell, 2004] Chappel, D.. Understanding BPM Servers. David Chappell & Associates. Microsoft Corporation. October, 2004.

[Chen et al., 2006] Chen, Y., Fan, C., Tsai, W.T., Paul, R., Chung, J..Architecture Classification for SOA-Based Applications. Proceedings of the Ninth IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing. 2006.

[Crespo, 2000] Crespo, S.. Composition in WebFrameworks. DSc. Thesis. Rio de Janeiro: PUC, 2000 (in Portuguese).

[Erl, 2006] Erl, T. Service-Oriented Architecture: Concepts, Technology, and Design. Prentice Hall. 2006.

[Gartner, 2003] Gartner Research. Service-Oriented Architecture Scenario. April, 2003. Disponível em: <http://www.gartner.com/resources/114300/114358/114358.pdf>. Acesso em: 5 de maio 2007.

[Gartner, 2006] 2006 CIO Agenda, November 2006.

[Graham et al., 2002] Graham, S.; Simeonov, S.; Boubez, T.; Davis, D.; Daniels, G.; Yuichi, N. and Neyama, R. Building Web Services with Java: making sense of XML, SOAP, WSDL, and UDDI. Indianapolis: Sams, 2002. 581p.

[Hansen, 2003] Hansen, R. P.. GlueScript: uma linguagem específica de domínio para composição de web services. São Leopoldo, 2003. 89 fl. Dissertação (Mestrado) – Universidade do Vale do Rio dos Sinos, Centro de Ciências Exatas e Tecnológicas, Programa Interdisciplinar de Pós-Graduação em Computação Aplicada.

[Heuser, 2004] Heuser, L.. Enterprise Services Architecture & Semantic Web Services. SAP Research. SAP Group. 2004.

[IBM, 2005] IBM Corporation. IBM SOA Foundation: providing what you need to get started with SOA. Service oriented architecture solutions. White Paper, 2005.

[IBM, 2006] IBM Corporation. Increasing IT flexibility with IBM WebSphere ESB software.White Paper, 2006.

[Keen et al., 2004] Keen, M.; Acharya, A.; Bishop, S.; Hopkins, A.; Milinski, S.; Nott, C.; Robinson, R.; Adams, J.; Verschueren, P.. Patterns: Implementing an SOA Using an Enterprise Service Bus. IBM Redbooks, First Edition, July, 2004.

[Kreger, 2001] Kreger, H. Web Services Conceptual Architecture. IBM Software Group, May 2001.

[Link et al, 2006] Link, S., Jakobs, F., Neer, L., Abeck, S. Architecture of and Migration to SOA´s Presentation Layer, C&M Research Report, Universität Karlsruhe, 2006.

[Manolescu and Lublinsky, 2007a] Manolescu, D.; Lublinsky B.. Draft Service-Oriented Arquitecture Defined, do livro Enterprise Patterns - Services, Orchestration and Beyond, para ser publicado por Morgan-Kaufmann. Disponível em <http://orchestrationpatterns.com>. Acesso em: 22 de mar. 2007.

[Microsoft, 2005a] Microsoft Corporation. Business Activity Monitoring - BAM. Microsoft White Paper. April 2005.

[Microsoft, 2005b] Microsoft Corporation. Melhorias no tempo de execução do BizTalk Server 2006. Microsoft White Paper. November 2005.

[Microsoft, 2006] Microsoft Corporation. Learn About Service Oriented Architecture (SOA). December, 2006.

[Newcomer, 2002] Newcomer, E. Understanding Web Services. Independent Technology Guides. Dadid Chappell, Series Editor. 2002.

[OASIS, 2006] OASIS Reference Model for Service Oriented Architecture V 1.0. August 2, 2006. Disponível em <http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=soa-rm>. Acesso em: 22 de mar. 2007.

[Oellermann, 2001] Oellermann Junior, W.L. Architecting Web Services. New York: Apress, 2001. 654p.

[Oracle, 2005] Strategies for SOA Success. Ziff Davis Media Custom Publishing. December 2005.

[Oracle, 2006] Oracle SOA Suite. Quick Start Guide 10g (10.1.3.1.0). September 2006.

[Rheinheimer, 2004] Rheinheimer, L. R. WSAgent: Um Agente Baseado em Web Services para promover a Interoperabilidade entre Sistemas Heterogêneos no Domínio da Saúde. Tese de Mestrado pela Unisinos. 2004.

[Rogers and Hendrick, 2005] Rogers, S.; Hendrick, S. D.. Oracle Builds Comprehensive SOA Platform. IDC White Paper. January 2005.

[Roy and Ramanujan, 2001] Roy, J.; Ramanujan, A. Xml Shema Language: Taking XML to The Next Level, IT Pro, vol. 3, No. 2, March/April. 2001, pp. 37-40.

[Sampaio, 2006] Sampaio, C. SOA e Web Services em Java. Rio de Janeiro: Editora Brasport 2006.

[SAP, 2006] SAP Group. SAP NetWeaver: Using IT practices to bridge the worlds of business and IT. November, 2006.

[SAP, 2007] SAP Group. Transform your organization with enterprise service-oriented architecture. January, 2007.

[Schimidt, 2003] Schimidt, R. Composite Applications for the Enactment of Dynamic Inter-Organizational Business Processes, in Proceedings of the 1st Int. Workshop "Component Based Business Information Systems Engineering" (Internal Conference on Object Oriented Information Systems). Genova September 2nd, 2003.

[Sehmi, 2006] Sehmi, A. Schwegler, B.. Modeling and Messaging for Connected Systems. The architecture journal. Input for Better Outcomes. Journal 7. Microsoft. ARC 098-105109.

[TIInside, 2007a] Ano 3, Número 22, Março de 2007. Página 18 a 22.

[TIInside, 2007b] Ano 3, Número 21, Janeiro de 2007. Página 03 e 33 a 22.

[IWeek, 2006] InformationWeek Ano 8, Número 168, Setembro de 2006. Página 28 a 33.

[W3C, 2007] Web Services architecture: W3C working group note. February. 2004. Disponível em: <http://www.w3.org/TR/ws-arch>. Acesso em: 19 de mar. 2007.