UNIVERSIDADE FEDERAL DE SANTA CATARINA

CENTRO TECNOLÓGICO

DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA

CURSO DE SISTEMAS DE INFORMAÇÃO

RODRIGO PEREIRA DA SILVA

AVALIAÇÃO DA SCA NO DESENVOLVIMENTO DE APLICAÇÕES BASEADAS EM SOA

FLORIANÓPOLIS

2007

UNIVERSIDADE FEDERAL DE SANTA CATARINA

CENTRO TECNOLÓGICO

DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA

CURSO DE SISTEMAS DE INFORMAÇÃO

RODRIGO PEREIRA DA SILVA

AVALIAÇÃO DA SCA NO DESENVOLVIMENTO DE APLICAÇÕES BASEADAS EM SOA

Trabalho de Conclusão de Curso submetido à Universidade Federal de Santa Catarina como parte dos requisitos para a obtenção do grau de Bacharel em Sistemas de Informação

Prof. Renato Fileto - Orientador

FLORIANÓPOLIS

2007

RODRIGO PEREIRA DA SILVA

AVALIAÇÃO DA SCA NO DESENVOLVIMENTO DE APLICAÇÕES BASEADAS EM SOA

Trabalho de conclusao de curso submetido a Universidade Federal de Santa Catarina como parte dos requisitos para obtencao do grau de Bacharel em Sistemas de informacao.

Orientador: Prof. Renato Fileto, Dr.
Universidade Federal de Santa Catarina
fileto@inf.ufsc.br

Banca examinadora

Prof. Frank Augusto Siqueira, Dr.
Universidade Federal de Santa Catarina
frank@inf.ufsc.br

Prof. Mário Antônio Ribeiro Dantas, Dr.
Universidade Federal de Santa Catarina
mario@inf.ufsc.br

Sumário

LISTA DE FIGURAS

LISTA DE TABELAS

RESUMO

A crescente demanda por aplicações orientada a serviços aumenta o interesse pela criação de modelos e especificações que unifiquem as melhores práticas no desenvolvimento de sistemas baseados em SOA.

A SCA (*Service Component Architecture*) é um conjunto de especificações para facilitar a criação e a composição de serviços em alto nível para a construção de aplicações que utilizam a abordagem SOA (*Service Oriented Architecture*). Este trabalho avalia a SCA como uma tecnologia para auxilar o desenvolvedor na implementação de aplicações orientada a serviços, através de um estudo de caso . Além disso, o trabalho aprensenta a especificação *Assembly Model* da SCA, que especifica a composição de serviços.

ABSTRACT

The increase demand for service oriented systems make that vendors focus their efforts in the creation of models and specifications that unifies the best pratics on SOA development.

SCA (Service Component Architecture) is a set of specifications to facilitate the creation and composition of services on building applications that employ SOA (Service Oriented Architecture). This work assesses how SCA makes the developer work easier on implementing SOA applications, through a case study. Besides, it introduce the Assembly Model specifications that specify the composition of services.

1. Introdução

1.1 Descrição do Problema

O mundo corporativo atual dispõe de uma imensa gama de plataformas, linguagens de programação e formas de armazenamento de dados para desenvolver os seus sistemas. Esse cenário favorece a criação de software utilizando diversas tecnologias distintas. Todavia, as corporações estão vendo cada vez mais a necessidade de suas aplicações interoperarem com sistemas legados e sistemas de terceiros, tais como fornecedores, clientes e parceiros.

O conceito de SOA (*Service Oriented Architecture*) surgiu para permitir a interoperabilidade de sistemas heterogêneos. SOA permite conectar as unidades funcionais de uma aplicação distribuída, implementadas como serviços com interfaces pré-definidas e auto-descritivas, para possibilitar acoplamento dinâmico. Ela tem sido largamente utilizada pelas empresas para permitir a troca de informações de negócios entre os seus sistemas e desses com sistemas externos.

Apesar dos benefícios da SOA, o desenvolvimento de aplicações utilizando esse tipo de arquitetura não é feito de forma trivial. Considerando que existem diversas linguagens de programação, *frameworks,* plataformas, e bibliotecas para conectar e prover serviços, o número de alternativas tecnológicas para implementação de aplicações SOA é alto. Conseqüentemente, o desenvolvedor deixa de focar na implementação da lógica de negócio do serviço para tratar dos detalhes de implementação em diferentes tecnologias.

Recentemente, 18 grandes empresas de TI, entre elas IBM, Sun e Oracle, juntaram-se para definir um modelo para o desenvolvimento de aplicações que utilizam a abordagem SOA. Como resultado disso, elas criaram a SCA (*Service Component Architecture*). A SCA consiste em um conjunto de especificações para o desenvolvimento de aplicações baseadas em SOA. Ela visa auniformizar a construção e a concepção de sistemas em SOA, utilizando e estendendo as melhores práticas e abordagens aplicadas atualmente. A SCA almeja que as corporações economizem tempo e

dinheiro com a sua utilização, uma vez que a sua API *(Application Programming Interface)* é simples e pequena, o que proporciona uma curva de aprendizado curta e minimiza os erros em sua aplicação.

1.2 Objetivos do trabalho

Esse trabalho propõe uma avaliação da facilidade de implementação de aplicações em SOA utilizando a SCA. Será desenvolvido como estudo de caso um sistema para busca de informações de empresas que possuem ações negociadas na BOVESPA utilizando os modelos da SCA e a sua implementação Tuscany. Dessa forma, pode-se avaliar na prática os seguintes benefícios da SCA:

- O desenvolvedor não precisar conhecer as APIs que invocam e provêm serviços.
- Separação dos detalhes de infra-estrutura (segurança, qualidade de serviço e autenticação) da implementação dos componentes que consomem e provêm serviços.

Os objetivos específicos desse trabalho são:

Conhecer as especificações e conjuntos de modelos da SCA:

a. SCA *Assembly Model*
b. SCA *Java Implementation Model Specification*
c. SCA *Policy Framework*
d. SCA *Bindings*.

2. Detectar eventuais limitações da SCA
3. Avaliar a complexidade envolvida no uso da SCA e do Tuscany
4. Avaliar benefícios da utilização da SCA em um estudo de caso

1. Arquitetura Orientada a Serviços

Uma *arquitetura de software* é um conceito abstrato que possibilita diversas interpretações. Segundo a ANSI/IEEE, uma arquitetura de software trata de como os componentes fundamentais de um sistema se relacionam intrinsecamente e extrinsecamente [1]. Serviços são os componentes fundamentais de uma arquitetura orientada a serviços. Não há um consenso quanto à definição de serviços, porém, eles podem ser considerados uma referência a um componente de software binário baseado num contrato [2].

Um serviço pode ter as seguintes características [3]:

- executar uma determinada operação de negócio ou acessar um banco de dados para prover informações de negócio
- acessar outros serviços, e com determinada tecnologia de execução, acessar um sistema legado e responder a diferentes tipos de requisições
- é relativamente independente de outros softwares, onde mudanças no programa cliente acarretam poucas ou nenhumas mudanças para o serviço; e mudanças na lógica interna do serviço acarretam poucas ou nenhuma mudança para o programa cliente;

Sistemas baseados em SOA podem ser classificados como sistemas cooperativos abertos distribuídos, uma vez que, os serviços possuem as seguintes funções: cooperam para atingir um determinado objetivo, passam a compor o sistema dinamicamente e executam em máquinas diferentes.

A arquitetura orientada a serviços é de grande valia para as empresas que desejam que seus sistemas interajam com aplicações de parceiros, fornecedores e clientes. Além disso, as empresas não necessitam desenvolver novos sistemas para suprir suas necessidades de negócios, uma vez que, as funções de negócio dos sistemas existentes podem ser utilizadas como serviços ou

composição de serviços de determinado domínio de negócio. Dessa forma, elas podem atender às suas estratégias mais rapidamente, além de reduzir o orçamento de TI (Tecnologia da Informação).

As características relevantes de uma arquitetura orientada a serviços são:

- Reuso “Caixa-Preta”
- Distribuição
- Heterogeneidade ambiental
- Composição
- Coordenação
- Dinamismo e adaptabilidade
- Estados
- Sincronia

Abaixo são discutidas cada uma dessas características.

2.1 Reuso “Caixa-Preta”

O principal objetivo do reuso de *software* é economizar tempo de desenvolvimento e recursos humanos, ou seja, dinheiro. O reuso pode ser classificado quanto ao conhecimento do componente de software: o reuso caixa-branca e o caixa-preta [2].

No reuso caixa-branca o desenvolvedor conhece a lógica interna do componente enquanto no caixa-preta ela é desconhecida. A herança de uma determinada classe é um exemplo clássico de reuso caixa-branca, onde o desenvolvedor necessita compreender como a classe pai foi construída. A utilização de uma Interface Java é um exemplo de reuso caixa-preta, em que é necessário um contrato para a reutilização de um determinado componente de software.

Os serviços podem ser considerados como uma caixa-preta, onde o consumidor do serviço não necessita saber a forma de implementação da função de negócio que está sendo provida pelo serviço. A arquitetura orientada a serviços utiliza largamente esse conceito e é dirigida por contratos, onde são estabelecidos os relacionamentos entre os componentes (serviços) da aplicação.

2.2 Distribuição

As aplicações orientadas a serviços fazem uso do conceito de distribuição, uma vez que os serviços podem estar sendo executados em máquinas diferentes. Com o uso da distribuição é possível acessar sistemas legados, de fornecedores e de clientes de forma transparente.

2.3 Heterogeneidade Ambiental

O desenvolvimento de software dispõe de uma imensa gama de linguagens de programação, sistemas operacionais e hardware diferentes. Isso favorece a construção de sistemas em ambientes heterogêneos. A comunicação e troca de mensagens entre esses sistemas não são feitas de uma forma trivial, dadas as possíveis incompatibilidades entre as linguagens e plataformas dessas aplicações.

SOA permite que sistemas heterogêneos se comuniquem e troquem mensagens em um formato padrão. De maneira geral, os componentes (serviços) são executados em plataformas distintas e os sistemas são desenvolvidos em diferentes linguagens de programação e *frameworks*.

2.4 Composição

Os sistemas desenvolvidos com linguagens procedurais eram tipicamente fechados, ou seja,

era necessário criar uma outra aplicação para uma extensão do sistema e compô-la para ficar transparente ao usuário.[4]. Com o advento da orientação a objetos e o encapsulamento, foi possível criar aplicações distintas com apenas algumas alterações de código. Porém, a orientação a objetos não resolve o problema de composição de aplicações, onde seria possível a composição de componentes existentes e testados em novas aplicações sem nenhuma alteração de código.

Em SOA, os serviços podem ser vistos como componentes e as interfaces como os contratos estabelecidos para o uso dos componentes (reuso caixa-preta). Dessa forma, é possível a composição de serviços em novas aplicações para um determinado domínio de negócio.

2.5 Coordenação de serviços

A diferença entre a coordenação e a composição é que a coordenação se refere a sincronização, temporização e ordenação, já a composição trata da combinação dos elementos em um todo com funcionalidade definida pela composição. A coordenação de serviços trata como os serviços trabalham juntos para atingir um determinado objetivo. Existem duas formas de coordenação: orquestração ou coreografia. [3]. A orquestração cuida da execução do processo de negócio. Nela, um serviço coordena todos os outros serviços através de troca de mensagens, onde os serviços enviam uma mensagem para o serviço coordenador e este, dependendo do conteúdo da mensagem, toma a decisão de qual serviço será invocado. A orquestração sempre se refere ao controle da execução dos processos de uma das partes envolvidas no processo como um todo. A figura 2.1 ilustra isso.

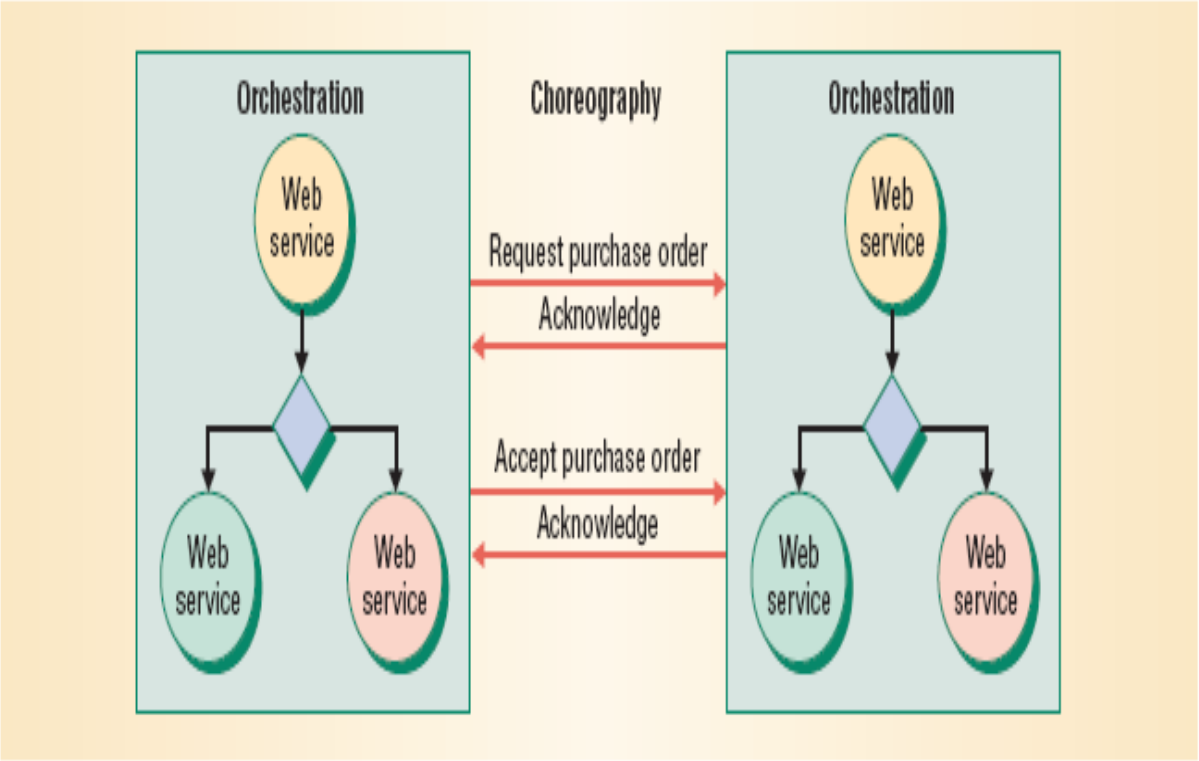

Figura 1 – Ilustração de coordenação de serviços [4]

A coreografia de serviços, por outro lado, consiste na especificação das seqüências das interações entre os serviços, de forma que eles cooperem harmoniosamente. Ela cuida da seqüência de mensagens que podem ser trocadas entre partes diferentes do processo.

Para facilitar o desenvolvimento da coordenação de serviços foram criadas algumas linguagens, entre elas a BPEL (*Business Process Execution Language*) [5] para a orquestração e a WS-CDL (*Web Service Choreography Description Language*) [6] para o coreografia.

2.6 Dinamismo e adaptabilidade

As regras de negócio dos sistemas estão em constante evolução. Dessa forma, as aplicações

devem ser estruturadas para “acolher as mudanças” [7]. Em uma arquitetura orientada a serviços, o dinamismo está sempre presente, uma vez que um serviço é descoberto no registro de serviços em tempo de execução. Além disso, a escolha do serviço a ser invocado é feita durante a execução do sistema e pode haver diferentes opções de serviços disponíveis.

Uma mudança no processo de negócio refletirá numa alteração na composição e/ ou coordenação dos serviços. Isso pode representar um novo processo, a inclusão de um serviço ou até mesmo a exclusão de um processo inteiro. Então, os sistemas devem possuir uma estrutura que se adapte às mudanças de maneira mais simples possível, onde poucas alterações devam ser feitas.

2.7 Estados

O estado de um componente pode pertencer a duas classes: sessão ou persistente. O estado persistente ocorre quando o dado enviado por uma das partes durante uma comunicação pode ser recuperado a qualquer momento [8]. Um exemplo é o envio de uma mensagem com os dados a serem persistidos num comando SQL e o envio de outro comando para recuperar os dados.

O estado de sessão corresponde ao estado dos componentes somente durante a comunicação ou sessão, uma vez terminada, os dados e estados são liberados. A comunicação assíncrona é um exemplo de dados que permanecem apenas na sessão.

Na arquitetura orientada a serviços, os dados tipicamente permanecem somente durante a sessão e são enviados através do protocolo SOAP (*Simple Object Access Protocol*) [9].

2.8 Sincronia

A troca de mensagens, os seus modos de transmissão e a semântica da comunicacão são partes fundamentais em sistemas distribuídos uma vez que são a única maneira que os componentes

possuem para se comunicar e sincronizar as suas ações[10]. A troca de mensagem pode ser síncrona ou assíncrona.

A comunicação síncrona consiste no bloqueamento do processamento até que a mensagem enviada pelo emissor seja respondida pelo receptor. Já na comunicação assíncrona o processamento não é bloqueado.

2.9 Principais problemas em SOA

Os principais problemas/dificuldades atuais no desenvolvimento de sistemas orientados a serviços são listados abaixo [3]

- Muitas vezes não é trivial converter um componente em um Web Service
- Dispêndio do tempo dos desenvolvedores na construção de um *middleware* para a conexão entre os serviços. Por exemplo, para uma aplicação consumir um serviço, é necessário que seja construído um código para acesso a esse serviço
- Reuso e composição de serviços precisam ser feitos manualmente pelos programadores
- Ausência de mecanismos para a identificação das informações que os serviços disponibilizam
- Dificuldade em descobrir os serviços para suprir uma dada necessidade no catálogo de serviços

3.SCA

A SCA (*Service Component Architecture*) é uma proposta de padronização para a composição e a implantação de aplicações orientadas a serviços. Atualmente ela está na versão 1.0 e é suportada por um grupo de empresas, denominado OSOA (*Open Service Oriented Architecture*). O objetivo desse grupo é a manutenção e o desenvolvimento de tecnologias que convirjam para os interesses mútuos no desenvolvimento de aplicações baseadas em SOA. Porém, OSOA não é um orgão de padronização. A versão 1.0 da SCA foi enviada para padronização no consórcio internacional OASIS (*Organization for Advancement of Structured Information Standards*) que dirige o desenvolvimento, convergência e adoção de padrões para *e-business* [11].

3.1 Elementos da SCA

A SCA é composta pelos seguintes elementos[3]:

Assembly Model – Consiste num conjunto de modelos e especificações para a composição de serviços.

Especificações de Implementação de Componentes – Define como implementar os serviços em uma gama de linguagens como Java, C++, BPEL, PHP, entre outras

Especificação de *Binding* - Define como acessar os componentes através de tecnologias como Web Services, JMS (Java *Message Servic*e), RMI-IIOP, REST, entre outras.

Policy Framework – Consiste na adição de detalhes de infra-estrutura na aplicação, como segurança, controle de transação, qualidade de serviço *(Quality of Service),* entre outros.

3.1.1 – Assembly Model

A *Assembly Model* consiste num conjunto de modelos para a composição de serviços

fortemente ou fracamente acoplados [12]. De maneira geral, os serviços com forte acoplamento pertencem ao mesmo módulo e os componentes com fraco acoplamento são de módulos diferentes. A configuração das composições de serviços são feitas em arquivos XML, onde são definidas as conexões dos componentes que formam a composição.

O elemento mais básico da SCA *Assembly Model* é o componente que consiste na implementação em uma determinada linguagem, como Java, C++ ou BPEL, de uma função de negócio. Outro elemento é o *service*, que consome ou provê as informações produzidas pelos componentes. A dependência da informação de um serviço por um outro serviço é chamada referência. O conjunto dos elementos acima formam as composições de serviços e o conjunto das composições de serviços formam um SCA *Domain*, que provê a funcionalidade de um determinado módulo do sistema.

Os próximos itens irão detalhar cada um dos elementos de um SCA *Domain* que é o principal artefato criado na fase de montagem.

3.1.1.1 *Services*

Os S*ervices* consistem nas operações que o componente fornece para o ambiente externo. Eles podem ser especificados de diversas maneiras, dependendo da linguagem que a implementação foi desenvolvida. Por exemplo, um componente implementado em Java, provavelmente utilizará uma Interface Java para descrever os seus *services*, enquanto um componente implementado em BPEL, utilizará o WSDL (*Web Services Description Language*).

A especificação de *service* abaixo utiliza uma Interface Java com *annotations.* Neste exemplo o componente provê o método getInfoEmpresa para ser invocado. A anotação @Remotable indica que o componente pode ser invocado por clientes remotos.

```
@Remotable
public interface BovespaInfo {
        public String getInfoEmpresa();
```

```
}
```

3.1.1.2 *Reference*

A *reference* especifica as operações que o componente necessita invocar para realizar determinada tarefa. Uma *reference* pode ser usada para acessar um serviço como por exemplo: um Web Service ou um EJB (Enterprise Java Bean). A *reference* é composta basicamente por uma interface que possui todos os métodos que o componente irá invocar [3].

No exemplo abaixo SetorService que está destacado em negrito, é uma *reference* do componente EmpresaServiceComponent. A implementação dessa reference é em Java.

```
<component name="EmpresaServiceComponent">
<service name="EmpresaService">
            <interface.wsdl
interface="http://bovespainfo#wsdl.interface(BovespaInfo)" />
            <binding.ws uri="http://localhost:8085/BovespaInfoService"/>
         </service>
    <implementation.java class="helloworld.EmpresaServiceImpl" />
           <reference name="setorService" target="SetorService"/>
    </component>

 <component name="SetorService">
        <implementation.java class="helloworld.SetorServiceImpl"/>
 </component>
```

3.1.1.3 *Property*

As *property* são atributos dos serviços que correspondem a valores de variáveis ou entidades que são utilizadas pelas implementações dos componentes em sua lógica de negócio. Elas correspondem a dados externos da implementação, dessa forma, são desacopladas de uma implementação específica, permitindo que se troque uma implementação de forma simples, sem alteração na definição do serviço no SCA *Domain*. Um exemplo de *property* pode ser o tipo de moeda que uma implementação deva usar, no caso um componente usa o tipo real, e outro pode usar o tipo

dólar. Dessa forma, a implementação não está atrelada a um determinado tipo de moeda, o que provê o reuso de outras implementações de maneira fácil.

3.1.1.4 *Component*

O *component* é o elemento mais básico de uma composição de serviços, As suas 2 principais funções são consumir e prover serviços [3]. Ele basicamente corresponde à configuração de uma implementação em uma determinada linguagem de programação ou *framework,* tais como Java, EJB, C++ e BPEL.

O simbolo abaixo representa todos os elementos descritos até agora no diagrama da *Assembly Model.*

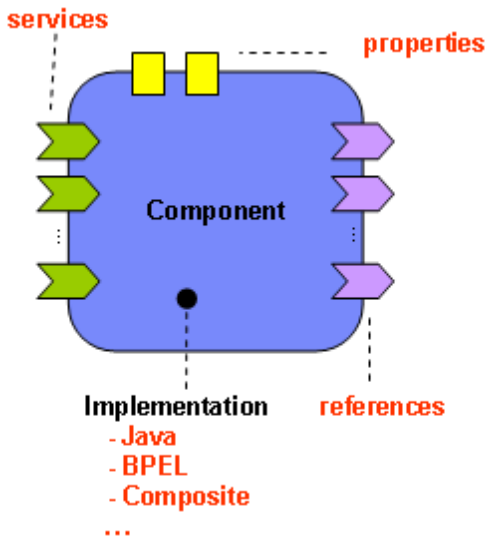

Figura 2 Símbolo de um SCA *component* [13]

O código abaixo mostra um exemplo da configuração de um *component para* prover informações de ações negociadas na BOVESPA. A tag <implementation.java> define que a sua implementação é em linguagem Java. A tag <service> denota que ele é consumido através da *service* AcoesBovespaService, que possui a interface Java chamada services.bovespa.interface.AcoesBovespaService, espeficicada pela tag <interface.java>. Esse componente é acessado através da tecnologia JMS, definido pela tag <binding.jms>. Ele invoca o serviço intraCorretoraService que é um Serviço Web, definido pela tag <binding.ws>. Além disso, ele possui uma *property currency* que especifica o tipo de moeda que a implementação do componente deve usar.

```
<component name="AcoesBovespaServiceComponent">
      <implementation.java class="services.bovespa.impl.AcoesBovespaServiceImpl">

      <service name="AcoesBovespaService">

            <interface.java
                  interface="services.bovespa.interface.AcoesBovespaService" />

            <binding.jms />
```

```
        </service>

        <reference name="intraCorretoraService"
              target="intraCorretoraService">

              <binding.ws>

        </reference>

        <property name="currency">real</property>

</component>
```

3.1.1.5 Composição de serviços

A composição de serviços consiste na ligação dos componentes para servir a um determinado fim. Esses componentes podem ser desenvolvidos para atender especificamente à aplicação ou podem ser sistemas existentes implementados em linguagens e plataformas distintas do componente consumidor. A composição desses componentes irá prover um serviço para determinada operação ou lógica de negócio do sistema. Com isso, a composição pode ser vista como um simples serviço. Os serviços compostos trabalhando mutuamente para atender a um determinando negócio constituem o sistema. A figura abaixo representa um diagrama da SCA para uma composição de serviços:

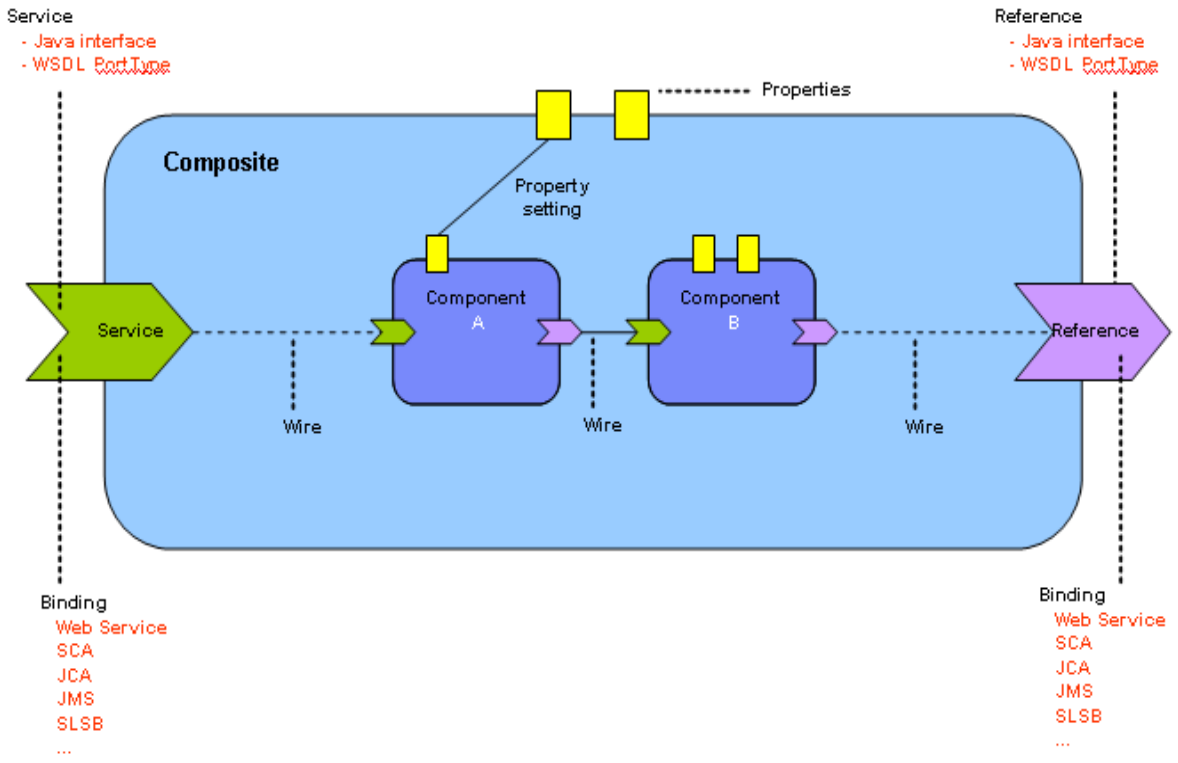

Figura 3 Ilustração de uma composição de serviços[13]

3.2 Fases do projeto de aplicações na SCA

O desenvolvimento de aplicações orientadas a serviços utilizando SCA é feito em 3 fases: Desenvolvimento, Montagem e Implantação [3].

3.2.1 Desenvolvimento

Após o desenvolvimento dos novos componentes ou a identificação dos componentes existentes da aplicação que implementarão as regras de negócio do sistema, é feita a configuração do comportamento de execução. Basicamente, isso é feito configurando arquivos XML, seguindo os modelos da SCA.

3.2.2 Montagem

Nesta etapa, o analista de negócio ou um desenvolvedor compõe um ou mais componentes para atender a uma determinada função de negócio. Uma composição de serviços pode ser executada em um processo de uma máquina ou os componentes da composição podem ser distribuídos em máquinas diferentes, independentemente de plataformas e sistemas operacionais dessas máquinas.

3.2.3 Implantação

Na implantação são feitas as seguintes tarefas [3]:

- montagem da composição de serviços em unidades funcionais de negócio, ou seja, que atendam a uma determinada função/processo de negócio; definição *das property* que foram estabelecidas durante a montagem;

- conexão dos pontos de acesso das composições de serviços as *references* de outras composições;atribuição dos valores de qualidade de serviço, tais como, segurança, autenticação e transação, entre outros.

3.3 Benefícios da SCA para o desenvolvimento em SOA

Os principais benefícios da SCA são:

1 – Fraco acoplamento: Componentes podem ser integrados com outros componentes sem

necessitar saber como eles foram implementados. Fraco acoplamento é um requisito para uma aplicação em SOA.

2 – Flexibilidade: Componentes podem ser facilmente repostos por outros componentes. É um requisito para uma sistema em SOA.

3 – Produtividade: Fácil composição de componentes simples e compostos para formar aplicações orientadas a serviços.

4 – Serviços: Podem ser invocados facilmente de forma síncrona ou assíncrona.

5 – Composição de Serviços: São descritas de forma clara e precisa através de arquivos XML

Os benefícios listados acima, se devem fundamentalmente à configuração dos componentes e composições de serviços em arquivos XML [14]. Isso possibilita a abstração dos detalhes de desenvolvimento dos componentes e serviços da invocação e utilização deles. Além disso, o uso de arquivos XML proporciona portabilidade à SCA, uma vez que esse tipo de arquivo é interpretado em qualquer linguagem de programação e sistema operacional.

4. Ambiente Experimental e Estudo de Caso

As ferramentas utilizadas no ambiente experimental foram eclipse [15], netbeans [16] e a Tuscany [17]. O eclipse foi utilizado para desenvolver o código da aplicação e os modelos UML foram criados no netbeans. A Tuscany [17] foi utilizada para o desenvolvimento dos artefatos da SCA e como o ambiente de execução da aplicação. Uma vez que esta ferramenta é relativamente pouco

conhecida, a próxima sub-seção é dedicada a ela.

Fabric3 [18] é outra implementação *open-source* da SCA porém ainda está em desenvolvimento inicial. Todas as outras implementações da SCA são de empresas que fazem parte do consórcio de especificação da SCA e são pagas.

4.1 Tuscany

Tuscany é um projeto *open-source* desenvolvido pela Apache Foundation que implementa a especificação SCA 1.0 e fornece um ambiente leve de execução que pode ser facilmente acoplado a uma ferramenta de desenvolvimento. Ele tem implementações da SCA em Java e C++. O estudo de caso desse trabalho utilizou a implementação em Java.

A figura abaixo ilustra o ambiente de execução da SCA implementado pela Tuscany composto pelos seguintes itens[17]:

1. SCA spec API: Consiste na implementação da especificação cliente da SCA em Java.
2. Tuscany API: Composta por extenções da SCA desenvolvidas pela Tuscany.
3. Core: Possui a implementação do ambiente de execução SCA.
4. Extenções:
    1. *Component Implementation:*Consiste num conjunto de implementações da SCA em outras linguagens como BPEL, Python, C++, Ruby.
    2. *Binding*: Suporte a outros protocolos de *binding* como Axis2, CXF, EJB.
    3. Interface *Binding*: Extensão das interfaces que definem os serviços, como Java, WSDL.
    4. *Databinding*: Extensão dos tipos de dados que trafegam entres os domínios, como SDO, JAXB.
    5. Plataformas: O ambiente que hospeda o ambiente de execução Tuscany.

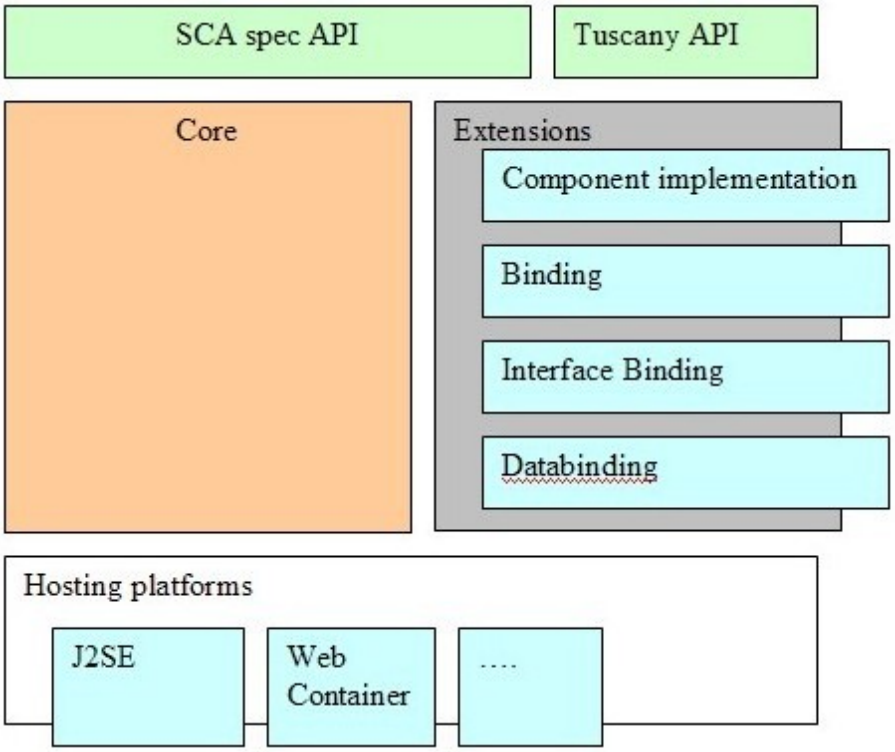

Figura 4 – Ambiente de execução Tuscany[17].

Apesar da Tuscany implementar a versão 1.0 da especficação SCA, há uma certa dificuldade do desenvolvimento de uma aplicação orientada a serviços utilizando essa versão. O principal ponto que leva a essas dificuldades é o desenvolvimento manual dos artefatos da SCA, ou seja, os arquivos XML que definem os componentes e as composições de serviços são escritos manualmente pelo desenvolvedor.

Uma possível solução para isso é o desenvolvimento de uma ferramenta gráfica de desenvolvimento dos artefatos da SCA. Dessa forma, os artefatos podem ser gerados automaticamente pela ferramenta, facilitando o trabalho e evitando que o desenvolvedor cometa erros durante na escrita dos artefatos. O projeto Eclipse SOA Tools Plataform [15] almeja exatamente isso, uma ferramenta para o desenvolvimento de aplicações orientada a serviços utilizando a SCA que possibilite projeto, configuração, *deployment,* monitoração e gerencimento desse tipo de software através de interfaces gráficas e wizards.

Infelizmente, o Eclipse SOA Tools Plataform está em estágio inicial de desenvolvimento e não suporta a SCA ainda.

Um dos pontos fortes da Tuscany são as suas extensões. O quadro abaixo ilustra essas estensões.

Implementações Suportadas	Bindings	Data Bindings	Interfaces Suportadas
Componentes Java Spring Assemblies[11] BPEL Xquery[12] OSGI[13]	Web Services usando Apache Axis2[14] JMS JSON-RPC EJB Feed	JAXB SDO Axiom Castor XMLBeans	Java WSDL 1.1

Tabela 1 – Extensões presentes na Tuscany

Essas extensões facititam a adoção da Tuscany como ambiente de execução de uma aplicação SCA, uma vez que provêm uma gama de tecnologias para uso. Além disso, elas potencializam o uso da SCA pela facilidade de troca e manutenção das tecnologias utilizadas na aplicação.

4.2 Estudo de caso

Neste trabalho foi desenvolvido um estudo de caso utilizando algumas tecnologias atuais juntamente com a SCA, com o intuito de avaliar os benefícios e dificuldades na utilização dessa tecnologia emergente.

O estudo de caso se refere a um sistema para a obtenção de informações financeiras de empresas que possuem ações negociadas na BOVESPA [18]. Os usuários desse software são investidores que pretendem aplicar dinheiro em renda variável. O sistema lista todas as empresas que possuem ações na BOVESPA e o investidor escolhe a empresa da qual deseja visualizar os indicadores financeiros para análise.

O intuito desse estudo de caso é verificar os benefícios para os desenvolvedores na implementação de aplicações orientadas a serviços utilizando a SCA. Durante o seu projeto, buscou-se utilizar o maior número possível de artefatos da SCA, como *references*, *property*, *service*, *component*, *domain*, entre outras.

Para descrever esse estudo de caso foram utilizados os seguintes artefatos da UML: diagrama de caso de uso, caso de uso e diagrama de seqüência.

4.2.1 Projeto UML

O diagrama de caso de uso da figura 5 mostra a relação entre o ator *Investidor* e o caso de uso *Consultar Empresa.* O diagrama possibilita a compreensão de forma rápida do ator e de como ele interage com o sistema. Nesse estudo de caso o diagrama é trivial.

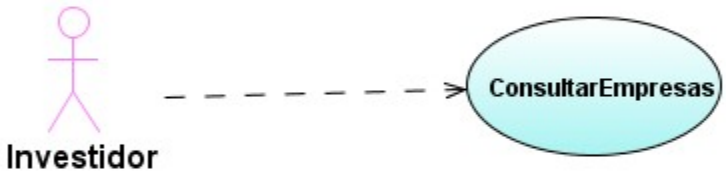

Figura 5 – Diagrama de Caso de Uso

A listagem abaixo contém a descrição do caso de uso consultar empresas. Nela sãoidentificadas todas as ações na interação entre o investidor e o sistema.

Consultar Empresas

Caso de Uso	Consultar Empresas
Atores	Investidor
Tipo	Primário
Descrição	O investidor deseja consultar as empresas que possuem ações negociadas na BOVESPA. O usuário informa a empresa da qual ele deseja verificar as informações financeiras.

Consultar Empresas – Seqüência Típica de Eventos

Ação do Ator	Resposta do Sistema
O usuário informa o seu nome.	
	O sistema retorna a lista de empresas que possuem ações na BOVESPA.
O usuário informa o código da empresa da qual deseja ver os indicadores financeiros.	

	O sistema retorna o lucro, o *payout* e o dividendo da empresa.

O diagrama de sequência abaixo da figura 6 mostra a interação entre os objetos, serviços e o ator do sistema usado como estudo de caso. No passo 1, o usuário informa o seu nome. Após isso, o sistema busca a lista de empresas invocando o método getEmpresas do serviço EmpresaService. Como uma empresa possui um setor associado a ela, é necessário chamar o serviço SetorService que retorna o setor de uma determinada empresa. As empresas são apresentadas ao investidor que informa o nome da empresa que deseja visualizar os indicadores de mercado, passo 2. É invocado o método getIndicadoresEmpresa da BovespaService que faz o cálculo dos indicadores no passo 2.2 e depois retorna os indicadores.

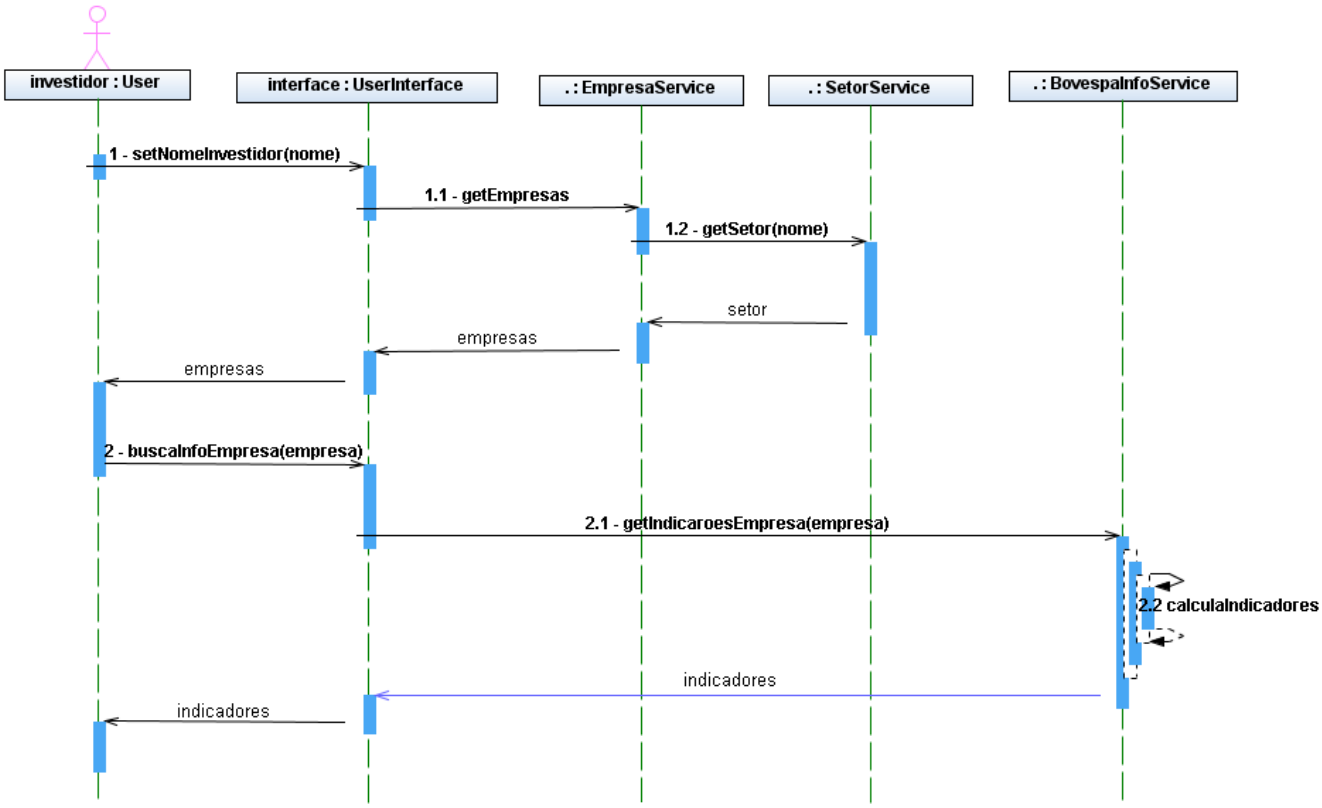

Figura 6 – Diagrama de Seqüência

4.2.2 Componentes e Serviços

Foram criados dois componentes: EmpresaServiceComponent e BovespaInfoServiceComponent. A configuração desses componentes num arquivo de definição de composição de serviços da SCA aparece abaixo. O EmpresaServiceComponent é acessado através de um Web Service definido em <binding.**ws** uri="http://localhost:8085/BovespaInfoService" />. A sua interface é definida utilizando a WSDL, definido em <interface.wsdl interface="http://bovespainfo#wsdl.interface(BovespaInfo)" />. A implementação do componente é feita em Java e o nome da classe é EmpresaServiceImpl, definido em <implementation.java

class="helloworld.EmpresaServiceImpl" /> . Esse componente possui uma referência, ou seja, consome um outro serviço, chamado setorService. Essa *reference* é definida em:<reference name="setorService" target="SetorService" />. O BovespaInfoServiceComponent é acessado através da tecnologia RMI, desenvolvida em Java. A implementação do componente é feita pela classe BovespaInfoServiceImpl.

```
<component name="EmpresaServiceComponent">
        <service name="EmpresaService">
        <interface.wsdl
                        interface="http://bovespainfo#wsdl.interface(BovespaInfo)"
/>
                <binding.ws uri="http://localhost:8085/BovespaInfoService" />
        </service>
        <implementation.java class="EmpresaServiceImpl" />
        <reference name="setorService" target="SetorService" />
</component>

<component name="BovespaInfoServiceComponent">
        <service name="BovespaInfoService">
        <tuscany:binding.rmi host="localhost" port="8099"  serviceName="BovespaInfoRMIService" />
        </service>
        <implementation.java class="BovespaInfoServiceImpl" />
        <reference name="indicadorFinanceiroService"
                target="IndicadorFinanceiroServiceComponent">
        </reference>
</component>
```

4.2.3 Execução passo a passo do estudo de caso

Para facilitar a compreensão do funcionamento do estudo de caso juntamente com a SCA, o sistema em execução e a interação entre os componentes e serviços são descritos passo a passo nessa seção.

O diagrama de seqüência da seção 5.1 pode ser utilizado para acompanhar a execução do estudo de caso . Primeiramente, o investidor informa o seu nome, passo do 1 do diagrama. A tela

abaixo ilustra o sistema em execução nesse passo:

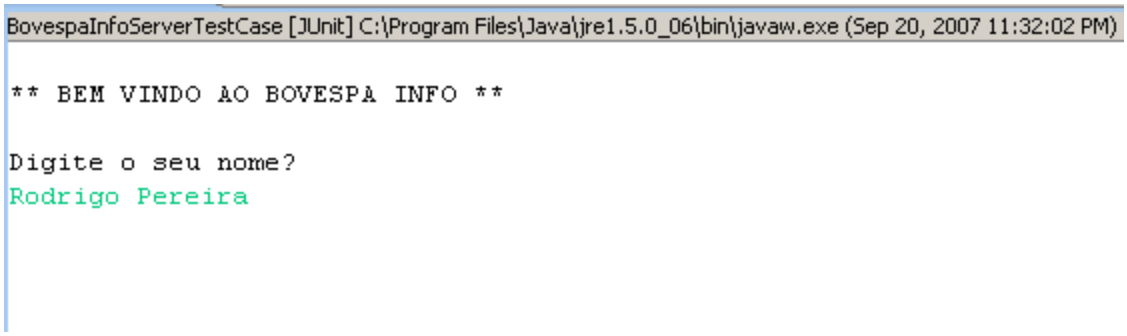

Figura 7 – Tela inicial

No passo 1.1 do diagrama de seqüência, é invocado o EmpresaService, para obter a lista de empresas. Para fazer isso é utilizado o seguinte código:

```
1   SCADomain scaDomain = SCADomain.newInstance("bovespainfo.composite");
2   EmpresaService empresaService = scaDomain.getService(
3                                   EmpresaService.class,
4                                   "EmpresaServiceComponent/EmpresaService");
5
6   System.out.println(empresaService.getEmpresas());
```

Na linha 1, é instanciado um SCA Domain. Após isso, na linha 2, é obtida uma instância do componente EmpresaServiceComponent que pertence a esse domínio. Esse componente consome um Web Service. Conforme declarado na composição de serviço:

```
<component name="EmpresaServiceComponent">
   <service name="EmpresaService">
      <interface.wsdl
```

```
interface="http://bovespainfo#wsdl.interface(BovespaInfo)" />
        <binding.ws uri="http://localhost:8085/BovespaInfoService"/>
    </service>
        <implementation.java class="helloworld.EmpresaServiceImpl" />
        <reference name="setorService" target="SetorService"/>
</component>
```

Esse Web Service provê um lista de empresas que possuem ações negociadas na BOVESPA. Para obter essa lista, basta executar o método getEmpresas() do componente, conforme linha 6. A implementação da forma de acesso a esse Web Service é feita pelo ambiente de execução da SCA, ou seja, a Tuscany. Conforme descrito na seção 4.1, A Tuscany implementa diversas formas de *binding.* Esse web service, invoca o serviço SetorService para obter o setor da empresa, passo 1.2 do diagrama de seqüência. O resultado é mostrado abaixo:

Figura 8 – Lista de empresas retornadas pelo Web Service

No passo 2 do diagrama de sequencia, o investidor informa o código da empresa da qual ele deseja visualizar os indicadores financeiros. Após isso, o sistema retorna os indicadores. A tela abaixo ilustra esses dois passos:

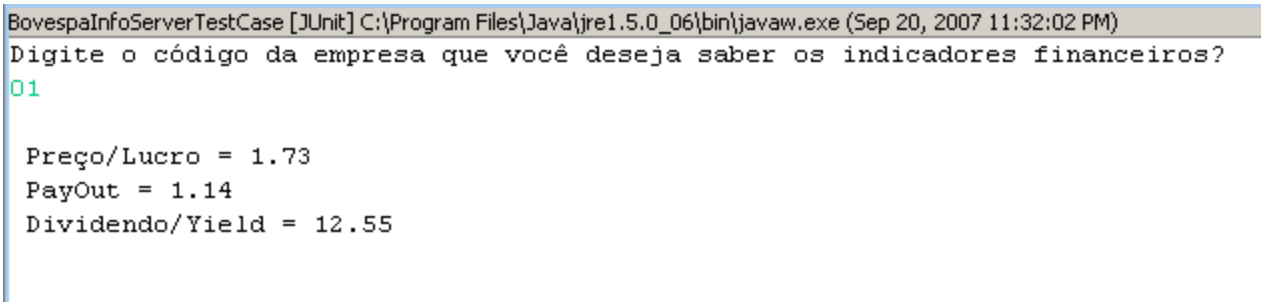

Figura 9 – Exibição dos indicadores financeiros da empresa com código 01

O código que implementa esse passo é o seguinte:

```
1 System.out.println("\n\nDigite o código da empresa que você deseja saber os
2 indicadores financeiros?");
3 BufferedReader buf = new BufferedReader(new InputStreamReader(System.in));
4 String codigoEmpresa = buf.readLine();
5 BovespaInfoService bovespaInfoService = (BovespaInfoService) Naming.
6 lookup("//localhost:8099/BovespaInfoRMIService");
7 System.out.println(bovespaInfoService.buscaInfoEmpresa(codigoEmpresa));
```

Entre as linhas 1 e 4 é obtido o código da empresa. Na linha 5, é localizado o serviço BovespaInfoService, definido na composição de serviços da SCA, confome abaixo:

```
1       <component name="BovespaInfoServiceComponent">
2           <service name="BovespaInfoService">
3               <tuscany:binding.rmi host="localhost" port="8099"
```

```
4       serviceName="BovespaInfoRMIService"/>
5           </service>
6                   <implementation.java class="helloworld.BovespaInfoServiceImpl"/>
7                   <reference name="indicadorFinanceiroService"
8       target="IndicadorFinanceiroServiceComponent"></reference>
9   </component>
```

Nas linha 3 e 4 é definida a forma de *binding* desse componente. No caso, foi utilizada a tecnologia RMI. O ambiente de execução Tuscany, cria uma instância desse serviço, que recebe requisições na porta 8099, quando ele é inicializado. Dessa forma, quando é invocado o método buscaInfoEmpresa, é executado o código implementado na classe BovespaInfoServiceImpl, conforme declarado na linha 6. Essa classe é implementada em Java.

4.2.4 Avaliação da SCA e do Tuscany

A principal vantagem da SCA identificada no estudo de caso foi a facilidade no desenvolvimento de uma aplicação baseada em SOA. Toda a complexidade de implementar o código que consome e provê serviços é encapsulada na implementação da especificação SCA *Assembly Model* pela Tuscany. Para mudar a tecnologia de *binding* de um componente, basta alterar uma linha no arquivo de composição de serviços em XML. Futuramente, as ferramentas darão suporte visual à SCA, e isso será feito alterando uma propriedade do componente visualmente.

Exemplificando essa vantagem, o componente BovespaServiceInfoComponent é acessado através de RMI, conforme definido abaixo:

```
<component name="BovespaInfoServiceComponent">
      <service name="BovespaInfoService">
            <tuscany:binding.rmi host="localhost" port="8099"
                  serviceName="BovespaInfoRMIService" />
      </service>
      <implementation.java class="helloworld.BovespaInfoServiceImpl" />
      <reference name="indicadorFinanceiroService"
            target="IndicadorFinanceiroServiceComponent">
      </reference>
</component>
```

Para mudar a forma de acesso de RMI para EJB, por exemplo, basta mudar a <tucany:binding.rmi para <tucany:binding.ejb.

Dessa forma, o desenvolvedor não necessita conhecer a API de RMI e EJB para construir uma aplicação orientada a serviços. Ele pode focar os seus esforços nas regras de negócio do sistema, e não nos detalhes de infra-estrutura.

Após o desenvolvido o estudo de caso, verifica-se claramente a falta de uma ferramenta visual para o desenvolvimento dos artefatos da SCA e da implementação dos componentes. Essa ferramenta poderia utilizar a Tuscany, que implementa as especificações da SCA. A ferramenta gr'afica poderia ter *wizards* para facilitar a criação das composições de serviços. Atualmente, é necessário conhecer os arquivos XML da SCA para desenvolver uma composição.

A Tuscany e a SCA ainda estão em fase de desenvolvimento inicial, dessa forma o grau de maturidade dessas tecnologias é baixo. Contudo, importantes empresas como IBM estão investindo nessas tecnologias e pretendem utilizá-las em larga escala pois a adoção de SOA é vantajosa do ponto de vista financeiro.

5 Trabalhos Relacionados

A SCA é uma tecnologia nova, conseqüentemente existem muito poucos trabalhos relacionados a ela. Não foram encontrados trabalhos de conclusão de curso e teses de mestrado que falem dela.

Zou [19] discute SOA como uma metodologia de desenvolvimento de software que tem sido utilizada por décadas e que não provê detalhes suficientes do desenvolvimento, reuso e integração de serviços. Para preencher essas lacunas, a autora sugere a combinação da BPEL4WS com a SCA. A BPEL4WS é uma linguagem para construir processos de negócios e integrar serviços com fraco acoplamento, enquanto a SCA aulixia no desenvolvimento de composições de serviços de modelos de negócios desenvolvidos utilizando a BPEL4WS.

Beisige [20] dicute como os detalhes de infra-estrutura e qualidade de serviço podem ser associados aos artefatos da SCA. O artigo faz uma minuciosa análise do *Policy Framework Specification* da SCA, dando detalhes de todos os elementos que ela possui e os artefatos que podem ser relacionados com eles.

6 Conclusões

O desenvolvimento de sistemas que possuam a capacidade de interoperar-se com outros sistemas para obter e compartilhar informações de negócios é um desafio atual para a comunidade de sistemas de informação. A abordagem SOA tem sido utilizada largamente pelas empresas e tende a se difundir ainda mais nos próximos anos. A SCA visa padronizar o desenvolvimento de aplicações orientadas a serviços, uma vez que SOA possui lacunas ainda não preenchidas.

Este trabalho apresentou a especificação *Assembly Model* da SCA, que possui um conjunto de regras para a composição de serviços. Além disso, foi desenvolvido um estudo de caso com o intuito de mostrar as vantagens da adoção da SCA, do ponto de vista do desenvolvedor de software, A principal vantagem é que o desenvolvedor não precisa se ater a detalhes da infra-estrutura da aplicação. Dessa forma, ele pode focar o seu trabalho no desenvolvimento das regras e componentes de negócio.

Podem ser desenvolvidos diversos trabalhos relacionados a esse tema, como por exemplo, a aplicação da SCA utilizando diversas tecnologias de implementação de componentes, como C++, Java e PHP. Dessa forma, pode ser melhor avaliada a facilidade que a SCA provê na invocação de componentes desenvolvidos em tecnologias distintas dentro de uma mesma aplicação.

Outro trabalho futuro importante é o desenvolvimento de uma aplicação que utilize diversas tecnologias de *binding* dos componentes, como por exemplo, Web Services, JMS, JSON-RPC, EJB, entre outros. Isso permitirá avaliar se o desenvolvedor realmente não precisará compreender muitos detalhes dessas tecnologias, pelo fato da SCA prover uma visão de mais alto nível da implementação que as utiliza.

Referências

1] *Recommended Practice for Architectural Description of Software-Intensive Systems*; ANSI/IEEE Std 1471-2000,

[2] Guy Bieber, Jeff Carpenter; *Introduction to Service-Oriented Programming;* 2002; http://www.openwings.org Acessado em Jun/2007

[3] Margolis, Ben. SOA for the business Developer – Concepts, BPEL and SCA. McPress , pp. 193-226 , 2007.

[4] PELTZ, Chris. Web Services *Orchestration and Choreography*. IEEE [Computer, 36:10] , Outubro de 2003

[5] *Business Process Execution Language for Web Services* Version 1.1, http://www-106.ibm.com/developerworks/webservices/library/ws-bpel/ Acessado em Junho / 2007

[6] W3C. Web Services *Choreography Working Group Charter*. Disponível em: http://www.w3.oorg/2005/12/wscwg-charter.html. Acesso em: Maio de 2007.

[7] Larman, Craig. *Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development* . Prentice Hall. pp. 298-305, 2007.

[8] Clemens, Szyperski. *Independly Exetnsible Systems – Software Engineering Potential and Challenges* em Proceedings of 19th Australian Computer Science Conference, Melbourne, Australia, Janeiro 1996

[9] W3C. Web SOAP Version 1.2 Part 0: Primer (Second Edition) . Disponivel em: http://www.w3.org/TR/2007/REC-soap12-part0-20070427/ . Acesso em: Setembro de 2007.

[10] Bernardette Charron-Bost, Friedemann Mattern e Gerard Tel; *Synchronous, asynchronous and causally ordered communication* em Distributed Computing, Vol. 9 No. 4, pp. 173 - 191, 1996

[11] OASIS. *Organization for the Advancement of Structured Information Standards* . Disponivel em: http://www.oasis-open.org/who/. Acesso em: Setembro de 2007.

[12] STAL, Michael. *Using Architectural Patterns and Blueprints for Service-Oriented Architecture*. IEEE, 2006.

[13] IBM. *Service Component Architecture: Build systems using SOA.* Disponível em:

http://www.ibm.com/developerworks/library/specification/ws-sca/. Acesso em: Janeiro de 2007.

[14] Chappel, David. *Introducing SCA*. OSOA, 2007.

[15] Eclipse *Foundation*. Eclipse - *an open development platform*. Disponivel em: http://www.eclipse.org/ Acesso em: Setembro de 2007.

[16] Sun Microsystem. NetBeans IDE. Disponivel em: http://www.netbeans.org/. Acesso em: Setembro de 2007.

[17] Apache. *Apache Tuscany SCA Java Architecture Guide.* Disponivel em: http://incubator.apache.org/tuscany/sca-java-architecture-guide.html. Acesso em: Agosto de 2007.

[18] Bolva de Valores de São Paulo. Disponivel em: http://www.bovespa.com.br/Principal.asp . Acesso em: Julho de 2007.

[19] Zhile Zou, Zhenhua Duan. *Building Business Processes or Assembling Service Components:*
Reuse Services with BPEL4WS and SCA. IEEE, 2006.

[20] Beisiegel, Michael; Kavantzas, Nickolas; Malhotra, Ashok; Pavlik, Greg; Sharp, Chris. SCA *Policy Association Framework*. IBM, 2007.

Anexos

Empresa.java

```
package br.inf.ufsc.ine5632.bovespainfo.entidades;

public class Empresa {

	private String nome;

	private String codigo;

	private Setor setorAtuacao;

	public Empresa(String nomeEmpresa, int codigo) {
		this.nome = nomeEmpresa;
		this.codigo = "0"+codigo;
	}

	public String getCodigo() {
		return codigo;
	}

	public void setCodigo(String descricao) {
		this.codigo = descricao;
	}

	public String getNome() {
		return nome;
	}

	public void setNome(String nome) {
		this.nome = nome;
	}

	public Setor getSetorAtuacao() {
		return setorAtuacao;
	}

	public void setSetorAtuacao(Setor setorAtuacao) {
		this.setorAtuacao = setorAtuacao;
	}

	@Override
	public String toString() {
		return "Código da Empresa: "+this.codigo+"\nNome da Empresa:
"+this.nome+"\nSetor de Atuação: "+this.getSetorAtuacao().getNomeSetor()+"\n";
	}
}
```

Setor.java

```java
package br.inf.ufsc.ine5632.bovespainfo.entidades;

public class Setor {

	private String nomeSetor;

	public Setor(String nomeSetor) {
		this.nomeSetor = nomeSetor;
	}

	public String getNomeSetor() {
		return nomeSetor;
	}

	public void setNomeSetor(String nomeSetor) {
		this.nomeSetor = nomeSetor;
	}
}
```

IndicadorMercado.java

```java
package br.inf.ufsc.ine5632.bovespainfo.entidades;

public class IndicadorMercado {

	private double precoLucro;
	private double payOut;
	private double dividendoYield;

	public IndicadorMercado(double precoLucro, double payOut, double
dividendoYield) {
		super();
		this.precoLucro = precoLucro;
		this.payOut = payOut;
		this.dividendoYield = dividendoYield;
	}

	public double getDividendoYield() {
		return dividendoYield;
	}
	public void setDividendoYield(double dividendoYield) {
		this.dividendoYield = dividendoYield;
	}
	public double getPayOut() {
		return payOut;
	}
	public void setPayOut(double payOut) {
		this.payOut = payOut;
	}
	public double getPrecoLucro() {
		return precoLucro;
	}
```

```
	public void setPrecoLucro(double precoLucro) {
		this.precoLucro = precoLucro;
	}

}
```

BovespaInfoServer.java

```
package br.inf.ufsc.ine5632.bovespainfo.server;


import java.io.IOException;


import org.apache.tuscany.sca.host.embedded.SCADomain;


/*
 * Classe responsável por inicializar um SCA Domain
 */
public class BovespaInfoServer {


    public static void main(String[] args) {


        SCADomain scaDomain = SCADomain.newInstance("bovespainfo.composite");


        try {
            System.out.println("BovespaInfo server started (press enter to shutdown)");
```

```
            System.in.read();
        } catch (IOException e) {
            e.printStackTrace();
        }

        scaDomain.close();
        System.out.println("BovespaInfo server stopped");
    }

}
```

BovespaInfoServiceImpl.java

```
package br.inf.ufsc.ine5632.bovespainfo.services;

import java.text.DecimalFormat;
import java.util.HashMap;
import java.util.Map;

import org.osoa.sca.annotations.Reference;

import br.inf.ufsc.ine5632.bovespainfo.entidades.IndicadorMercado;
import br.inf.ufsc.ine5632.bovespainfo.services.interfaces.BovespaInfoService;
```

```
public class BovespaInfoServiceImpl implements BovespaInfoService {

	private String[] nomeEmpresas = { "Perdigão", "Sadia", "GOL", "TAM",
			"Bradesco", "Itaú", "Unibanco", "Petrobrás", "Gerdau",
			"Companhia Siderúrgica Nacional" };

	private Map<Integer, IndicadorMercado> indicadores = new HashMap<Integer,
IndicadorMercado>();

	public BovespaInfoServiceImpl() {
		init();
	}

	public String buscaInfoEmpresa(String codigoEmpresa) {
		return this.calculaIndicadoresEmpresa(codigoEmpresa);
	}

	private String calculaIndicadoresEmpresa(String codigoEmpresa) {
		String retorno = "";
		IndicadorMercado indicadorMercado = indicadores.get(new Integer(
				codigoEmpresa));
		DecimalFormat formatador = new DecimalFormat("0.00");
```

```
        retorno = " Preço/Lucro = "
                        + formatador.format(indicadorMercado.getPrecoLucro()) + "\n"
                        + " PayOut = "
                        + formatador.format(indicadorMercado.getPayOut()) + "\n"
                        + " Dividendo/Yield = "
                        + formatador.format(indicadorMercado.getDividendoYield());
        return retorno;
}

private void init() {
        double precoLucro = 0.0;
        double payOut = 0.0;
        double dividendoYield = 0.0;

        for (int i = 0; i < nomeEmpresas.length; i++) {
                precoLucro = (double) 10 * Math.random();
                payOut = (double) 8 * Math.random();
                dividendoYield = (double) 15 * Math.random();
                indicadores.put(new Integer(i), new IndicadorMercado(precoLucro,
                                payOut, dividendoYield));
        }
```

```
	}


}
```

EmpresaServiceImpl.java

```
package br.inf.ufsc.ine5632.bovespainfo.services;

import java.util.ArrayList;
import java.util.List;

import org.osoa.sca.annotations.Reference;

import br.inf.ufsc.ine5632.bovespainfo.entidades.Empresa;
import br.inf.ufsc.ine5632.bovespainfo.entidades.Setor;
import br.inf.ufsc.ine5632.bovespainfo.services.interfaces.EmpresaService;
import br.inf.ufsc.ine5632.bovespainfo.services.interfaces.SetorService;

/**
 *
 */
public class EmpresaServiceImpl implements EmpresaService {

	public List<Empresa> listaEmpresas = new ArrayList<Empresa>();

	public String[] nomeEmpresas = { "Perdigão", "Sadia", "GOL", "TAM",
"Bradesco",
			"Itaú", "Unibanco", "Petrobrás", "Gerdau",
			"Companhia Siderúrgica Nacional" };

	SetorService setorService;

	@Reference
    public void setSetorService(SetorService iSetorService) {
        this.setorService = iSetorService;
    }

	void init() {
		Empresa empresa = null;
		Setor setor = null;
		for (int i = 0; i < nomeEmpresas.length; i++) {
			empresa = new Empresa(nomeEmpresas[i], i);
			setor = new Setor(setorService.getSetor(obtemNomeSetor(i)));
			empresa.setSetorAtuacao(setor);
			listaEmpresas.add(empresa);
		}
	}

	private String obtemNomeSetor(int indice) {
```

```
            String auxSetor = "";

            if (indice < 2) {
                  auxSetor = "Alimentos";
            } else if (indice < 4) {
                  auxSetor = "Aviação";
            } else if (indice < 7) {
                  auxSetor = "Bancos";
            } else if (indice < 8) {
                  auxSetor = "Petróleo";
            } else if (indice < 10) {
                  auxSetor = "Siderurgia";
            }

            return auxSetor;
      }

      public String getEmpresas() {
            init();
            String retorno = "*** Lista de Empresas ***\n\n";

            for (Empresa e: listaEmpresas){
                  retorno += e.toString() + "\n";
            }
            return retorno;
      }

}
```

SetorServiceImpl.java

```
package br.inf.ufsc.ine5632.bovespainfo.services;



import java.util.ArrayList;

import java.util.Collection;

import java.util.List;



import br.inf.ufsc.ine5632.bovespainfo.entidades.Setor;

import br.inf.ufsc.ine5632.bovespainfo.services.interfaces.SetorService;
```

```
public class SetorServiceImpl implements SetorService {

	String[] nomeSetores = { "Alimentos", "Aviação", "Bancos", "Petróleo",
			"Siderurgia" };

	List<Setor> setores = new ArrayList<Setor>();



	public SetorServiceImpl() {
		init();
	}



	void init() {
		Setor setor = null;
		for (int i = 0; i < nomeSetores.length; i++) {
			setor = new Setor(nomeSetores[i]);
			setores.add(setor);
		}
	}

	public String getSetor(String nome) {
```

```
                for (Setor setor : setores) {
                        if (setor.getNomeSetor().equalsIgnoreCase(nome)) {
                                return setor.getNomeSetor();
                        }
                }
                return null;
        }

        public Collection<Setor> getSetores() {
                return setores;
        }

}
```

BovespaInfoService.java

```
package br.inf.ufsc.ine5632.bovespainfo.services.interfaces;

import org.osoa.sca.annotations.Remotable;

@Remotable
public interface BovespaInfoService {

    public String buscaInfoEmpresa(String codigoEmpresa);
}
```

EmpresaService.java

```
package br.inf.ufsc.ine5632.bovespainfo.services.interfaces;

import org.osoa.sca.annotations.Remotable;
```

```java
@Remotable
public interface EmpresaService {

    public String getEmpresas();
}
```

SetorService.java

```java
package br.inf.ufsc.ine5632.bovespainfo.services.interfaces;


import java.util.Collection;


import org.osoa.sca.annotations.Remotable;


import br.inf.ufsc.ine5632.bovespainfo.entidades.Setor;
@Remotable
public interface SetorService {
                public Collection<Setor> getSetores();
                public String getSetor(String descricao);
```

}

BovespaInforServerTestCase.java

```
package br.inf.ufsc.ine5632.bovespainfo.test;


import java.io.BufferedReader;

import java.io.IOException;

import java.io.InputStreamReader;

import java.net.Socket;

import java.rmi.Naming;


import junit.framework.TestCase;


import org.apache.tuscany.sca.host.embedded.SCADomain;

import org.junit.Test;


import br.inf.ufsc.ine5632.bovespainfo.services.interfaces.BovespaInfoService;

import br.inf.ufsc.ine5632.bovespainfo.services.interfaces.EmpresaService;


public class BovespaInfoServerTestCase extends TestCase {


        private SCADomain scaDomain;
```

```
@Override
protected void setUp() throws Exception {
	try {
		scaDomain = SCADomain.newInstance("bovespainfo.composite");
	} catch (Exception e) {
		e.printStackTrace();
	}
}

@Test
public void testPing() throws IOException {
	new Socket("127.0.0.1", 8085);
}

@Test
public void testServiceCall() {
	try {
		BufferedReader bufas = new BufferedReader(new InputStreamReader(
				System.in));
		bufas.readLine();
		System.out.println("** BEM VINDO AO BOVESPA INFO **\n");
		System.out.println("Digite o seu nome?");
```

```
                BufferedReader bufa = new BufferedReader(new InputStreamReader(
                                System.in));
                String nome = bufa.readLine();
                EmpresaService empresaService = scaDomain.getService(
                                EmpresaService.class,
                                "EmpresaServiceComponent/EmpresaService");

                System.out.println(empresaService.getEmpresas());

                while (true) {
                        System.out
                                        .println("\n\nDigite o código da empresa que você deseja
saber os indicadores financeiros?");
                        BufferedReader buf = new BufferedReader(new InputStreamReader(
                                        System.in));
                        String codigoEmpresa = buf.readLine();

                        BovespaInfoService bovespaInfoService = (BovespaInfoService) Naming
                                        .lookup("//localhost:8099/BovespaInfoRMIService");
                        System.out.println("\n"
                                        + bovespaInfoService.buscaInfoEmpresa(codigoEmpresa));
                }
```

```
            } catch (Exception e) {
                e.printStackTrace();
            }
        }

        @Override
        protected void tearDown() throws Exception {
            scaDomain.close();
        }

}
```

bovespainfo.composite

```
<?xml version="1.0" encoding="UTF-8"?>
<composite xmlns="http://www.osoa.org/xmlns/sca/1.0"
      targetNamespace="http://bovespainfo"
      xmlns:tuscany="http://tuscany.apache.org/xmlns/sca/1.0"
      xmlns:hw="http://bovespainfo"
    name="bovespainfo">

    <component name="EmpresaServiceComponent">
          <service name="EmpresaService">
              <interface.wsdl
interface="http://bovespainfo#wsdl.interface(BovespaInfo)" />
              <binding.ws uri="http://localhost:8085/BovespaInfoService"/>
          </service>
            <implementation.java
class="br.inf.ufsc.ine5632.bovespainfo.services.EmpresaServiceImpl" />
            <reference name="setorService" target="SetorService"/>
    </component>

      <component name="SetorService">
            <implementation.java
class="br.inf.ufsc.ine5632.bovespainfo.services.SetorServiceImpl"/>
      </component>

      <component name="BovespaInfoServiceComponent">
```

```
          <service name="BovespaInfoService">
              <tuscany:binding.rmi host="localhost" port="8099"
serviceName="BovespaInfoRMIService"/>
          </service>
            <implementation.java
class="br.inf.ufsc.ine5632.bovespainfo.services.BovespaInfoServiceImpl"/>
    </component>

</composite>
```

empresa.wsdl

```
<wsdl:definitions targetNamespace="http://bovespainfo"
xmlns:tns="http://bovespainfo" xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
      name="bovespainfo">

     <wsdl:types>
        <schema elementFormDefault="qualified" targetNamespace="http://bovespainfo"
xmlns="http://www.w3.org/2001/XMLSchema">

            <element name="getEmpresas">
                <complexType>
                </complexType>
            </element>

            <element name="getEmpresasResponse">
                <complexType>
                    <sequence>
                        <element name="getEmpresasReturn" type="xsd:string"/>
                    </sequence>
                </complexType>
            </element>

        </schema>
    </wsdl:types>

    <wsdl:message name="getEmpresasRequest">
        <wsdl:part element="tns:getEmpresas" name="parameters"/>
    </wsdl:message>

    <wsdl:message name="getEmpresasResponse">
        <wsdl:part element="tns:getEmpresasResponse" name="parameters"/>
    </wsdl:message>

    <wsdl:portType name="BovespaInfo">
        <wsdl:operation name="getEmpresas">
            <wsdl:input message="tns:getEmpresasRequest"
name="getEmpresasRequest"/>
            <wsdl:output message="tns:getEmpresasResponse"
name="getEmpresasResponse"/>
        </wsdl:operation>
```

```
    </wsdl:portType>

    <wsdl:binding name="BovespaInfoSoapBinding" type="tns:BovespaInfo">
        <wsdlsoap:binding style="document"
transport="http://schemas.xmlsoap.org/soap/http"/>
        <wsdl:operation name="getEmpresas">
            <wsdlsoap:operation soapAction=""/>
            <wsdl:input name="getEmpresasRequest">
                <wsdlsoap:body use="literal"/>
            </wsdl:input>
            <wsdl:output name="getEmpresasResponse">
                <wsdlsoap:body use="literal"/>
            </wsdl:output>
        </wsdl:operation>
    </wsdl:binding>

    <wsdl:service name="EmpresaService">
        <wsdl:port binding="tns:BovespaInfoSoapBinding" name="EmpresaSoapPort">
            <wsdlsoap:address
location="http://localhost:8085/EmpresaServiceComponent"/>
        </wsdl:port>
    </wsdl:service>

</wsdl:definitions>
```