UNIVERSIDADE FEDERAL DO ESTADO DO RIO DE JANEIRO
CENTRO DE CIÊNCIAS EXATAS E TECNOLOGIA

Relatórios Técnicos
do Departamento de Informática Aplicada
da UNIRIO

n° 0015/2010

Estudo dos Principais Conceitos sobre Service Component Architecture (SCA)

Leonardo Guerreiro Azevedo
Flávia Santoro
Fernanda Baião

Departamento de Informática Aplicada

UNIVERSIDADE FEDERAL DO ESTADO DO RIO DE JANEIRO
Av. Pasteur, 458, Urca - CEP 22290-240
RIO DE JANEIRO – BRASIL

Projeto de Pesquisa

Grupo de Pesquisa Participante

Patrocínio

PETROBRAS

Estudo dos Principais Conceitos sobre Service Component Architecture (SCA)*

Leonardo Guerreiro Azevedo, Flávia Santoro, Fernanda Baião

Núcleo de Pesquisa e Prática em Tecnologia (NP2Tec)
Departamento de Informática Aplicada (DIA) – Universidade Federal do Estado do Rio de Janeiro (UNIRIO)

{azevedo, flavia.santoro, fernanda.baiao}@uniriotec.br

Abstract. SCA (Service Component Architecture) corresponds to a model for application development in a Service-Oriented Architecture. Initially, it was developed by vendors, such as BEA, IBM, Oracle, SAP etc., in order to simplify application development. The main goal of this report is present concepts of SCA.

Keywords: Service-Oriented Architecture (SOA), Service Component Architecture (SCA).

Resumo. SCA (Service Component Architecture) provê um modelo para construir aplicações e sistemas baseado em uma Arquitetura Orientada a Serviços. SCA é uma tentativa iniciada por fornecedores de software (tais como BEA, IBM, Oracle, SAP etc.) para simplificar a construção de aplicações em uma Arquitetura Orientada a Serviços. O objetivo deste relatório é apresentar os conceitos de SCA.

Palavras-chave: Arquitetura Orientada a Serviços (SOA), Arquitetura de Componente de Serviços (SCA).

* Trabalho patrocinado pela Petrobras.

Sumário

Figuras

1 Introdução

Service Component Architecture (SCA) é uma tentativa iniciada por fornecedores de software (tais como BEA, IBM, Oracle, SAP etc) de simplificar a construção de uma Arquitetura Orientada a Serviços (SOA – Service Oritented Architecture). As empresas fundaram a organização Open Service Oriented Architecture organization (osoa.org) para hospedar as especificações[1] para SCA [Barber, 2007a]. Atualmente o órgão responsável pela especificação dos padrões é a OASIS (Organization for the Advancement of Structured Information Standards). As versões correntes dos padrões podem ser encontradas em [Barber e Kaiser, 2009a; Mahbod *et al.*, 2007; Marino e Rowley, 2009].

SCA provê um modelo para construir aplicações e sistemas baseado em uma Arquitetura Orientada a Serviços. Ele é baseado na idéia de que funções do negócio são providas como uma série de serviços, os quais são combinados juntos a fim de criar soluções que sirvam a uma necessidade particular do negócio. Estas aplicações compostas podem conter tanto serviços novos criados especificamente para a aplicação, mas também funções do negócio provenientes de sistemas e aplicações existentes, as quais são reutilizadas como parte da composição. SCA provê um modelo para composição de serviços e para criação de componentes de serviços, incluindo o reuso de funções de aplicações existentes dentro da composição SCA [Barber, 2007b].

SCA é um modelo que tem o objetivo de incluir uma grande variedade de tecnologias para componentes de serviços e para os métodos de acesso que são utilizados para conectá-los. Para componentes, isto inclui não apenas diferentes linguagens de programação, mas também *frameworks*, e ambientes geralmente utilizados com estas linguagens. Para métodos de acesso, composições SCA permitem o uso de várias tecnologias de comunicação e acesso a serviços que são usadas em geral, incluindo, por exemplo, web services, sistemas de mensagem e *Remote Procedure Call* (RPC).

O objetivo deste relatório é apresentar os conceitos de SCA (Service Component Architecture). A maioria dos conceitos descritos neste relatório são descritos em [Chappel, 2007].

Este relatório foi produzido pelo Projeto de Pesquisa em SOA como parte das iniciativas dentro do contexto do Projeto de Pesquisa do Termo de Cooperação entre NP2Tec/UNIRIO e a Petrobras/TIC-E&P/GDIEP.

Esse relatório está organizado em 4 capítulos, sendo o capítulo 1 a presente introdução. O capítulo 2 apresenta SCA em detalhes. Nos capítulos 3 e 4, são apresentadas as conclusões do trabalho e as referências bibliográficas, respectivamente.

2 Service Component Architecture

As especificações SCA definem como criar componentes e como combiná-los em aplicações completas. Os componentes em uma aplicação SCA podem ser construídos usando Java ou outras linguagens tais como modelos de programação definidos por

[1] http://www.osoa.org/display/Main/SCA+Resources

SCA, ou eles podem ser construídos utilizando outras tecnologias, como, por exemplo, Business Process Execution Language (BPEL) ou o *framework* Spring[2]. Independente da tecnologia para componentização utilizada, SCA define um mecanismo de montagem comum para especificar como estes componentes são combinados em aplicações [Chappel, 2007].

SCA foca em três aspectos[3] de uma SOA [Barber, 2007a]:

- Composição: como empacotar componentes de software para eles serem reutilizados por outras aplicações.
- Montagem (*Assembly*): define como componentes podem ser colocados juntos e tratá-los como um serviço.

  Uma composição permite considerar um grupo de componentes e tratá-los como um único serviço – isto facilita o reuso.

  A especificação de montagem define também como colocar componentes e composições juntas.

  SCA permite definir detalhes fora do código, sem mudar o serviço, tais como: criptografia de dados, autenticação e fila de mensagens.
- Política: como tratar restrição de acesso ou assinatura digital em uma SOA.

  Em SCA, políticas são descritas utilizando WS-Policy e WS-PolicyFramework. Políticas podem ser definidas fora do serviço, o que garante alterá-las sem ter que recompilar o serviço.

Existem muitas APIs, *frameworks* de persistência, protocolos de rede e linguagens de programação atualmente. Esta complexidade torna difícil construir uma Arquitetura Orientada a Serviços flexível que esteja de acordo com as necessidades da organização. SCA provê uma API única que encapsula os detalhes inerentes, tornando o desenvolvimento mais simples.

Chappell [2007] apresenta que SCA disponibiliza para desenvolvedores uma interface única para serviços escritos em diferentes linguagens e arquiteturas. Mecanismos de ligação (*bindings*) permitem acessar tipos de componentes em particular. Aplicações construídas com SCA podem usar web services, JMS, JCA e EJBs. As ligações também podem ser especificadas fora do código.

2.1 Componentes e composições

SCA propõe uma noção generalizada de componente. SCA especifica também como componentes podem ser combinados em grandes estruturas chamadas de composições (*composites*). A Figura 1 apresenta um exemplo em alto nível de composição incluindo três componentes. A figura ilustra que uma aplicação SCA pode ser acessada por software que não seja SCA, tais como JavaServer Pages (JSP), cliente web service etc. Componentes em uma aplicação SCA podem acessar dados como uma aplicação qualquer. Uma opção para isto é usar Service Data Objects (SDO) [Barber, 2009b] em conjunto com uma tecnologia Java de acesso a dados padrão, tais como JDBC ou Java EE Java Persistence API (JPA).

[2] http://www.ibm.com/developerworks/webservices/library/os-springsca1/index.html

[3] http://www.osoa.org/display/Main/Component+Parts+of+SCA

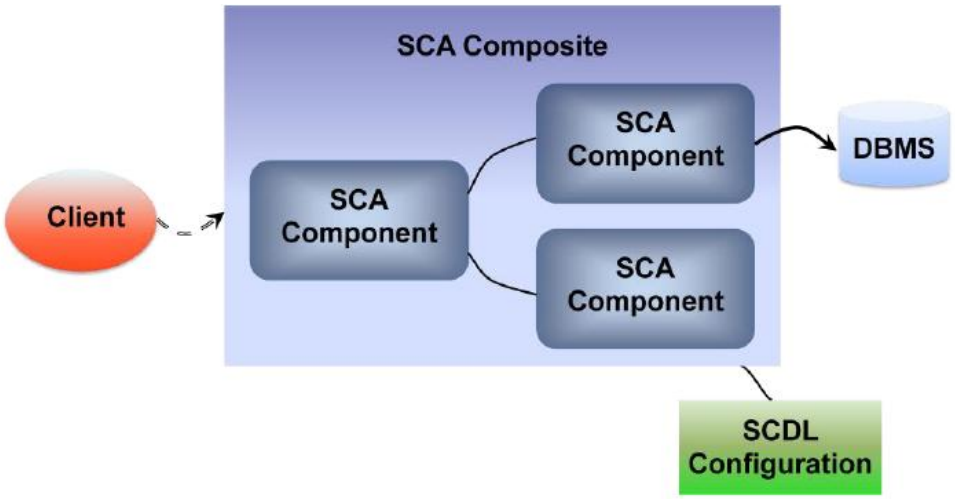

Figura 1 – Exemplo de composição incluindo três componentes

Uma composição SCA é descrita em arquivo de configuração que termina com ".composite". Este arquivo usa um formato baseado em XML chamado de Service Component Definition Language (SCDL) para descrever os componentes que a composição contém e especificar como eles se relacionam. Para o exemplo da Figura 1, teríamos o arquivo de configuração SCDL apresentado na

```
<composite name="ExampleComposite" ...>

     <component name="Component1">
     ...
     </component>

     <component name="Component2">
     ...
     </component>

     <component name="Component3">
     ...
     </component>

</composite>
```

Figura 2 – Exemplo de arquivo SCDL [Chappell, 2007]

2.2 Domínios

Um importante conceito em SCA é o conceito de domínio. Um domínio pode conter uma ou mais composições. Cada composição pode conter componentes implementados em um ou mais processos executando em uma ou mais máquinas. A Figura 3 apresenta um exemplo de domínio SCA.

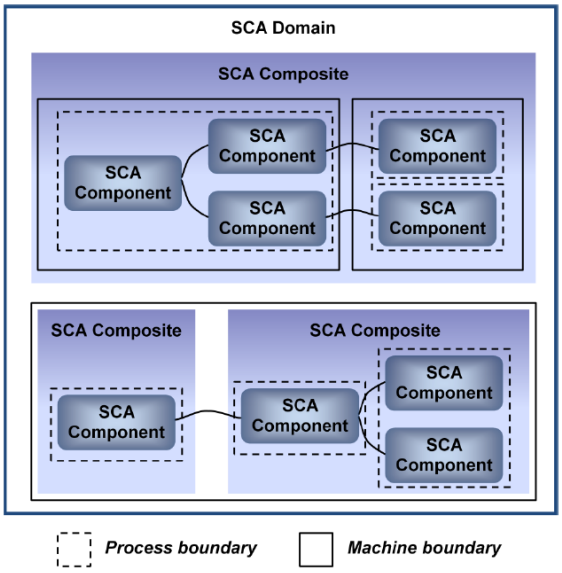

Figura 3 – Exemplo de domínio SCA [Chappell, 2007]

O domínio apresentado na Figura 3 contém três composições (destacadas pela cor de fundo), três computadores (destacados pelos retângulos com lados contínuos) e cinco processos (destacados pelos retângulos com lados tracejados). A composição mais acima na figura consiste de três componentes espalhados em três processos em duas máquinas diferentes. As outras duas composições da parte inferior da figura executam todos os seus componentes em uma única máquina, divididos em três processos separados. Como a comunicação acontece entre estes componentes, se ela é intra-processo, inter-processo, ou inter-computador, pode ser definido diferentemente por cada fornecedor SCA. Independente da escolha, composições não ultrapassam os limites do domínio.

Criar composições que envolvam mais de um domínio – e, portanto, diferentes fornecedores – não é o objetivo da primeira versão de SCA. Por outro, lado é vantajoso para o desenvolvedor trabalhar em um domínio que envolve apenas um fornecedor, pois o desenvolvedor não precisa tratar a complexidade da configuração de aplicações de múltiplos fornecedores.

Mesmo que uma composição SCA execute em um ambiente de um único fornecedor, ela pode ainda se comunicar com aplicações fora do seu próprio domínio. Para fazer isto, um componente SCA pode torna-se ele mesmo acessível usando um protocolo interoperável como, por exemplo, web service. A Figura 4 apresenta um exemplo de comunicação entre domínios, onde um domínio utiliza o ambiente de execução do fornecedor X e o outro domínio utiliza o ambiente de execução do fornecedor Y.

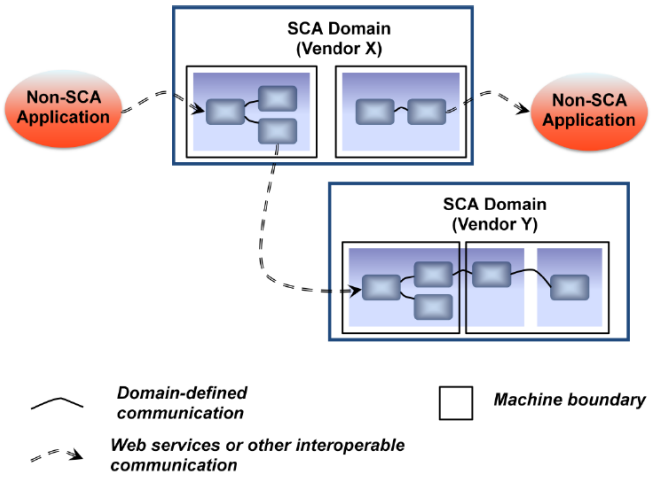

Figura 4 – Exemplo de comunicação entre domínios.

2.3 Serviços, referências, propriedades e ligações

Em SCA, um componente é uma instância de uma implementação que foi apropriadamente configurada. A implementação é o código que provê a funcionalidade do componente, tais como uma classe Java ou um processo BPEL. A configuração expressa em SCDL define como o componente interage com o mundo exterior. Independente da tecnologia utilizada para implementar o componente, ela está relacionada a um conjunto de abstrações em comum, incluindo serviços, referências, propriedades, e ligações, para especificar as interações do serviço.

Cada componente tipicamente implementa alguma lógica do negócio, exposta como um ou mais serviços. Um serviço provê um número de operações que podem ser acessadas pelo cliente do componente. Um componente Java, por exemplo, pode descrever seus serviços usando interfaces Java normais, enquanto um componente implementado com BPEL irá descrever seus serviços usado Web Services Description Language (WSDL).

Um componente também pode invocar serviços providos por outros componentes ou por outros domínios. Um componente pode indicar os serviços que ele invoca utilizando referências. Cada referência define uma interface contendo operações que o componente precisa invocar.

Definir referências de forma explícita oferece várias vantagens. Expressar formalmente as dependências entre componentes ajuda a explicitar o relacionamento entre códigos e permite também realizar injeção de dependência (*dependency injection*). Isto significa que ao invés do desenvolvedor escrever código que localiza o serviço que um componente depende, o *runtime* SCA pode localizar o serviço para o desenvolvedor. Isto implica menos código e a facilidade de mover componentes de um ambiente para outro sem necessidade de alterar código de busca por serviço que os componentes contêm.

Juntamente com serviços e referências, um componente pode também definir uma ou mais propriedades. Cada propriedade contém um valor que pode ser lido pelo componente a partir do arquivo de configuração SCDL quando ele é instanciado. Por exemplo, um componente pode ler uma propriedade para saber em que parte do mundo ele está executando, permitindo customizar seu comportamento apropriadamente.

Ligações (*bindings*) especificam como a comunicação ocorre entre um componente SCA e qualquer outra coisa. Um componente que se comunica com outro componente em um mesmo domínio não precisa ter especificado nenhum *binding*, mesmo executando em processos diferentes ou em máquinas diferentes. Neste caso, o *runtime* determina qual *binding* utilizar. Entretanto, se o componente está se comunicando com uma aplicação SCA ou não SCA de outro domínio, então o criador do componente deve especificar um ou mais *bindings* para realizar esta comunicação.

Como *bindings* separam como um componente se comunica do que ele faz, eles permitem que a lógica de negócio do componente fique separada dos detalhes de comunicação.

Portanto, as abstrações fundamentais de um componente SCA são: serviços, referências, propriedades e (algumas vezes) ligações (*bindings*) (Figura 5). Algumas tecnologias atendem bem a implementação das abstrações de um componente SCA. Por exemplo, o *framework* Spring provê suporte explícito para serviços, referências, e propriedades, e mapeá-los em conceitos semelhantes ao SCA é direto [Beisiegel *et al.*, 2007]. Semelhantemente, BPEL também provê suporte nativo para as abstrações de um componente SCA. O conceito de *partnerLinks*, por exemplo, pode ser mapeado para serviços e referências. Enquanto extensões são necessárias para usar propriedades [Chapman *et al.*, 2007]. No entanto, tanto Spring como BPEL não foram construídos para tratar SCA de forma nativa. Dessa forma, foi desenvolvido o Java *component model* para construir componentes SCA [Barack, 2007]. Foi necessário criar um novo modelo Java porque os modelos anteriores (por exemplo, Enterprise JavaBeans – EJB) foram definidos não tendo serviços como parte fundamental e não foram construídos para endereçar componentes SCA. Além disso, porque *bindings* separam detalhes de comunicação da lógica do negócio. Um modelo de componente Java baseado em SCA pode suportar diferentes estilos de comunicação de uma maneira comum.

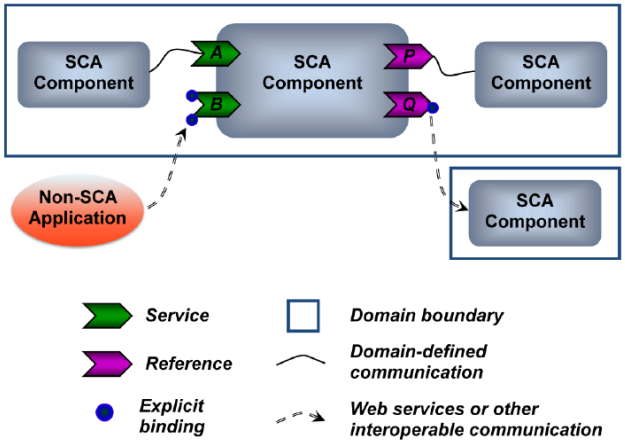

Figura 5 – Abstrações fundamentais de SCA

Diferente das tecnologias J2EE, o modelo de programação Java de SCA utiliza anotações (*annotations*) ao invés de chamadas a APIs. Dessa forma, para um serviço com clientes locais uma interface Java normal e uma classe Java são suficientes. Já um serviço que é acessível por clientes remotos deve indicar este fato pela marcação de uma interface com as anotações apropriadas.

2.4 Exemplo do modelo de componente Java de SCA

O exemplo apresentado na Figura 6 importa a anotação *@Remotable* para indicar que o serviço provido pela interface *AS* pode ser acessado por clientes remotos. Para este componente, isto é suficiente para a classe Java, sendo necessário um pouco mais de código no arquivo SCDL. O *runtime* SCA é responsável por fazer tudo o que for necessário para o serviço ser acessível para clientes remotos. Para clientes no mesmo domínio, cada *runtime* SCA pode usar características próprias. No entanto, a fim de garantir interoperabilidade entre fornecedores, todas as interfaces remotas devem ser expressas em WSDL para que possam ser acessadas via SOAP. Observe que a interface local *MD* não tem nenhuma anotação e não precisa ser expressa em WSDL. As operações *add* e *subtract* da interface *AS* são acessíveis tanto por clientes locais quanto por clientes remotos, por causa da anotação *@Remotable*, enquanto que as operações *multiply* e *divide* são acessíveis apenas por clientes executando no mesmo processo como uma instância da classe *Calculator*.

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

```
@Remotable
public interface AS {
    int add(int a, int b);
    int subtract(int a, int b);
}

public interface MD {
   int multiply(int a, int b);
   int divide(int a, int b);
}
public class Calculator implements AS, MD {
  public int add(int a, int b) {
    return a + b;
  }
  public int subtract(int a, int b) {
    return a - b;
  }
  public int multiply(int a, int b) {
    return a * b;
  }
  public int divide(int a, int b) {
    if (b == 0) {
      throw new IllegalArgumentException();
    }
    else {
      return a / b;
    }
   }
}
```

Figura 6 – Exemplo de componente SCA

No modelo de programação Java de SCA, referências são expressas com a anotação *@Reference*. Por exemplo, se a classe *Calculator* (Figura 6) depende de um serviço de monitoramento para monitorar seu uso, a referência para este serviço seria expressa como na Figura 7. *MonitorService* é uma interface, e então o componente pode invocar métodos desta interface como invoca método de qualquer classe chave, por exemplo, "*monitorService.usageCount(x);*". Todavia, o componente nunca precisa criar uma instância da classe que implementa a interface *MonitorService*. Ao invés disso, o *runtime* localiza automaticamente um componente que provê esta interface, e então ajusta o valor de *monitorService* para apontar para este serviço. Ao invés de confiar no desenvolvedor para escrever o código necessário para acessar o serviço, esta responsabilidade é do *runtime*. Embora esta abordagem seja muitas vezes chamada de injeção de dependência (*dependency injection*), ela é também algumas vezes referenciada como inversão de controle (*inversion of control*). Os detalhes de como um

runtime encontra uma instância do serviço que satisfaz esta referência é específico de domínio; como isto acontece é de responsabilidade de cada *runtime* SCA.

```
@Reference protected MonitorService monitorService;
```

Figura 7 – Especificação de referência no modelo de programação Java de SCA

Uma propriedade é identificada utilizando a anotação *@Property* (). Esta anotação pode pode ser atribuída a um campo de uma classe Java ou a um método *setter* de uma classe, e indica que um valor deverá ser lido do arquivo de configuração SCDL da composição a qual o componente pertence. Propriedades podem ser atribuídas a tipos simples ou complexos. O objetivo é prover uma maneira de configurar um componente via valores que são lidos em tempo de execução.

```
@Property
protected String region;
```

Figura 8 – Definição de propriedade no modelo de programação Java de SCA

No desenvolvimento tradicional, o desenvolvedor tem que tratar diferentes protocolos providos por diferentes tecnologias (por exemplo, SOAP sobre HTTP implica construções em JAX-WS, protocolo de enfileiramento de mensagem implica o uso de Java Message Service – JMS). Já SCA encapsula diferentes protocolos de diferentes tecnologias implementados em diferentes APIS, permitindo que cada serviço remoto e cada referência especifique os protocolos que ele suporta utilizando ligações (*bindings*). O modelo de programação é o mesmo independente do protocolo utilizado. Por exemplo, para ser acessível via SOAP sobre HTTP, um serviço SCA usa o ligação web service[4], enquanto que para acesso utilizando o protocolo de mensageria (*messaging queued protocol*) usa ligação JMS[5]. De forma semelhante a ligação de *EJB session bean*[6] permite acessar *beans* de sessão usando o protocolo Internte Inter-ORB (IIOP).

A versão 1.0 do modelo de componente Java SCA não define maneira para um desenvolvedor especificar ligações diretamente em Java. Ao invés disso, as ligações de um serviço ou referência são escolhidas em tempo de execução, para comunicação intra-domínio, ou ajustadas explicitamente no arquivo de configuração SCDL do componente. Um exemplo de ligação (*binding*) para um serviço de componente é apresentado na Figura 9. Neste exemplo, são especificados o protocolo utilizado para comunicação, através do ".ws" no nome do elemento (indica ligação web service), e onde o serviço pode ser acesso, através do atributo URI do elemento. De forma semelhante, o elemento *binding.jms* especifica ligação JMS e *binding.ejb* indica ligação de *bean* de sessão EJB.

```
<binding.ws uri="http://www.qwickbank.com/services/serviceA"/>
```

Figura 9 - Exemplo de *binding* para um serviço de componente

Outros atributos importantes de SCA são:

- @OneWay: especifica que uma operação não retorna resposta, logo, não bloqueia quem fez a requisição;

[4] http://www.osoa.org/download/attachments/35/SCA_WebServiceBinding_V100.pdf?version=2

[5] http://www.osoa.org/download/attachments/35/SCA_JMSBinding_V100.pdf?version=2

[6] http://www.osoa.org/download/attachments/35/SCA_EJBSessionBeanBinding_V100.pdf?version=1

- @Scope: controla o ciclo de vida do componente: por exemplo, um componente conversacional (*conversational*) mantém estado entre chamadas de métodos, enquanto que um componente sem estado (*stateless*) não.
- @Callback: permite a definição de uma interface de *callback*. Isto suporta comunicação em dois caminhos (*two-way communication*) entre componentes usando o que SCA chama de interface bi-direcional.

Nem todos os atributos são utilizados com todos os *bindings*. Por exemplo, o atributo *@Scope* com a opção conversacional só pode ser utilizado com protocolos que permitem passar informação de sessão, como, por exemplo, SOAP *binding* usando WS- ReliableMessaging. Como em qualquer ambiente de programação, desenvolvedores SCA devem entender a tecnologia para poder utilizá-la corretamente.

2.5 Configuração de um componente

A configuração de um componente é feita no arquivo SCDL. Cada componente é definido usando o elemento *component*, e componentes estão contidos dentro de um elemento *composite*. O que deve ser especificado para um componente depende se está sendo definida comunicação com outros componentes dentro do mesmo domínio SCA ou com software fora do domínio. Um exemplo para o primeiro caso é apresentado na Figura 10.

```
<component name="Component1">
   <implementation.java class="services.examples.Calculator"/>
        <property name="region">
           Europe
        </property>
</component>
```

Figura 10 – Exemplo de configuração de comunicação dentro do mesmo domínio

No caso de comunicação fora do domínio, por exemplo, se o serviço *Calculator* e o *Monitor* são remotos, o arquivo de configuração é um pouco mais complexo, como apresentado na Figura 11. Neste caso, estão explícitos os elementos *service* e *reference* para o serviço remoto e a referência para o componente que ele define. Dado que o componente está comunicando com software fora do seu domínio, o *runtime* não pode escolher um *binding*. Ao invés disso, o componente explicitamente especifica que um *binding* interoperável deve ser utilizado. É tarefa do *runtime* SCA gerar as interfaces WSDL a partir das interfaces Java, preparar o serviço para poder ser invocado via SOAP, e executar tudo o que for necessário para permitir que este componente comunica via *web services*.

```
<component name="Component1">
    <implementation.java class="services.examples.Calculator"/>
    <service name="AS">
        <binding.ws uri="http://www.qwickbank.com/services/serviceA"/>
    </service>
    <reference name="MonitorService">
        <binding.ws uri="http://www.qwickbank.com/services/serviceM"/>
    </reference>
    <property name="region">
        Europe
    </property>
</component>
```

Figura 11 – Exemplo de configuração de comunicação para componentes localizados em domínios distintos

De forma semelhante ao elemento *implementation.java*, componentes BPEL usam o elemento *implementation.bpel*, componentes construídos utilizando Spring especificam *implementation.spring*, enquanto que componentes utilizando o modelo de componente C++ de SCA[7] utilizam *implementation.cpp*. Uma composição inteira pode também agir como um componente dentro de outra composição, neste caso é especificado o elemento *implementation.composite*. A Figura 12 apresenta a configuração de um componente escrito em BPEL que se comunica apenas com componentes do seu domínio.

```
<component name="Component1">
    <implementation.bpel process="ExampleProcess"/>
    <property name="region">
        Europe
    </property>
</component>
```

Figura 12 – Exemplo de configuração de componente BPEL

2.6 Composições

Composições agrupam componentes em combinações úteis que também podem fazer parte de outras composições. O uso de composições tem muitas vantagens. Prover um conjunto de abstrações bem definidas para componentes pode ajudar os desenvolvedores a entenderem melhor como aplicações devem ser projetadas. Manter estas abstrações consistentes através de diferentes tecnologias também torna mais fácil construir aplicações usando diferentes linguagens e diferentes *runtime*. Além disso, é útil uma forma de instalar uma aplicação inteira como uma unidade, e dado que componentes provêem serviços descritos e bem definidos, uma ferramenta gráfica pode permitir agrupar e re-agrupar vários componentes como necessário para tratar um determinado problema.

Para alcançar estes objetivos é necessário definir como componentes se relacionam dentro de uma composição. Relacionamentos são descritos na especificação de modelo de composição (*assembly*) de SCA[8].

Uma referência em um componente é conectada a um serviço em outro componente usando um *wire* (fio) (Figura 13). Um *wire* é uma representação abstrata do relacionamento entre uma referência e algum serviço que está de acordo com os requisitos da referência. O tipo de comunicação que um *wire* provê pode variar – ele depende das especificidades do *runtime* que está sendo usado, quais *bindings* estão especificados (se houver), dentre outras coisas.

Assim como um componente expõe serviços, uma composição pode também expor um ou mais serviços. Estes serviços estão implementados, na realidade, por componentes dentro da composição. Para tornar estes serviços disponíveis, a pessoa responsável pela criação da composição pode promover (*promote*) os serviços.

A Figura 13 apresenta um exemplo de promoção feita pela composição do serviço *A* implementado pelo componente 1. De forma semelhante referências definidas por

[7] http://www.osoa.org/download/attachments/35/SCA_ClientAndImplementationModel_Cpp-V100.pdf?version=2

[8] http://www.osoa.org/download/attachments/35/SCA_AssemblyModel_V100.pdf?version=1

componentes podem ser promovidas para serem visíveis fora da composição. A Figura 13 apresenta a promoção das referências T e U.

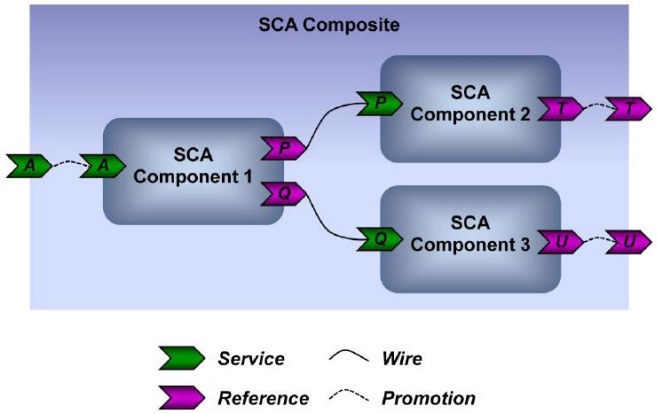

Figura 13 – Exemplo de conexão entre componentes em uma composição e promoção de serviços

Todos os relacionamentos em uma composição são expressos no arquivo de configuração SCDL, como ilustrado na Figura 14. O atributo *autowire* ajustado para verdadeiro indicaque o *runtime* SCA deve automaticamente tentar conectar os serviços e referências definidas pelos componentes na composição. Para realizar isto, o *runtime* procura por correspondências entre as referências e os serviços expostos pelos componentes da composição. Para poder corresponder, um serviço deve prover a interface que a referência requer, permitindo usar um *binding* compatível. Apesar de não demonstrado no exemplo, é possível definir um *wire* explícito entre componentes usando um elemento *wire*.

```
<composite name="ThreeComponents" autowire="true"...>

    <component name="Component1">
        <implementation.bpel process="Process1"/>
    </component>

    <component name="Component2">
        <implementation.java class="services.examples.class2"/>
    </component>

    <component name="Component3">
        <implementation.java class="services.examples.class3"/>
    </component>

    <service name="A" promote="Component1/A">
        <binding.ws/>
    <service/>

    <reference name="T" promote="Component2/T"/>
    <reference name="U" promote="Component3/U"/>

</composite>
```

Figura 14 – Arquivo SCDL de configuração de uma composição

3 Policy framework

O *Policy framework* permite a especificação das interações entre partes de uma aplicação distribuída. Este *framework* define duas categorias de políticas:

- Políticas de interação: modifica como um componente interage com outros componentes. Por exemplo, políticas para definir requisitos de segurança ou transferência de mensagem confiável. Políticas de interação são tipicamente aplicadas a *bindings*.
- Políticas de implementação: modificam como um componente se comporta localmente. Por exemplo, este tipo de política pode especificar que um componente deve ser executado dentro de uma transação (apesar da versão inicial – 1.0 - da especificação SCA não definir políticas de transação).

Políticas são definidas no arquivo de configuração SCDL. Para componentes SCA escritos em JAVA, políticas podem ser definidas usando anotações atribuídas para interfaces, métodos etc. Por exemplo, a anotação *@Confidentiality* indica que a comunicação deve ser confidencial (ou seja, encriptada), enquanto *@Authentication* indica que autenticação é requerida.

Em SCA, em cada domínio, existe o papel administrador de política o qual define o significado de uma política no seu domínio pela especificação de intenções (*intents*) e conjunto de políticas (*policySets*), cada qual contém uma ou mais políticas. Por exemplo, um *binding* para um serviço pode ter um *policySet* associado descrevendo suas políticas de interação, enquanto que um *binding* para uma referência pode ter um outro *policySet* descrevendo suas políticas de interação. Quando um *wire* é criado entre eles, os *policySets* são combinados, e suas interseções determinam o conjunto de políticas usadas para esta comunicação.

SCA não define como políticas devem ser descritas dentro de um domínio. Cada fornecedor pode implementar de uma maneira diferente. Entretanto, é indicado que, para políticas definidas para comunicação entre domínios, seja utilizada uma linguagem independente de fornecedor, como, por exemplo, WS-Policy. Uma observação importante é que, mesmo utilizando o mesmo fornecedor SCA, dois departamentos podem necessitar definir políticas diferentes. Para este caso, Chappel [2007] indica criar dois domínios diferentes, um para cada departamento.

3.1 Exemplo de arquitetura de aplicação em SCA

A Figura 15 ilustra um exemplo de arquitetura de aplicação criada com SCA. Neste exemplo, o cliente é uma página JSP[9] (JavaServer Page). Esta página JSP invoca o serviço *A*, o qual é provido por um componente SCA que é parte de uma composição simples em algum domínio SCA. Este componente foi implementado em BPEL, e seu serviço foi promovido para ser visível fora da composição, o que é expresso no arquivo de configuração SCDL. Este componente BPEL possui referências para outros dois serviços, P e Q. Serviço P é provido por um componente construído de acordo com o modelo de componente Java de SCA, e é um componente que faz parte de outra composição em outro domínio. Dessa forma, a comunicação com este componente é feita utilizando web services (ou outro protocolo interoperável). Serviço Q é implementado como um componente Spring SCA que é parte da mesma composição que o componente BPEL. Logo, a comunicação entre o componente BPEL e o componente Spring considera o *binding* SCA específico de domínio, modificado por qualquer política que tenha sido especificada.

[9] http://java.sun.com/products/jsp/

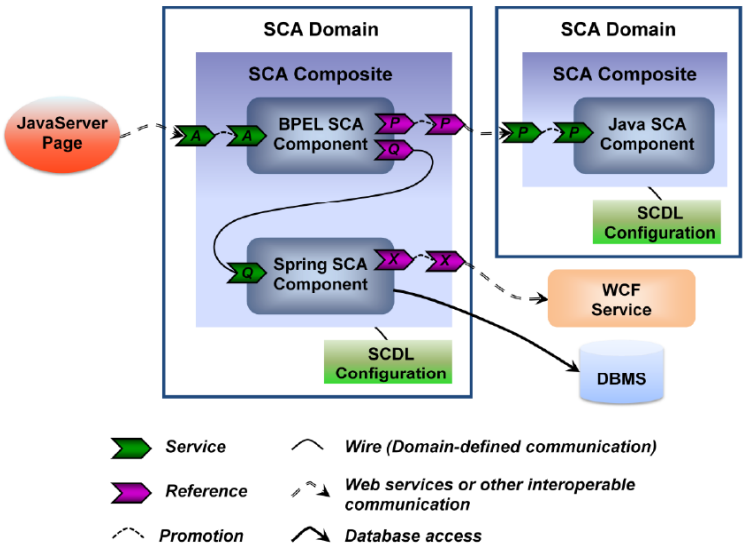

Figura 15 – Exemplo de aplicação criada com SCA

O componente Spring implementa o serviço Q, e invoca o serviço X. Este serviço é provido por uma aplicação criada com Windows Communication Foundation[10] (WCF) da Microsoft. Este serviço não é baseado em SCA; portanto, a comunicação entre o componente Spring e ele deve ser feita utilizando protocolo para web services. Finalmente, o componente Spring acessa um banco de dados. Este acesso pode ser feito utilizando, por exemplo, SDO [Barber, 2009b].

3.2 *Runtime* para SCA

A especificação não diz nada a respeito de como a SCA deve ser implementado. Existem várias implementações de código livre, tais como, Tuscany[11,12,13] [Laws *et al.*, 2009; MAHBOD, 2007] e Fabric3[14], enquanto vários fornecedores estão realizando suas próprias implementações, tais como Oracle[15]. Uma lista de exemplos e implementações[16] de SCA é apresentada pelo Open Service Oriented Architecture group.

O *runtime* SCA pode prover um número de *containers*, um para cada tecnologia de componentes que ele suporta. O elemento *implementation* do arquivo de configuração SCDL informa ao *runtime* qual o tipo de container ele precisa para uma composição particular e onde encontrar a implementação dos componentes na composição.

Embora não seja requerido, um desenvolvedor SCA pode tornar pública a interface entre o *runtime* e os containeres. Dessa forma, terceiros podem criar seus próprios containeres, tornando o *runtime* SCA mais extensível. Uma opção para esta interface é a tecnologia Java Business Integration[17] (JBI), uma especificação criada pela comunidade Java (Java Community Process). JBI suporta apenas tecnologias baseadas

[10] http://www.davidchappell.com/IntroducingWCFv1.2.1.pdf

[11] http://tuscany.apache.org/

[12] http://www.ibm.com/developerworks/webservices/library/ws-sca-tuscany/index.html

[13] http://www.ibm.com/developerworks/webservices/library/ws-soa-scafutureproof/

[14] http://www.fabric3.org/

[15] http://www.oracle.com/technology/architect/entarch/pdf/oracle_sca_the_power_of_the_composite.pdf

[16] http://www.osoa.org/display/Main/Implementation+Examples+and+Tools

[17] http://en.wikipedia.org/wiki/Java_Business_Integration

em Java, entretanto usando a mesma, um container C++ seria algo problemático, e ela não teve grande apoio dos principais fornecedores Java. Tuscany e Fabric3, por exemplo, não implementam JBI.

Outra opção possível é utilizar OSGi[18,19] para implementar o *runtime* SCA. Este especificação foi criada por um grupo formado por fornecedores múltiplos, e tem tido mais apoio dos fornecedores de SCA do que JBI. OSGi define como empacotar código, e apesar de ser baseado em Java, alguns fornecedores de *runtimes* SCA permitem extensões tais como um novo *binding* para ser adicionado no pacote OSGi.

Fornecedores também estão provendo ferramentas para que desenvolvedores tenham facilidades para trabalhar com SCA. Em relação a ferramentas de código aberto, existe o projeto de ferramentas de desenvolvimento baseadas em SCA para Eclipse (SOA Tools Platform Project[20]).

4 Conclusões

A especificação SCA oferece muitas opções, e pessoas diferentes vêem SCA de formas diferentes. Uns podem considerar que SCA corresponde a determinados aspectos da especificação enquanto que outros podem considerar todas as características que a especificação define. Fornecedores também vêem a tecnologia de forma diferente e enfatizam diferentes aspectos de SCA. Por exemplo, um fornecedor pode suportar os aspectos de ligação SCA e seu novo modelo de programação para componentes Java, mas não suportar a versão C++ para este modelo. Dado que as especificações explicitamente permitem extensões, observe cada fornecedor para saber quais customizações em seus produtos SCA eles provêem.

Por SCA prover uma alternativa para abordagens anteriores, tais como EJB e JAX-WS, ela pode oferecer um novo modo de criar lógica de negócio em Java em uma arquitetura orientada a serviços. Muitas questões ainda estão em aberto, mas o potencial da tecnologia é evidente.

5 Referências

BARBER, G. **Service The Business Value Proposition of SCA.** Open Service Oriented Architecture (OSOA), Novembro, Março, 2007a. Disponível em <http://www http://www.osoa.org/display/Main/The+Business+Value+Proposition+of+SCA>. Acessado em 11 Dez. 2009.

BARBER, G. **Service Component Architecture Home.** Open Service Oriented Architecture (OSOA), Novembro, 2007b. Disponível em <http://www.osoa.org/display/Main/Service+Component+Architecture+Home>. Acessado em 11 Dez. 2009.

BARBER, G., KAISER, M. **Service Component Architecture Specifications**. Open Service Oriented Architecture (OSOA), Abril, 2009a. Disponível em <http://www.osoa.org/display/Main/Service+Component+Architecture+Specifications>. Acessado em 11 Dez. 2009.

[18] http://www.osgi.org/About/WhyOSGi

[19] http://www.osgi.org/Specifications/HomePage

[20] http://www.eclipse.org/stp/

BARBER, G., KAISER, M. **Service Data Objects Specifications**. Open Service Oriented Architecture (OSOA), Abril, 2009b. Disponível em <http://www.osoa.org/display/Main/Service+Data+Objects+Specifications>. Acessado em 11 Dez. 2009.

BARACK, R. *et al.* **SCA Service Component Architecture: Java Common Annotations and APIs.** SCA Version 1.00, March, 2007. Disponível em < http://www.osoa.org/download/attachments/35/SCA_JavaAnnotationsAndAPIs_V100.pdf?version=1 >. Acessado em 11 Dez. 2009.

BEISIEGEL, M. *et al*. **SCA Service Component Architecture: Spring Component Implementation Specification**. SCA Version 1.00, March, 2007. Disponível em <http://www.osoa.org/download/attachments/35/SCA_SpringComponentImplementationSpecification-V100.pdf?version=1>. Acessado em 11 Dez. 2009.

CHAPPELL, D. **Introducing SCA**. Chappell & Associates., 2007. Disponível em <http://www.davidchappell.com/articles/introducing_sca.pdf>. Acessado em 11 Dez. 2009.

CHAPMAN, M. *et al.* SCA Service Component Architecture: Client and Implementation Model Specification for WS-BPEL. SCA Version 1.00, March, 2007. Disponível em <http://www.osoa.org/download/attachments/35/SCA_ClientAndImplementationModelforBPEL_V100.pdf?version=1>. Acessado em 11 Dez. 2009.

CHATTERJEE, C. **Service-Component Architectures: A programming model for SOA.** Dr. Dobb's, Agosto, 2007. Disponível em < http://www.ddj.com/architect/201202701>. Acessado em 11 Dez. 2009.

LAWS, S., COMBELLACK, M., FENG, R., MAHBOD H., NASH S. **Tuscany in Action**. MEAP Began, 2009.

MAHBOD, H., FENG, R., LAWS, S. **Building SOA with Tuscany SCA: A simple service-oriented infrastructure**. SOA World Magazine, November, 2007. Disponível em < http://java.sys-con.com/node/458183>. Acessado em 11 Dez. 2009.

MARINO, J., ROWLEY, M. **Understanding SCA (Service Component Architecture).** Addison-Wesley Professional, Junho, 2009.

MAHBOD, H., FENG, R., LAWS, S. **Java Feature — What Is SCA? A simple model for creating service-oriented applications.** SOA World Magazine. Disponível em <http://java.sys-con.com/node/325183>. Acessado em 11 Dez. 2009.

OASIS. **Open CSA**. Organization for the Advancement of Structured Information Standards (OASIS). Disponível em <http://www.oasis-opencsa.org/sca>. Acessado em 11 Dez. 2009.

6 Apêndice A

A OASIS possui um comitê chamado SCA (Composite Services Architecture) que inclui fornecedores e usuários do mundo todo para o desenvolvimento e adoção de Service Component Architecture (SCA) e Service Data Objects (SDO).

São promovidos workshops[21] para disseminar o uso das especificações. SCA e SDO provêem um interface de programação única para um grande variedade de serviços e fontes de dados. Os objetivos do workshop são:

- Discutir o estado corrente de SOA e as dificuldades no seu crescimento;
- Explicar o racional existente em SCA e SDO e como estas tecnologias simplificam o desenvolvimento, teste e implantação de aplicações SOA;
- Demonstrar a tecnologia em ação com exemplos executando no projeto Apache Tuscany, o qual corresponde a uma implementação de código-aberto de SCA e SDO.
- Ilustrar como SCA e SDO trabalham com BPEL (Business Process Execution Language). A combinação de SCA, SDO e BPEL permitem grande flexibilidade em SOA.

As versões correntes das especificações de SCA podem ser encontradas em [Barber e Kaiser, 2009a], enquanto que as especificações correntes de SDO estão disponíveis em [Barber e Kaiser, 2009b].

[21] http://www.oasis-opencsa.org/sca-roadshow/overview