Service Component Architecture (SCA) – Standard for Programming-At-Large in an Enterprise Service Bus (ESB) Environment

Applies to:

Composition, Service Bus-Based Integration, Service-Oriented Architecture, Standards. For more information, visit the Service Bus-based Integration homepage.

Summary

Service Component Architecture (SCA) is a standard for composing business applications by assembling components built in potentially heterogeneous environments. Enterprise Service Bus (ESB) paradigm enables creation of service-oriented homogeneous IT environment on top of heterogeneous applications and platforms. This article describes how SCA technology supports programming-at-large in ESB environments.

Author: Sanjay Patil

Company: SAP Labs – Palo Alto

Created on: 28 December 2009

Author Bio

Sanjay Patil is a Standards Architect at SAP Labs - Palo Alto. His areas of interest include SOA, Business Process Management, Web Services, and Social Media. He currently co-chairs the OASIS Technical Committtees for standardization of Web Services Reliable Messaging (WS-RX), and SCA-BPEL technology.

Table of Contents

Introduction

Service Component Architecture (SCA) is an emerging standard for composing business applications by assembling components that are designed and developed in potentially heterogeneous environments.

Enterprise Service Bus (ESB) is a paradigm for creating a service-oriented homogeneous IT environment on top of heterogeneous applications and platform infrastructure, so that any business applications that leverage functionality from various other services can be easily created and deployed.

This article aims to describe how SCA provides a programming model at large for utilization of the ESB environments.

This article assumes familiarity with the concepts of the SCA standard as well as the ESB paradigm.

Overview

Service Component Architecture (SCA) is a standard developed under the OASIS consortium. SCA defines a standard for design and development of service-oriented components in potentially heterogeneous programming environments that can be easily assembled into business applications. SCA standard comprises of a series of specifications developed by the various SCA Technical Committees addressing the different aspects of this technology.

The goal of this paper is to discuss how Service Component Architecture (SCA) technology provides a programming model for Enterprise Service Bus (ESB).

This paper is divided into three parts:

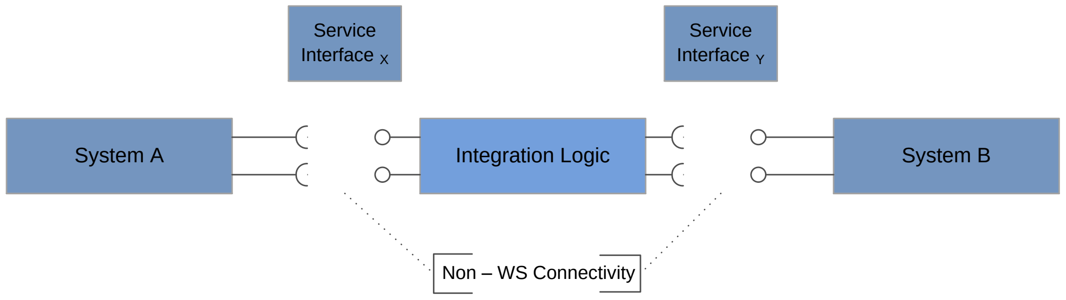
- The first part sets the context for ‘what is an ESB’ by describing some key IT scenarios where an ESB based solution is typically used. An exemplary business problem and an ESB based solution for that problem are used to highlight the concepts.
- The second part builds upon the exemplary ESB based solution and describes certain use cases that highlight the need for a standard ESB programming model.
- The third part provides an overview of the various SCA specifications and describes how SCA provides the standard programming model for ESB.

ESB Scenarios

Let us look at some examples of IT scenarios for which use of ESB would be most typical.

Application To Application (A2A) Integration

This is a classic Enterprise Application Integration (EAI) scenario where different systems are integrated together for creating new solutions. The interface and communication technology may not be standards based.

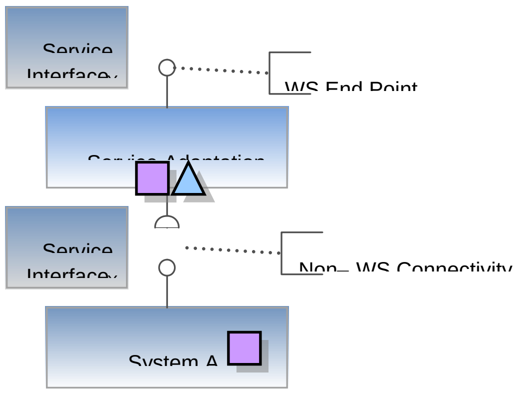


The integration logic may involve content based routing, mapping of messages, invocation of rules, etc.

ESB solutions are commonly considered as an evolution of the EAI products and therefore it is also commonly expected that ESBs provide support for the common EAI scenarios such as A2A Integration.

Service Adaptation

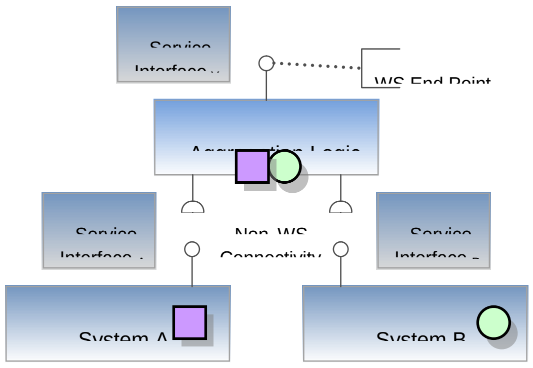
Large scale projects to adopt Service Oriented Architecture (SOA) typically involve defining services based on a top-down analysis of the business requirements. The Service Adaptation layer addresses any mismatch between the newly defining service interfaces and the functionality offered by existing systems.



The Service Adaptation layer may include mapping, enrichment of messages, bridging of communication protocol supported by the existing system with that of the newly defined service. Such functionality is commonly expected to be available in an ESB.

Service Aggregation / Orchestration

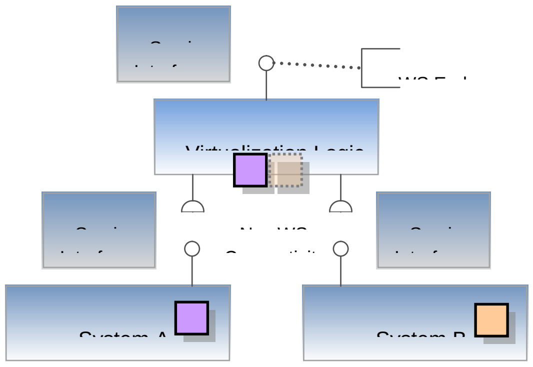
In this scenario, a new service is created by orchestrating other services providing useful functionalities. In a Web services world, the orchestration may be defined using WS-BPEL technology.



Note: There are differing views regarding whether the runtime for service orchestration should be considered as part of an ESB solution or not. However we can set aside this question as related to bundling of different capabilities by the different vendor products. As far as the scenario itself is concerned, it is safe to assume that an ESB should provide support for the design, development and deployment of service orchestrations in one way or another.

Service Virtualization

Often times, it is important to provide a virtual but stable service interface and service endpoint to the clients while the service instance itself may be changing. For example, different calls to the same service endpoint may be routed to different instances for load balancing purposes. Similarly, when a service implementation is undergoing an upgrade from an existing system to a new system, a virtual endpoint may be useful to insulate the service clients from the impact of any changes on the provisioning side.



It is generally expected that an ESB would support such service virtualization scenarios.

Essential Features of an ESB

While the definition of ESB is not standardized as such by a standards body, etc, there seems to be a general consensus regarding a set of features that an ESB must support:

Service Oriented

Unlike its predecessor category of EAI, an ESB takes a service oriented approach for integration. An ESB based solution is not merely plumbing of different systems to bridge the impedance mismatch between the connecting systems, but instead an ESB based development project takes a more service-oriented approach and models even the integration logic as reusable services.

Standards Support

ESBs are expected to support far more standards when compared with their predecessor product category of EAI. At the minimum, an ESB must support XML and Web services standards. In particular, support for reliable and secure Web services is commonly expected. As mentioned earlier, there are differing views for whether an ESB must support a process modeling capability, and related standards such as WS-BPEL.

Protocol Bridging

As highlighted by some of the scenarios, an ESB is expected to support different communication mechanisms and bridging between them.

Multi-Language Support

Regardless of whether the application containers are bundled together with an ESB solution or not, an ESB environment is typically expected to support different programming languages for creating the individual application components.

Integration Services

The traditional integration services such as for mapping, routing, of messages, are expected to be available in an ESB.

Pub/sub Eventing

Support for loosely coupled applications based on publish and subscribe style communications is also expected to be present in an ESB.

Deployment Flexibility

ESBs are also expected to be flexible in regards to their runtime footprint. Some ESBs may use a hub-less architecture that does not mandate all the message exchanges going through a central integration sever, whereby it becomes possible to control the footprint of the ESB based on the underlying systems it spans.

Example IT Scenario

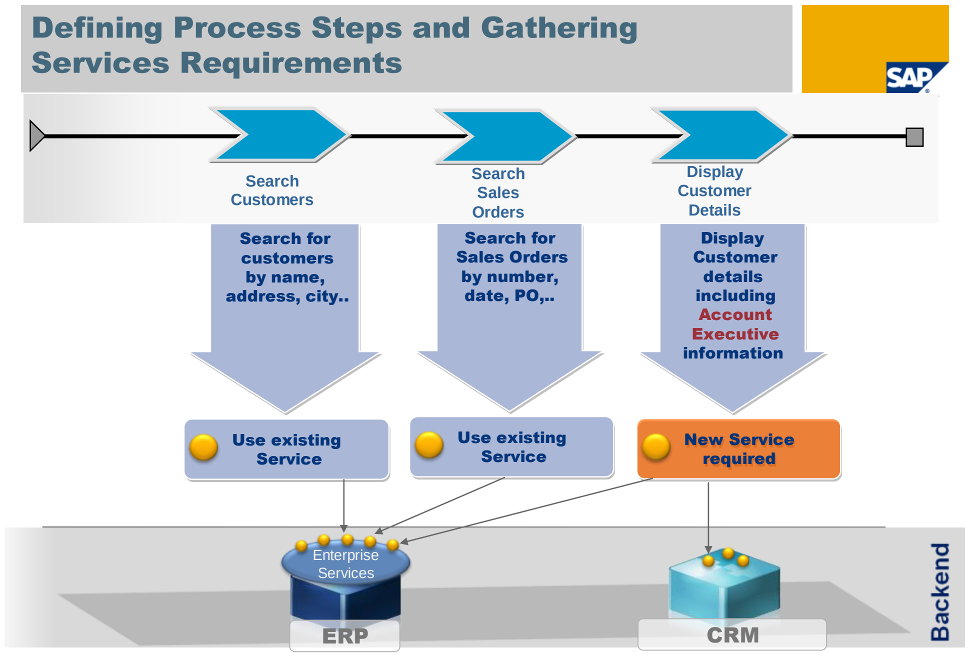
Let us now take an example of creating a business application for ‘Tracking Sales Order’ using which a customer can find out the status of his Sales Order. Such an application would involve different steps such as retrieving and verifying the sales orders data, checking with the supplier for order status, obtaining a tracking number, checking the delivery schedule, etc. Such multi step processes that involve integrating different in-house and partner applications are quite common.

Now consider a business requirement whereby an alert is to be raised whenever an important customer is tracking a sales order with a high dollar amount value. From an IT perspective, this translates into a new feature for the application to send an alert to the account executive after a determination is made that the customer issuing the tracking query is important and the dollar amount of the sales order is beyond a certain threshold value.

From a design standpoint, let us assume that this translates into –

a> A new service that takes as input the customer id, and using the interfaces of the backend CRM system, makes a determination of whether the customer falls into the important category and returns some information about customer, the account executive, etc.

b> A process component that uses an existing service of ERP system to pull sales order data, the new service from the previous step for pulling the customer data and applies some business logic such a checking the dollar amount of the sales order to decides about raising an alert to the account executive.


Defining Process Steps and Gathering Services Requirements
SAP
Search Customers
Search Sales Orders
Display Customer Details
Search for customers by name, address, city..
Search for Sales Orders by number, date, PO,..
Display Customer details including Account Executive information
Use existing Service
Use existing Service
New Service required
Enterprise Services
ERP
CRM
Backend

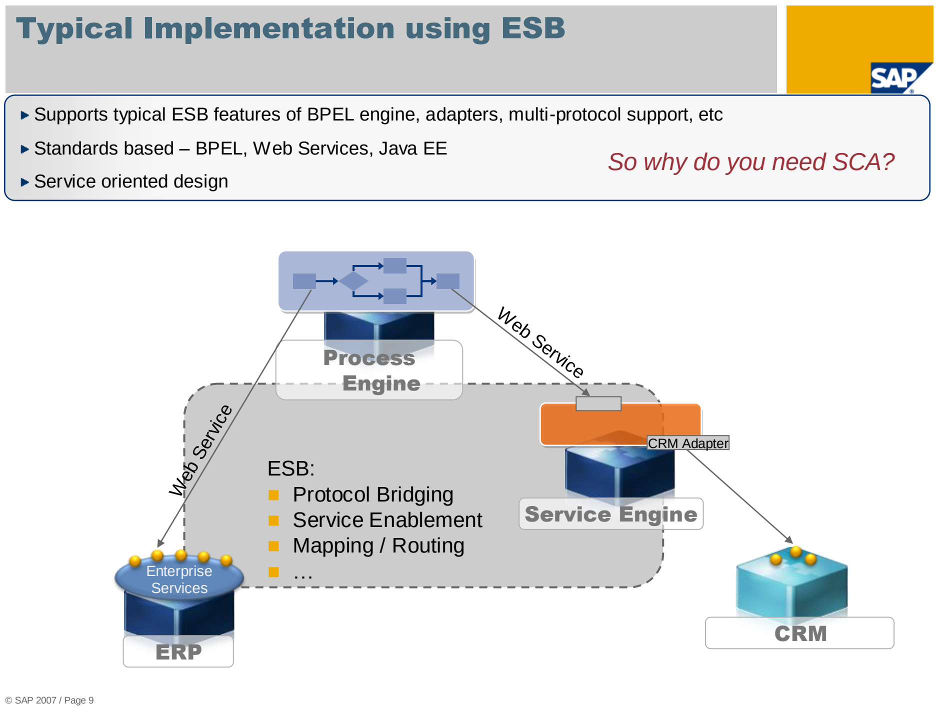
© SAP 2007 / Page 8

Typical ESB based Implementation

A typical implementation of the above scenario would involve using a orchestration engine for hosting the activities to invoke the existing ERP Web service and the newly created service that wraps around the functionality offered by the CRM system. The newly created service may offer itself as a Web service where as it may be communicating with the CRM system using its native protocol through an adapter.

So, where is ESB in this picture? Again, different vendors may have taken different approaches in bundling and tagging their concerning products as an ESB. Some products take an approach of considering ESB as a piece of infrastructure that supports various standard protocols as well as adapters to existing systems. Other vendor may include a service registry along with some design time tools for creating and publishing services. Yet another vendor may bundle in the mix a process engine, services for mapping/routing/rules, etc. In an extreme case, a vendor may put the entire platform offering including application servers, etc under the ESB umbrella.

Nevertheless, a typical implementation of the previously described problem scenario would look as follows:



Typical Implementation using ESB

- Supports typical ESB features of BPEL engine, adapters, multi-protocol support, etc
- Standards based – BPEL, Web Services, Java EE
- Service oriented design

So why do you need SCA?

So the question then becomes if the ESB technology is enough to support the common scenarios, why do we need an SCA standard in the current context? What is the benefit of standardization here? For answering this question, let us look at a few use cases where while a proprietary ESB based solution may be sufficient, there are good reasons for having a standard ESB programming model.

Use Cases for SCA

Use Web Services when necessary

It is not uncommon for vendors to provide the various service engines including process engine, Java EE based application server, etc, as part of the same platform. The different engines may even be hosted in the same physical process. In our example, the process engine and the container (say EJB) that hosts the service interfacing with the CRM system may very well be part of the same Java VM. In such cases, use of Web services technology for wire level interoperability is costly and unnecessary; and it makes sense to allow for use of other optimal communication mechanisms.

Why standardize?

Now there are products out there that allow use of Web services technology for design time and use of other optimized communication protocols during runtime. However, without a standard for handling this scenario, the solution becomes limited to where Web services are used for design time and the solution also becomes specific to the product features.

The benefit of standardization here is to promote a standard architecture and a consistent programming model whereby components using different interface technologies (e.g. Java, C++, WSDL) can be easily composed together. By using a standard model, developers can learn the concepts once and be productive in any environment supporting that standard (which translates to lower cost of development and maintenance).

End-to-end Quality of Service Modeling

When it comes to supporting QoS aspects such as security and reliability, it is important to address these concerns for the entire solution and not just for one or two parts of the solution. A solution that requires on-the-wire confidentiality should make sure that all the hops of the message exchange are secure. In our example, the connection between the process engine and the service engine (that connects with the CRM system), as well as the connection between the service engine and the CRM system, must support the same QoS attributes.

Typically, end-to-end QoS support is handled by enabling the required QoS aspects for each and every hop separately. This approach depends upon bridging the QoS models on the two ends of each and every hop separately, resulting into a brittle solution where managing any change to QoS aspects would be cumbersome and error-prone.

Why standardize?

In this situation, it is highly desirable to standardize upon a framework that allows for declaring the various QoS requirements as abstract policies and allows for the different application platforms and communication mechanisms to provide for the same. Such a standard model would not just simplify supporting end-to-end QoS in an efficient manner, it will also enable new runtimes to be part of the IT landscape and be readily utilized.

Managing Changes to Services

In today's dynamic business environment with many mergers and acquisitions and business decisions such as to outsource services to partners or use SaaS, cloud based services, etc, it is important to have flexibility and agility for quickly implementing changes to the business solutions.

Why Standardize?

Web services provide some benefit in this regard, that is, as long as the service contract is specified in WSDL, the service implementation or location may be changed. However, if the new service provider happens to be using some other communication/interface technology, it becomes harder or expensive to manage the change. Having a standard architecture that allows for changes to services irrespective of the service interface and access technology improves the flexibility of the system greatly.

Gathering management data

Deploying a solution involving many services, heterogeneous systems, etc, means that a new set of service dependencies is coming into existence for the IT to manage and worry about. Unless the service dependency information is proactively and meticulously fed into the management tool, it is difficult to get a reliable answer to question such as – what happens when a specific service is shutdown for some time, etc.

Why standardize?

By using a standard for modeling the composition of business applications on top of the underlying service-oriented components, it becomes possible or at least easier for the management tools to capture and track the service dependencies.

Tolerance towards new runtimes/infrastructure

IT shops regularly face the challenge of coping with new communication technologies, new interaction paradigms, and new languages for application development. It is not easy to enable the IT team to leverage the new technologies without disrupting the existing applications.

Why standardize?

While there may be good reasons motivating the adoption of new technologies, it will always be desirable to minimize the learning curve for the developer, deployer and other IT roles. By standardizing on an abstract programming model for components and configuration of the infrastructure, it becomes simpler for the IT shops to assimilate variety of technologies into the landscape.

SCA for ESB Programming

SCA Model and Specification Areas

Let us now look at how to model the solution for our business problem (Sales Order Tracking) using the SCA technology. Let us do a quick recap of the different SCA concepts and specification areas.

SCA allows taking a view of the end-to-end solution as a single application called – a composite. A composite comprises of individual components that are wired together.

For modeling assembly of the composite – see the SCA Assembly specification.

For developing components in different languages that can be part of the SCA composites – see the SCA Client & Implementations specifications in Java, BPEL and C/C++.

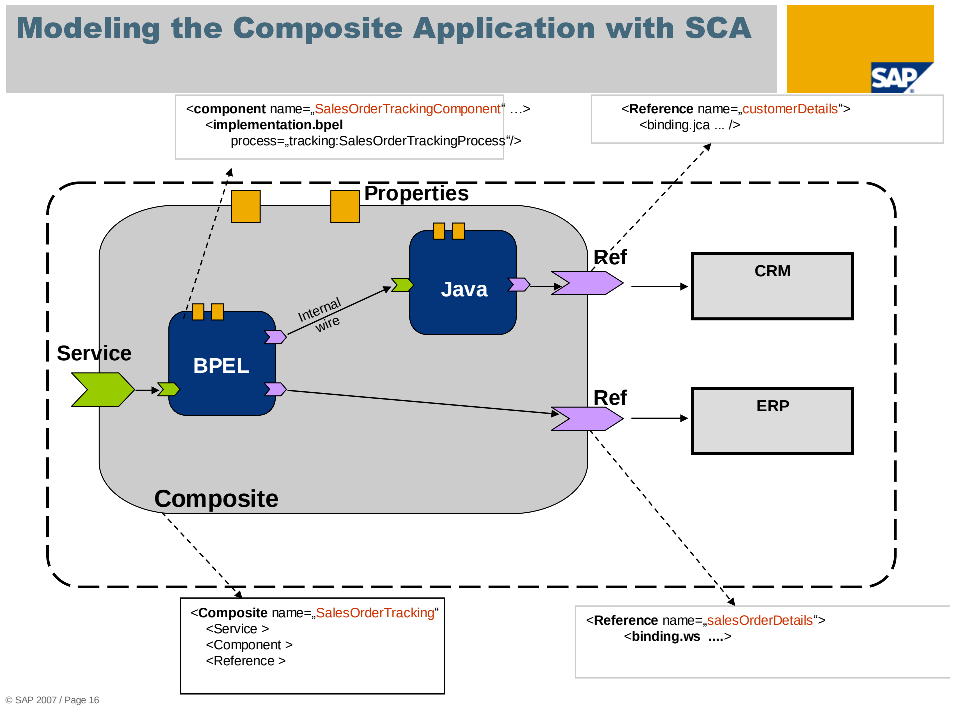
For a standard model for configuring the various communication mechanisms – see the respective SCA Binding specifications.

For the framework for declaring abstract QoS policies and configuring the concerning infrastructure that provides the policies – see SCA Policy Framework.

Modeling of SCA Composite

In our example, there are two components – a> a process component that invokes the services provided by the ERP system and service that interacts with the CRM system and b> the component that offers a service by adapting the functionality provided by the CRM system and applying some additional logic. See the example scenario discussed earlier.

There are also configurable properties that the two components expose for conrolling their runtime settings and behavior. So in essence, there are two aspects of configuring a component – its properties and the services it depends upon. Note also that the external services are modeled in the system in order to allow for the client side configuration. Similarly, the concept of a wire connecting different components is now a first class concept to which the designer can attach QoS requirements in abstract terms.



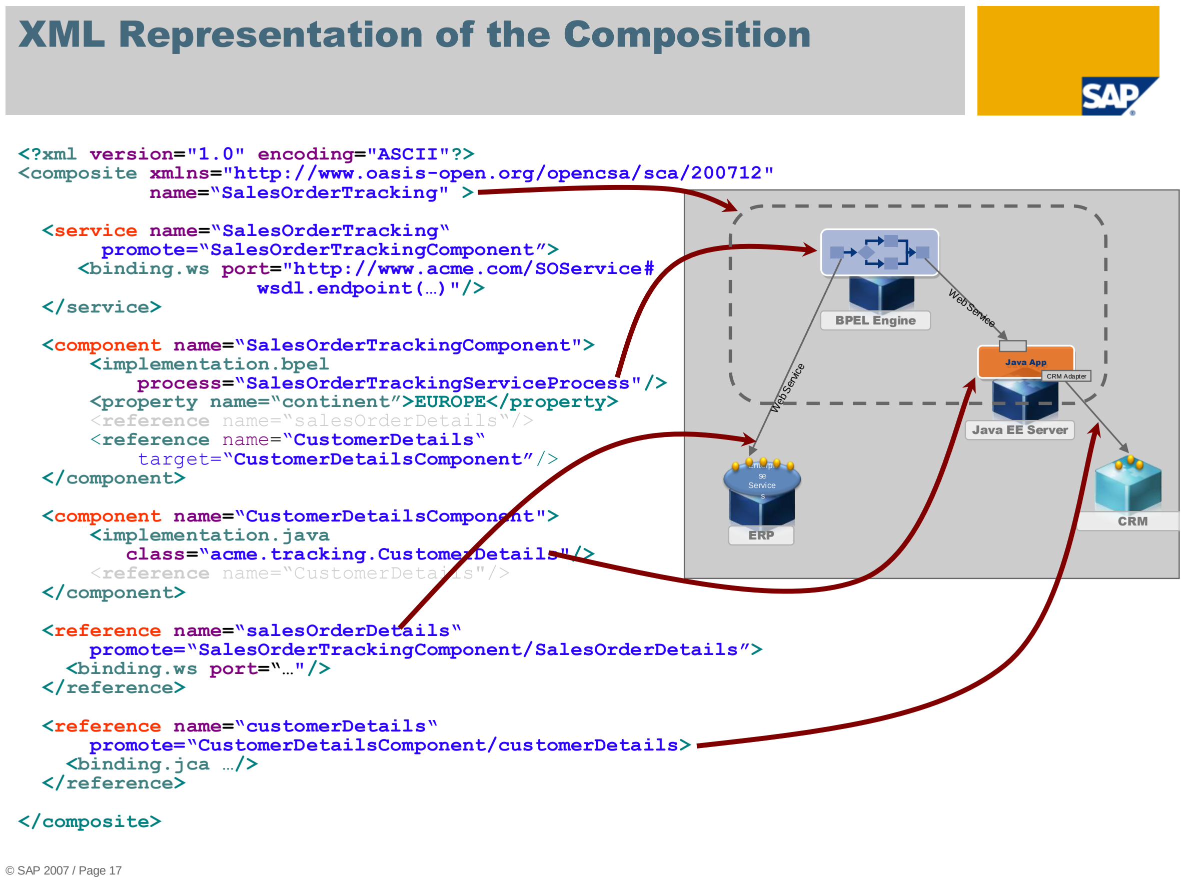
```
<component name=„SalesOrderTrackingComponent“ …>
    <implementation.bpel
        process=„tracking:SalesOrderTrackingProcess“/>
```

```
<Reference name=„customerDetails“>
    <binding.jca ... />
```

```
<Composite name=„SalesOrderTracking“
    <Service >
    <Component >
    <Reference >
```

```
<Reference name=„salesOrderDetails“>
        <binding.ws  ....>
```

Here is an XML representation of the composite:



XML Representation of the Composition

```
<?xml version="1.0" encoding="ASCII"?>
<composite xmlns="http://www.oasis-open.org/opencsa/sca/200712"
           name="SalesOrderTracking" >

  <service name="SalesOrderTracking"
       promote="SalesOrderTrackingComponent">
     <binding.ws port="http://www.acme.com/SOService#
                      wsdl.endpoint(…)"/>
  </service>

  <component name="SalesOrderTrackingComponent">
      <implementation.bpel
          process="SalesOrderTrackingServiceProcess"/>
      <property name="continent">EUROPE</property>
      <reference name="salesOrderDetails"/>
      <reference name="CustomerDetails"
          target="CustomerDetailsComponent"/>
  </component>

  <component name="CustomerDetailsComponent">
      <implementation.java
         class="acme.tracking.CustomerDetails"/>
      <reference name="CustomerDetails"/>
  </component>

  <reference name="salesOrderDetails"
      promote="SalesOrderTrackingComponent/SalesOrderDetails">
    <binding.ws port="…"/>
  </reference>

  <reference name="customerDetails"
      promote="CustomerDetailsComponent/customerDetails>
    <binding.jca …/>
  </reference>

</composite>
```

© SAP 2007 / Page 17

SCA Runtime

Now let us ask the question – what is the runtime for the SCA composite that is an assembly of components built using heterogeneous languages/platforms wired together using different communication technologies. The SCA Runtime is nothing but the different containers plugged together to the ESB.

Deployment of the SCA composite is equivalent to deployment of the different components to their respective containers, and configuration of the infrastructure to connect the clients with the services while ensuring that the QoS aspects are satisfied, etc. The beauty of the SCA approach is that – there is now a standard, high level model for design, development, composition and deployment of the end-to-end solution. This is nothing but support for programming-at-large for the ESB using the SCA standard!

Summary of the Benefits of SCA for ESB Scenarios

SCA is neutral to programming languages and communication technologies. It does not require yet another interface technology or communication mechanism, but it rather allows for accommodating the different existing technologies in a standardized manner. When no standard protocol is necessary to be used, SCA allows for declaring those interactions as internal to the SCA domain, what that means is – a proprietary, optimized communication mechanism could be used for those internal wires as long as the QoS requirements are met.

SCA does not require bridging of QoS models on a per hop basis. Instead, QoS requirements are centrally captured in the big picture of the composition, which are then translated in terms of the configurations of the individual communication infrastructures. This drastically simplifies the overall policy handling and leads to creation of manageable solutions.

Managing changes to services is inherently supported by the SCA design since wire is a first class concept in SCA. Scenarios such as separate deployment of wires and their re-deployment, etc, are explicitly addressed by SCA.

By providing a holistic view of the project, SCA makes it quite easy to provide the service dependency information to the management tools. For example, one of the steps in deployment of the SCA composite could be to feed the service dependency graph of that composite into the management tool's repository of service information.

SCA simplifies adoption of different new and old technologies. The neutral assembly and policy model and the framework for their binding to the different technologies make it possible for the different technologies to be part of an SCA application.

In essence, SCA provides a model for programming-at-large for the ESB and simplifies the use of ESB.

Related Content

OASIS Service Component Architecture / Assembly (SCA-Assembly) TC
Defining core SCA composition model to simplify SOA application development

OASIS Service Component Architecture / Policy (SCA-Policy) TC
Defining an SCA policy framework to simplify SOA application development

OASIS Service Component Architecture / Bindings (SCA-Bindings) TC
Standardizing bindings for SCA services and references to communication protocols, technologies and frameworks

OASIS Service Component Architecture / BPEL (SCA-BPEL) TC
Specifying how SCA component implementations for SOA can be written using BPEL

OASIS Service Component Architecture / C and C++ (SCA-C-C++) TC
Standardizing C and C++ use within an SCA domain for SOA

OASIS Service Component Architecture / J (SCA-J) TC
Standardizing Java (tm) use within an SCA domain for SOA

Copyright