Relevance of the Service Component Architecture Standard to a Business Process Expert

Summary

The Business Process Expert (BPX) Community was recently announced by SAP and is gaining great momentum. This article attempts to draw the attention of the BPX Community to a highly relevant standardization effort, led by SAP and others, called service component architecture (SCA).[1]

One of the main challenges faced by a BPX is how to bridge the gap between the sketchy drawings of the business processes conjured by the business department with the myriad details of the actual implementation and deployment of the software artifacts owned by the IT department. By standardizing key metadata for configuring and assembling business processes on top of reusable, heterogeneous components, SCA simplifies the task of a BPX role in bridging the divide between the business and IT.

In this document, the role of the BPX is contextualized by identifying the relationship of the BPX with other roles in the business and IT departments. By identifying specific tasks of the BPX, the requirements for tools and technology in assisting the BPX are highlighted. Details of SCA along with examples are provided to drive home the point that the SCA standardization effort is highly significant to the BPX Community.

Author: Sanjay Patil

Company: SAP

Created on: June 9, 2006

Author Biography

Sanjay Patil is a standards architect at SAP Labs, Palo Alto. His areas of interest are business process management, business events network, messaging, Java/J2EE, and Web services. He currently represents SAP on several standards committees under OASIS, JCP, WS-I, and so on.

Table of Contents

Who Is a BPX?

This document attempts to describe the role of a BPX in terms of how a BPX relates to other roles in a typical organization and how the individual tasks of each role relate to each other.

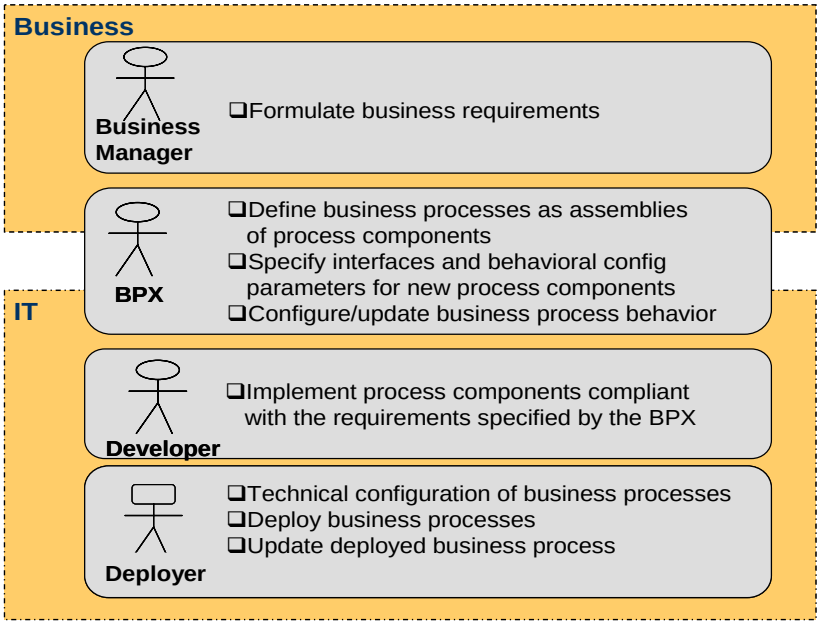

Fig 1. Role of BPX in Business and IT

The following is a description of the various roles and their tasks as depicted in Figure 1.

Roles

For the purposes of this document, the structure of an organization is assumed to be simple and consisting of merely two departments – business and IT. The business side sets the business objectives and drives the requirements that the IT department must fulfill. For the sake of simplicity, the business side consists of just one role – business manager – and the IT consists of two roles – developer and deployer.

Business manager: This role owns the business side and is committed to attaining high-level business goals such as driving revenue, improving customer satisfaction, reducing the cost of operations, and so on. This role drives the requirements the IT department is expected to fulfill.

BPX: Who is a BPX? This has been one of the first topics of discussion on the official BPX Web site [2]. Here are some excerpts borrowed directly from the material on the BPX Web site:

From the article Who is the Business Process Expert? [3]:

Business process experts have the business knowledge and IT savvy to make business process innovation happen in real time by adapting, composing, and executing end-to-end business processes using composition software tools and enterprise services.

From Mark Yolton's blog – Business Process Expert? What's the Big Deal? [4]:

They are the people who take business requirements, model them, and configure an application to perform those workflow actions to result in solutions. They are also often the people who take business requirements and then work with IT developers to customize or extend software to make it sing, thereby giving their companies a competitive advantage in speed, cost, or quality.

BPXs not only keep their eyes and ears open in the business world, but must also dip their toes into the technological, detail-oriented world of IT so that the business requirements can be unambiguously translated into software requirements for the IT department. The software requirements include both functional as well as nonfunctional aspects, such as security and reliability.

Developer: This IT role is responsible for developing software components. A developer is an expert in coding business logic and might not necessarily understand the larger business context in which the software components would be used. It is important that the functional and technical requirements on the software component are clearly communicated to this role, preferably in machine-readable form (by using interface definition languages, schemas, and so on).

Deployer: This role is responsible for deploying the software packages produced by the developer along with artifacts provided by the BPX, if any. A deployer makes the end-to-end business process available for the end users and updates the deployments whenever any change is to be made. This role understands the platform and infrastructure on which the software is deployed. In particular, a deployer must understand the different capabilities of the platform to meet the technical requirements of the business processes such as life-cycle management, assigning network addresses to business processes for external consumption, enabling secure and reliable communication for intraprocess or interprocess interactions, and so on. This role might not necessarily understand the semantics or the business-level relevance of the deployed applications. However, this role must be provided with clear information about the technical requirements, dependencies, and so on, preferably in machine-readable form to minimize the chances of deployment errors.

Tasks and Requirements of Different Roles

The order in which the various tasks of the different roles are concerted may vary per scenario. For example, in a project where the assembly created by the BPX consumes only preexisting software components, there is no need to bring the developer role into the picture. However, if the assembly created by the BPX includes definition of new components, involvement of the developer role might be paramount. For the sake of getting a full picture and understanding all the tasks associated with the different roles shown in Figure 1, let us consider a comprehensive scenario of top-down development.

In this top-down development scenario, the business department has decided to grow by supporting certain new business processes. Let us assume that brand new software components are to be developed for realizing the new business processes and that new deployments have to be made. The following steps describe the tasks carried out by the different roles engaged in this top-down scenario.

1. *Task*: Formulate business requirements

   Role: Business manager

   Description: Based on the new business challenges, expansion plans, and so on, the business manager succinctly formulates the requirements of the IT. A BPX works with the business manager to clarify the intended semantics, constraints, and so on of the new business processes.

   Requirement: Tools and technology for the business manager to specify the intended semantics of and constraints on the business processes.

2. *Task*: Define business processes as assemblies of reusable components

   Role: BPX

   Description: Based on the information about the new business process provided by the business manager, the BPX applies knowledge and experience and models each business process as an assembly of components. Each component in an assembly stands for a discrete functionality. Instead of building monolithic, point solutions, the BPX thinks strategically and factors each business process into a set of reusable components. A BPX knows which components exist in the IT and which need to be freshly implemented. This knowledge is critical for the BPX to avoid wasting effort in building redundant components. The individual components defined by the BPX can be implemented in-house or outsourced. Each component can be developed and deployed in a different language and runtime environment. The implementation of one component may be substituted by a new implementation in the future or even by a partner-provided service, as long as the externally visible contracts are not violated. These are all classic reasons for why one should not build monolithic, rigid systems.

In this step, the BPX is not actually coding the business logic of the individual components, but is merely specifying the contract on each component.

Requirement: Technology for the BPX to assemble business processes on top of components that require minimum know-how about the component's implementation details or the middleware and infrastructure needed for accessing the component functionality.

3. *Task*: Specify interfaces and behavioral configuration parameters for new process components

   Role: BPX

   Description: For each new component in the assembly of a business process, the BPX specifies in detail the contract on the component in terms of the functional interface the component should provide as well as the set of behavioral configuration parameters the component must expose to the BPX. The latter aspect is somewhat new to the enterprise service-oriented (SOA) community, although the pattern of injecting configuration values in a component for affecting its behavior is quite common in many existing component runtimes today, and is commonly referred as dependency injection [5].

There is an inherent conflict between reuse of a component and stability of the component interface. As our experience with enterprise SOA grows, it has become evident that except for some relatively straightforward scenarios, it is quite challenging to slice and dice components with the correct functional granularity. The current shape and size of a component based on a set of existing requirements might be at odds with some future scenarios. There might even be a conflict among the various current requirements, leading to a component interface that is either an awkward aggregation of unrelated operations, or has operation signatures cluttered with unrelated parameters. This challenge of defining component interfaces that are coarsely grained and are still useful to all the current and future consumers of the component is very real. Taking shortcuts and ignoring this challenge typically leads to brittle systems, in spite of religiously applying all the principles of enterprise SOA. An approach to address this challenge is to identify and expose the behavioral configuration parameters of the components in addition to the definition of the functional interface. The behavioral configuration parameters typically consist of a set of properties and references to external services consumed by the component implementation. By setting usage-specific values for the properties and service references, the behavior of the component can be customized.

The BPX is also expected to understand the quality of service (QoS) requirements on the component, such as security and reliability. The BPX does not have to specify the technologies to be used for meeting these requirements. It is the task of the deployer (see step 6, below) to understand the abstract QoS parameters specified by the BPX and bind them to suitable concrete technologies available in the IT. It is important to allow the BPX to specify the abstract QoS parameters in an unambiguous (and machine-readable) manner so that the deployer can independently choose and bind the suitable technologies.

To summarize, the behavioral configuration parameters and the abstract QoS parameters define the set of metadata that allows the BPX simultaneously to disengage from the mind-boggling details of the technology world and still meaningfully connect with the IT department. A BPX should not get sucked into the implementation or deployment details of each component, but, at the same time, a BPX should know just enough about these components to assemble them into business processes.

Requirement: The externally visible metadata of reusable components should include not only the definition of functional interfaces but also the declaration of behavior configuration parameters and abstract QoS parameters.

4. *Task*: Implement process components compliant with the requirements specified by the BPX

   Role: Developer

   Description: Depending upon the programming language and runtime environment of choice, the developer might have different tools and technology available for implementing a process component. The features provided by selected component implementation environment can vary from simple scripting to advanced features such as application programming interfaces (API) for handling long-running,

asynchronous interactions. Given the focus of this document on the BPX, details relevant to the developer role are not included here.

Requirement: Tools and technology to facilitate the development of process components compliant with the contract specified by the BPX, which includes definition of functional interfaces, behavioral configuration, and abstract QoS parameters.

5. *Task*: Configure business process behavior

   Role: BPX

   Description: It is likely that this step is skipped and the configuration of the business process defined in step 2 by the BPX role is sufficient to proceed to the next step. However, typically, the outcome of step 2 is a broad-ranging definition of the business process, where the different possible values for the configuration and QoS parameters are outlined, but no specific choices have been made. In the current step, the BPX chooses specific values for these parameters.

   Requirement: Rules for verifying that the underlying components comply with the specified contracts and assist the BPX in setting values for the configuration and the abstract QoS parameters.

6. *Task*: Technical configuration of business processes

   Role: Deployer

   Description: At the end of the previous step, a semantically complete definition of the assembly is handed off to the deployer. In the current step, the deployer inspects the assembly for abstract QoS parameters that must be bound to concrete technologies. An assembly includes wires between components that provide and consume services. A deployer must therefore configure the QoS aspects of both ends of the wires.

   Requirements: Tools and technology to facilitate binding the provider and consumer end points of each wire to the infrastructural technologies, such as security and reliable messaging.

7. *Task*: Deploy business processes

   Role: Deployer

   Description: The previous step beefs up the assembly definition with QoS–specific bindings. In the current step, the deployer takes care of the physical deployment of all the process components and making the end-to-end business process available for consumption to the end users. The deployer analyzes the business process assemblies and arrives at a repertoire for how each component must be deployed and in which order. The order of deploying the components is important, because certain information on how a consuming component can access a provider component might become available only after the provider component is physically deployed.

   Requirement: A well-defined methodology for analyzing the business process assemblies and arriving at a repertoire for deploying the process components.

8. *Task*: Update business process behavior

   Role: BPX

   Description: Certain changes in the business (communicated by the business manager) might require updating the business processes. Hopefully, the possibility of such changes has been anticipated by the BPX when creating the assemblies in the first place (step 2 above). If the possibility of change has been factored into the design, affecting the change to the business process is typically achieved by updating certain behavior configuration or abstract QoS parameter values. Such a change may be communicated by the BPX to the deployer either by creating new assembly definitions or by specifying only the delta between the currently deployed business process definition and the intended new business process definition.

   Requirement: Similar to that of steps 2 and 5, above.

9. *Task*: Update deployed business process

   Role: Deployer

   Description: Taking the changed business process or just the delta of the change, the deployer must now plan how to update the deployed process. There are several different strategies the deployer could choose from. Deployment strategy–specific details are not covered here, because this document focuses on the BPX role.

   Requirement: Tools that assist in formulating steps for affecting changes to the deployed processes, typically without breaking the continuity of the business processes that are running.

SCA – How Does It Relate to BPX?

Now let us look at what SCA has to offer the BPX role. Here is a summary of the requirements of tools and technology imposed by the typical tasks (specifically tasks described in steps 2, 3, 5, and 8 from the previous section) carried out by a BPX:

- Technology for the BPX to assemble business processes on top of existing components, technology that requires a minimum level of know-how about the component's implementation details or the middleware and infrastructure needed for accessing the component functionality.
- The externally visible metadata of reusable components should include not only the definition of functional interfaces, but also the declaration of behavior configuration parameters and abstract QoS parameters.
- Rules for verifying that the underlying components comply with the specified contracts and to assist the BPX in setting values for the behavior configuration parameters and the abstract QoS parameters.

SCA addresses the above requirements of the BPX by defining a model and a platform and language–neutral syntax for the concepts of component type, component (instances of component type), and composite (assembly of components into business processes). Other parts of SCA are aimed specifically at the developer and deployer roles: the details of which are not covered in this document.

Component Type

As noted earlier, the BPX needs the ability to customize the components by setting specific values for the behavior configuration parameters as well as abstract QoS parameters. The notion of component type precisely addresses this requirement by embellishing the functional interface with a set of properties and service references that can be used for configuring the component. Let us look at an example snippet of a component type:

```
<?xml version="1.0" encoding="ASCII"?>
<componentType xmlns="http://www.osoa.org/xmlns/sca/0.9">
  <service name="TravelService">
      <interface.wsdl interface="http://www.sap.com/TravelService#wsdl.interface(Travel)"/>
  </service>
  <reference name="bookingService">
   <interface.wsdl interface="http://www.sap.com/BookingService#wsdl.interface(Booking)"/>
  </reference>
  <property name="continent" type="xsd:string"> NorthAmerica</property>
  <property name="currency" type="xsd:string> USD </property>
    </componentType>
```


In this example, the contract for a travel service component type is defined. As per this contract, an implementation of this component type must:

- Implement the functional interface, “travel,” defined in the Wed Service Definition Language (WSDL) and identified by the namespace http://www.sap.com/TravelService.
- Consume a service with a functional interface, “booking,” defined in the WSDL and identified by the namespace http://www.sap.com/BookingService.
- Expose configuration properties for identifying the continent in which the travel service operates and the currency used by the travel service.

The component type definition does not mandate the use of any particular implementation language or runtime platform. This feature allows a lot of flexibility for the IT department to leverage its existing assets.

Composite

SCA defines how to model business processes by assembling instances of different component types (called components).

Figure 2 depicts the SCA model of a composite.

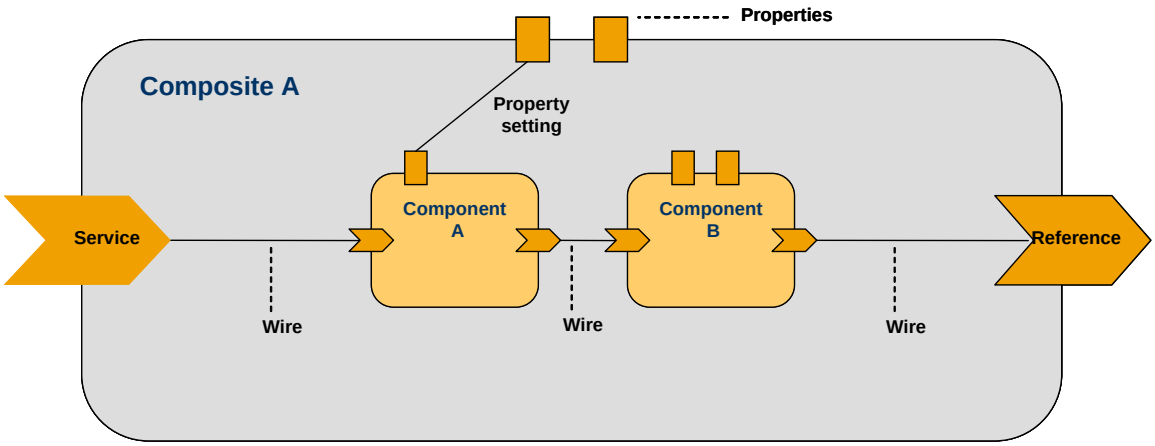

Fig 2. Business Process – An Assembly of Process Components

A composite is an assembly of a set of components related to each other by means of the wires connecting them. The wires are defined by setting the service references of the consuming components with values of the services provided by other components in the composite or services external to the composite.

A composite includes configuration of the properties of the assembled components.

A composite also identifies the services that are accessible to the external consumers. From an external consumer’s perspective, a composite offers a set of services, consumes a set of external services, and declares a number of properties to be set. This definition might sound similar to the concept of component type that was introduced earlier. The similarity is intentional and allows for recursion: a composite may be built by consuming other composites, thus enabling the BPX to model complex, nested assemblies.

Once a BPX creates a composite and hands it off to the deployer, the deployer can contribute the deployment-specific information to the composite, primarily the binding of wires to concrete protocols and technologies for fulfilling the required QoS.

Example of a Composite

To get a clear picture of a composite, let us take an example of a travel service. The following diagram (Figure 3) shows the schemata for the composite.

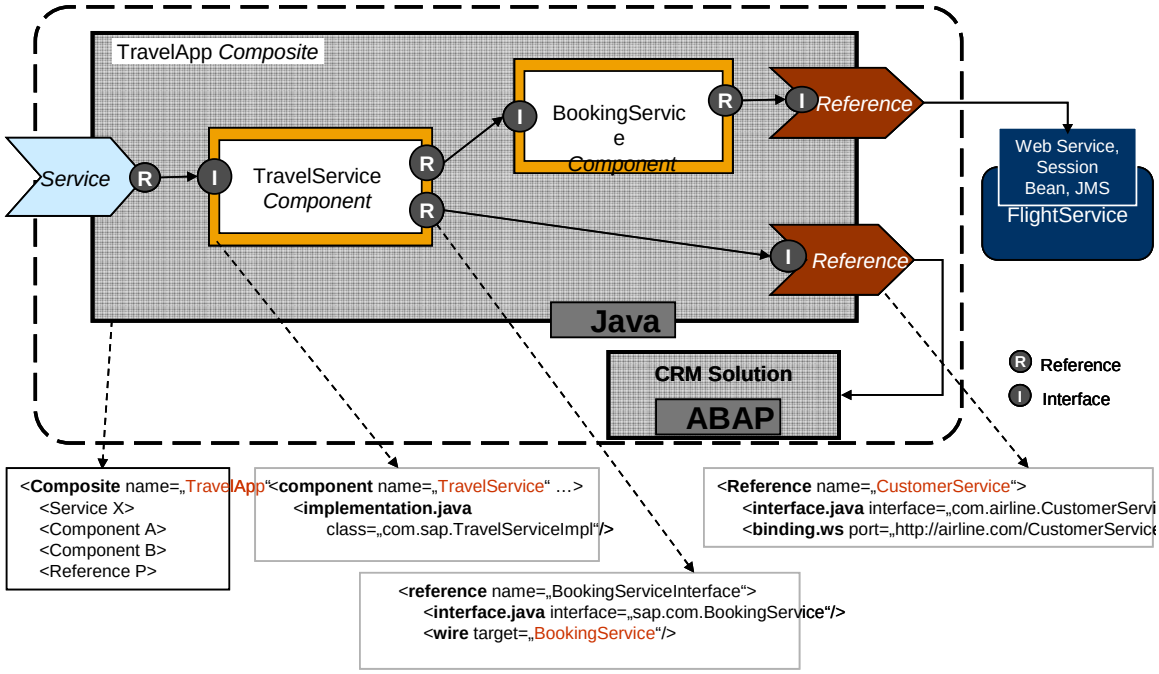

Fig 3. Example of SCA Composite – Travel App

Figure 3 also shows XML snippets of some of the elements of the composite. The following code puts the snippets together into a more complete XML representation of the composite.

```
<?xml version="1.0" encoding="ASCII"?>
<composite xmlns="http://www.osoa.org/xmlns/sca/0.9"
          name=“agency.TravelApp" >
 <service name=“TravelService">
  <interface.java interface="services.travel.TravelService"/>
  <binding.ws port="http://www.agency.com/TravelService#
         wsdl.endpoint(TravelService/TravelServiceSOAP)"/>
  <reference>TravelServiceComponent</reference>
 </service>
 <component name=“TravelServiceComponent">
  <implementation.java class="services.travel.TravelServiceImpl"/>
  <property name=“continent”>EUROPE</property>
  <reference name=“bookingService" target=“BookingServiceComponent"/>
  <reference name=“customerService" target=“CustomerService"/>
 </component>
 <component name=“BookingServiceComponent">
```



```
  <implementation.java class="services.travel.booking.BookingServiceImpl"/>
 </component>
 <reference name="CustomerService">
  <interface.java interface="services.travel.CustomerService"/>
  <binding.ws port="http://www.customer.com/CustomerService#
        wsdl.endpoint(StockQuoteService/CustomerServiceSOAP)"/>
 </reference>
</composite>
```

In this example, a BPX models a composite for a travel service business process. The high-level functions of this business process involve:

- Managing customer information
- Booking flights
- Coordinating user interaction and the overall flow

In this example, the business manager decides on direct use of an existing system that takes care of customer information management. For booking flights, the business manager decides to use external services provided by the airline companies. The airline companies offer special prices and discounts to this agency.

The BPX thinks that a new component is needed to wrap the flight booking service provided by the airline companies to capture the agency-specific business logic and data for the promotions, discounts, and so on. For managing the overall coordination of the different steps of the travel reservation, the BPX decides to have a separate component. In essence, the composite defined by the BPX consists of two components:

- Travel service, to coordinate the overall flow of the business process for travel reservations
- Booking service, to wrap the external flight booking service with agency-specific business logic and data

For actual implementation of the components, the developer may choose to use any suitable technology. For example, since the key challenge of the travel service component's business logic is coordination of the overall process flow, the developer may decide to use Business Process Execution Language (BPEL).[6] But because the booking service component functionality involves some algorithm-intensive business logic as well as elaborate data processing for managing the discounts, promotions, and so on, the developer may choose to use a Java/J2EE platform (or even a rules engine). The bottom line is that the developer's choices for component implementation are independent of how the BPX models the composite and defines the component contracts. The developer and the BPX share certain metadata and architecture in common, but both retain their own freedom.

Conclusion

The BPX role bridges the gap between the business and the IT organizations and is therefore crucial in materializing the promises of enterprise SOA. It is important that tools and technology be provided to spare the BPX from the details of the IT, but still allow forging traceable and repeatable contracts between the BPX and other IT roles. SCA is a standardization effort led by SAP and others to simplify the tasks of a BPX by defining the correct level of IT metadata and models.

It is important that the BPX Community remains engaged with the development of the SCA standards, at least by reviewing the SCA specifications and by providing feedback.

Further Information and Contacts

The SCA standard is currently under development and is not yet submitted to any standards body. The earlier drafts of this specification can be viewed at the SDN link for SCA [1].

The aspects of SCA relevant to roles other than BPX, such as developer and deployer, are not addressed in detail in this document. You may contact Sanjay Patil (sanjay.patil@sap.com) for additional information on SCA.

Acknowledgements

Martin Raepple helped in reviewing and improving this document significantly.

References

[1] SAP links for SCA

A joint white paper – https://www.sdn.sap.com/irj/servlet/prt/portal/prtroot/docs/library/uuid/97ecd5c6-0701-0010-a78e-c1cb74a3da47

Assembly specification, version 0.9 – https://www.sdn.sap.com/irj/servlet/prt/portal/prtroot/docs/library/uuid/092dddc6-0701-0010-268e-fd61f2035fdd

Big bank example – https://www.sdn.sap.com/irj/servlet/prt/portal/prtroot/docs/library/uuid/0e6cdbc6-0701-0010-cdbf-cd6dbdb9983a

Other technical specifications of SCA:

Java client and implementation, version 0.9 – https://www.sdn.sap.com/irj/servlet/prt/portal/prtroot/docs/library/uuid/7731d8c6-0701-0010-8d88-ec16175ac180

C++ Client and implementation, version 0.9 – https://www.sdn.sap.com/irj/servlet/prt/portal/prtroot/docs/library/uuid/0904dac6-0701-0010-b5bd-a72ea20c5fb0

[2] Business Process Expert Community

https://www.sdn.sap.com/irj/sdn/developerareas/bpx

[3] Who is the Business Process Expert?

https://www.sdn.sap.com/irj/sdn/developerareas/?rid=/webcontent/uuid/cc0bd202-0b01-0010-9fbf-da3953eed4c2

[4] Mark Yolton's blog – Business Process Expert? What's the Big Deal?

https://weblogs.sdn.sap.com/pub/wlg/3637

[5] Blog on dependency injection pattern and its use in SCA

https://www.sdn.sap.com/irj/sdn/weblogs?blog=/pub/wlg/2932

[6] OASIS WS-BPEL TC

http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsbpel

Copyright