TIBCO Proposal for SCA Eventing

Scott Vorthmann
Sabin Ielceanu
Pradeep Simha

version 1.0
November 25, 2009

1. Introduction

This document captures TIBCO's proposal for supporting eventing and publish-subscribe use-cases in SCA 1.2. The document is written as a "delta" against the proposal document already submitted to OASIS SCA Assembly TC by IBM, Oracle, and SAP, defining the key parts of that proposal TIBCO recommends changing. This document should not be considered to be an exhaustive list of the changes TIBCO might propose, but it certainly touches the high points.

2. Goals of the Proposal

TIBCO has three overarching goals for this proposal:

minimum-novelty - The SCA support for eventing and pub/sub should be accomplished without defining new standardization where there are existing standards that meet the needs.

developer-model - Implementation languages like Java and BPEL already support idioms for responding to or generating events in a very natural fashion. It should not be necessary to extend those idioms.

channel-metaphor - Obviously, we desire to give the application assembler a metaphor for composing applications that supports the decoupling inherent in pub/sub and eventing mechanisms. At the same time, we want to stay true to the SCA design paradigm of visible interaction patterns and encapsulation.

3. Mapping ComponentType to Component

The key to achieving our ***developer-model*** goal lies in the gap between the componentType of an implementation (authored by a developer) and the component (authored by an assembler) using that implementation. Our proposal exploits that gap by defining mapping rules between services and consumers, and between references and producers.

The effect is that the assembler can work with the model of channels, consumers, and producers, clearly distinguishing those relationships from the service wiring relationships with different semantics, while the developer can continue to ignore the mechanics of how events/messages are delivered to the implementation or accepted from it.

Another benefit is that existing annotation mechanisms for implementation languages like Java and BPEL can continue to serve, without needing extensions to support the new assembly metaphor.

While we believe that SCA could offer the channel metaphor without introducing distinct models for consumer and producer, we do accept that the latter will help the composite author or reader to understand the semantics better, as well as providing a natural place to define filters (on a consumer) without introducing the concept for services in general. However, we still insist that it is not necessary for an implementationType to have specific support for consumers and producers, other than the obvious requirement that implementation.composite must support them.

3.1 Exposing a Service as a Consumer

When constructing a component using a particular implementation, an assembler can define a consumer, and select the name of any componentType service that has some one-way operations. The name of such an operation is understood to define an event type, with the event shape dictated by the XSD metadata associated with the input message.

Non-one-way operations on the componentType service are simply ignored in this mapping process; the consumer will define event types or listen for related events.

By necessity, service and consumer names are pooled in a single set; no service name on a component, componentType, or composite may match a consumer name on the same artifact.

<this section needs an example>

3.2 Exposing a Reference as a Producer

When constructing a component using a particular implementation, an assembler can define a producer, and select the name of any componentType reference that has only one-way operations. The name of such an operation is understood to define an event type, with the event shape dictated by the XSD metadata associated with the input message.

ComponentType references whose interfaces contain non-one-way operations are not available for defining producers.

By necessity, reference and producer names are pooled in a single set; no reference name on a component, componentType, or composite may match a producer name on the same artifact.

<this section needs an example>

4. Interfaces for Event Type Grouping

SCA already has a nicely extensible notion of interface metadata. This notion is easily repurposed to define a concept of event type groups. This approach has several advantages.

First, the existing "canonical" form for metadata, WSDL 1.1, remains canonical regardless of whether channels, consumers, and producers are involved or not. This simplifies the mapping outlined above in section 3, as well as simplifying the user experience.

Second, we adhere to our ***minimum-novelty*** goal, and avoid defining a new syntax for equivalent metadata.

Finally, we gain a reuse mechanism. A WSDL portType serves as a convenient group of event types. In a composite where a number of producers and/or consumers refer to the same set of event types, this approach is less cumbersome to the user.

Note that this approach does not insist on exact match for portTypes; it is still the event types (operation names) that matter in performing an analysis of the composite. Just as in our service / reference portType compatibility rules, different portTypes can "overlap" by using the same operation names and message type metadata.

5. ChannelBinding Extension Point

The existing eventing proposal mentions bindings, without going into much detail. We believe that there is an equivalent extension point required, but we should not use "binding" as the name for it, to avoid confusion on the part of a user, and complexity in the SCA schemas. Let us call the extension point "channelBinding".

Naturally, we support the specification of a very simple "channelBinding.sca". It is possible that we would support a standard definition of "channelBinding.jms", assuming an acceptable degree of extensibility in its specification.

The channelBinding extension point probably requires three separate substitution groups in the schema: one each for channel, consumer, and producer. That said, since we did not bifurcate the binding substitution group for service and reference bindings, perhaps we could get by with just one, or perhaps two.

We will have to define matching rules for the channelBindings on a channel and any associated consumers and producers. For example, a consumer that consumes from two different channels, using two different channelBinding types, probably requires exactly one consumer channelBinding for each.

6. Filters

We feel strongly that it is not SCA's role to define event metadata other than the obvious requirement of event type. For that reason, we suggest that the notion of filtering on such metadata does not belong in the generic filter specification for a consumer or channel.

A channelBinding type might support the definition of filters on transport-specific metadata; these filters would appear in the consumer channelBinding, not in the generic filter list.

Generally, generic filters should be specified only in terms of the event type and/or the event shape infoset… the business data. Furthermore, just to be clear, SCA should offer no statements or guarantees about where filtering happens and how it affects network traffic, since there we should be making no assumptions about the implementation of any channelBinding type, including channelBinding.sca.

7. Domain Channels

The notion of channels defined at the domain level, used to connect different applications in the domain, is very important. However, we feel that the idea of "magically" referencing such channels without promotion, using the "//name" syntax, runs counter to the design principles and value proposition of SCA. First, it builds into applications assumptions about the state of the domain. Secondly, it introduces invisible, "back-channel" interaction between applications.