SCA Eventing Pub/Sub FAQ - 4th September 2009

1) Why do we need to change the current model?

We don't need to *change* the current model, we need to *add* to the current mode. In the current SCA 1.1 model, each invocation on a reference causes one invocation of the operation on one service provider; this is a typical client/server interaction where the client is requesting a specific service to do something (debit bank account for example.) Each operation has implied semantics, and the client is expecting a specific task to be performed by the service provider, possibly resulting in data being returned. When publishing an event, on the other hand, each event published may be received by zero, one or more recipients, with the originator unaware of how many recipients there are; the event is not addressed to any particular receiver. There are no implied semantics of what to do when an event is received and each recipient is free to process the event as it sees fit, including ignoring the event message. Thus events have no semantic contract between producer and consumer, unlike in a client/server model.

There is a difference between publishing the event 'WMD found' and invoking the method invokeWarMachine(). Subscribers of events can do whatever they want with an event instance or nothing at all. Different subscribers can react differently to the same event. There is no semantic contract that is externally available.

2) Why not reuse service and reference?

References wire or bind to services, but producers and consumers are not wired or bound directly to each other. Decoupling is an important principle in pub/sub as this allows publishers and consumers to come on and off stream during the lifetime of a system. Introduction of a new publisher or subscriber should not require redeploy/rewiring of other existing publishers/subscribers. In addition, services do not have cardinality and do not care who is wired to them. In an eventing system, the consumer might express where it wishes to receive events from. Rather than overload service and reference semantics we introduced the concepts well known and occur naturally in the pub/sub eventing world.

3) Why not reuse interfaces?

A producer can emit a message to multiple consumers, and each consumer may be interested in a different set of messages, thus defining a service interface would involve defining the set of all message types produced and consumed. This is a typical a MxN problem and is further made difficult by not necessarily knowing all the producers at design time. Contrast this to a client/server model, where implicit in the design is what service is being offered. Another way is to have the “service” interface defined by the producer, but this reverses the current model. This doesn't solve the problem of the subscriber (service) being interested in a subset of events that a producer produces. The current model requires the opposite, the reference-interface has to be a subset of the service-interface.
This would also require the consumer to know the interfaces of all the producers of the events it is interested in, this may be dynamic and is not very scalable. Furthermore in the current 1.1 model there are some implied semantics in the grouping of operations into an interface – collectively the operations are offering a service – but since there are no semantics between producer and consumer, grouping into interfaces doesn’t make much sense.

Additionally, a powerful concept in pub/sub is the ability to provide filters in subscriptions. Reusing interfaces creates an impedance mismatch with filtering since a message is only delivered if it passes the filter and the current 1.1 semantics do not allow an invocation to be ignored.

4) Why not just use JMS?

The proposal is designed as a thin configuration layer that permits the use of JMS, and other messaging systems in SCA Runtimes.
The model enables the use of most (all?) existing messaging systems. JMS is a Java API that can be used to access existing messaging/pub-sub systems, not a event model. The SCA model should be usable in BPEL, C/C++ and other languages. JMS also says nothing about how topics are modelled in a SCDL, how producers and consumers are connected or how the event related metadata is specified. The model allows for JMS to be used as a way to enqueue/dequeue/publish/subscribe messages, but does not require it.

5) Why do we need these extensions in SCA?

SCA is about configuring of composite applications. Being able to express and modify producers and consumers especially things like filtering and channel configuration, help extract configuration issues out of implementation code. For example, a consumer that reacts to stock prices going below $3 doesn't have to have implementation code changes if the requirement changes to $2. The price could be a configurable SCA property. The implementation code can freely implement the processing logic without worrying about configuration and assembler/deployers can focus on the configuration/deployment. This also allows a component to interact with the composite application using pub/sub or service-reference model. For example, when an event such as 'no WMD found' arrives at a component subscribe, the same component can invoke a service 'impeach' in a separate component. The usual separation of developer, assembler and deployer roles apply.

6) Why do we need channels?

To separate out eventing traffic into disjoint domains of interest, for example, US severe weather warning channel, and an Europe severe weather warning channel. A channel is the eventing analogy to a wire, allowing multiple producers to indirectly communicate with multiple consumers. A channel is thus a partitioning mechanism, in much the same way as JMS topics. It also provides the “destination”, which is the place to which a publisher sends the event to, unlike the client/server model where a request is sent to the service.

7) Can't we use java or wsdl as the payload definitions?

Java may be used to define event payloads when in a Java only sca environment, however as soon as components are introduced that are not java, an xml based payload description is required. Given that there is an impedance mismatch between an NxM eventing pub/sub and wsdl (there is not much reason to group events into a single interface, and its hard to determine the set of operations to group into an interface) not all of wsdl is deemed appropriate for the purpose of describing events. The most appropriate part of WSDL are the message definitions, but we need a way to associate event types with message definitions. Rather than extend and overload WSDL, we defined the optional EDL.

8) Why are there domain-level channels and composite-level channels?

Domain-level channels are visible to any component and composite in the domain, regardless of where they are in the composition hierarchy. In order to further support partitioning of areas of interest, composite channels are only visible to the components and composites defined with in the same composite as the channel. For example, if you want to disallow North American components from issuing events on the European Weather channel, define the European weather channel in its own composite together with the European components.

9) Why do we allow channel promotion?

We don’t. Channel promotion is not allowed in the proposed model, however promotion of producers and consumers are allowed to enable them to be reconfigured in outer scopes.

10) Why do we have namespace and type filtering?

Event types are qnames, and hence are defined relative to a namespace. Namespaces are controlled by an organisation, and there are some implied semantics in grouping definitions into a namespace, even if it is “these entities are defined by such and such an organisation”. Allowing one to filter on a namespace allows one to express an interest in receiving any event defined in that namespace without having to enumerate all of the event types defined in that namespace. This has the further advantage of allowing new types to added into the system and to be delivered if namespace filtering is used. If you do not want this type of wildcarding and flexibility, then one can always explicitly list the types of interest.

11) How does SCA interact with existing messaging implementation?

SCA pub/sub model submission does not mandate a particular implementation but enables the use of existing implementations. The model is meant to be overlaid over existing implementations without burdening existing implementations for conformance with SCA. For example, one could implement the producers and consumers using a bus architecture or a hub-and-spoke, channels can be mapped to JMS topics, filtering is allowed to be implemented at the producer end, subscriber end or some intermediary. One can implement the model using point-to-point or brokered notification. One can optimize (say in the CEP case) event message delivery by looking at the global view, which provides information about all the producers, subscribers, subscriptions and capabilities of the producers.

12) Can messages generated by external (non-SCA) producers be received by SCA consumers? And vice versa?

Yes, and yes! The SCA eventing model does not prohibit an event from a non-SCA source from being injected into a channel, not does it prohibit the delivery of an event to a non-SCA destination. Channels are abstraction of underlying messaging technologies, such as JMS, MQ, Rendezvous™ etc, so anything connected to the underlying infrastructure may publish and consume events.