Some Comments on CCA Event Specification Draft

Document Notes

Our comments are italicized – non-italicized text is copied from the event specification draft documents (for context). In the copied text, some words/phrases are bolded – these are addressed in the comments.

==========

Issues in Event Management

The main issue facing the event specification is that it must work in an HPC environment. In this environment, it is vital that the developers have **ultimate control over the amount of resources** that the event system takes.

Comment: *How does the current draft specification address this?*

The Publish/Subscribe Model

The **model that this specification adopts** comes from the Message Oriented Middleware domain. [..some words deleted.]
This hierarchy is organized much like a file system structure. This hierarchy is very important, as it allows one to subscribe to a whole group of topics at a time, since topics can be grouped together.

Comment: *Re: pub-sub model – seems very appropriate. Hierarchical and 'wildcardable' topics are a feature that greatly increases the power of a pub-sub technology. Once HPC application designers understand this, there might be no stopping them using the event service!*

As you say in footnote 3, wildcards do increase implementation complexity. If the aim of the event service is to be as fast, low-cost to implement/port, and scalable as possible, then it might be worth reconsidering wild cards as an optional/additional feature? This might be more important since the current draft has the event service as a singleton per framework – it's harder to scale singletons? For example, the JMS specification is silent on topic hierarchies, but several implementations support it. This could be left as an event service implementation issue?

Event Structure

Our events will have the following information: Message header: This contains the delivery information for the event. This is basic metadata for every event. What this contains at the very least is the destination of the event. More realistically, this would contain information about the sender of the event, an event ID, **and so on**.

Comment: *Two very useful headers features are correlation IDs and a 'reply-to' topic name. Not sure if you are anticipating supporting these – draft doesn't go in to that much detail?*

Topic Properties

Each topic has a number of properties associated with it. These properties allow a topic to be **local** to a single component, to have a set of **filters** associated with it, or for that topic to be **persistent**. We discuss each property in turn.

- Local Scale: This means that the events are only seen on a per component basis, not on a per-framework basis.
- Filtered components: This allows for filters to be run on the events when they are sent or received. The filtering runs on the event-service resource, not on the resource that is sending or receiving the events.

Comments:
Local scale *– not sure I understand this. Local to a single component? Why does it need to publish an event if local? We must be missing something obvious :-}*
Filters *– Do you mean filters as in 'selectors' in the JMS? If so, event filtering also imposes an increased load on the event service, especially if used extensively by multiple subscribers with different selectors. JMS (if memory served correctly?) for example restricts selectors to just operate on user-defined header fields of restricted types. It also requires a selector definition language to define the expressions for event selection. What sort of thing do you have in mind? This might be a feature we should think carefully about due its potential performance impact?*
Persistent *– Draft document doesn't cover this? Do you mean persistent as in (JMS terminology) logging events to disk-based queues to for reliable delivery? Or persistent as in 'always there' when an event service starts?*

Topic Management

Crucial to any MOM system is who can create and delete topics. One workable model is to only allow components **with the proper permissions** to programmatically create new persistent topics, and to allow any process to create temporary topics. A more realistic model **tracks the resources associated with a CCA task or computation, and gives each of these tasks their own set of resources (including queues, etc.)**.
Of course, the system must come with tools that manage topics, queues and the event system in general. These **tools with allow management of the topic space, logging and monitoring of the event system and management of topic queues**.

Comments*:*
Permissions*: What do you have in mind regarding permissions? How will these get defined/set/enforced (a security service?)*
Track resources associated with CCA task etc: *Not sure we understand what you have in mind here?*
Tools: *It's possible to have lightweight, memory-based event services that don't really need sophisticated tool support – e.g a typical CORBA event services. An event service*

can be configured extensively from a simple config file/descriptor. What queue management tasks do you have in mind?

Appropriate Use of Events

It is **vital to note that events are not a replacement for remote method calls or other methods of communication**. Defining the appropriate level of events, how often events are raised and processed is vital for events to be used successfully. The framework can only process events as best they can.

Comment: *Just some thoughts. Some applications are best designed as asynchronous, as this model has inherent advantages of loose coupling, reliability and sometimes increased concurrency. Might it be possible that once CCA has an event service, designers may decide to utilize it instead of synchronous RMI in appropriate circumstances? Obviously every service design/implementation has its tolerances, but can an implementation be as generally useful as possible to support application designers as best as possible? And can the spec support a range of implementations, some fast and simple, others less fast and scalable, others slower, scalable and smart?*