Distributed Runtime

A page to capture thoughts and notes about the Tuscany SCA distributed runtime. Feel free to help in developing the ideas here. I’ve put the source for the diagrams on this page in my sandbox here.

While the nature of a distributed runtime implies that more than one runtime of more than one type will be involved in a running SCA application, it seems sensible to work with distributing one type of runtime (java) before branching out.

Distributed SCA Runtimes - Notes

The assembly model specification deals briefly with distributed runtimes in its discussion of SCA Domains. 

"An SCA Domain represents a complete runtime configuration, potentially distributed over a series of interconnected runtime nodes."

The assembly spec, however, is not prescriptive about how an SCA Domain should be mapped and supported across multiple runtime nodes. Here I believe the term runtime node (or just node) is used to describe a process running an SCA runtime in which components can be run, e.g. the Java or C++ runtimes that Tuscany is developing.

Furthermore, the SCA specifications do not talk about the form of repository in which the metadata of the SCA Domain is held - although clearly the metadata must live somewhere. The Domain repository could take a variety of forms, from a simple shared set of directories to a sophisticated distributed database system. SCA runtimes should be able to interact with a variety of Domain repositories.

Motivation

Some use cases that provide motivation for distributing a domain:

1. Represent the widely distributed nature of a typical SOA so that SCA presents a cross enterprise description of assembled components
2. Support policy matching where components require particular resources and hence particular, and separate, nodes
3. HA/Load balancing/Performance scenarios where a single component appears on multiple nodes
4. Load balancing/Performance scenarios where domain is spread across multiple nodes (same as 1 & 2 I believe)
5. Dynamic wiring/Registry based service location, i.e. the SCA binding is called upon to automatically locate services based on registry entries. (overlaps with all of the above)

Terminology

See http://cwiki.apache.org/confluence/display/TUSCANYWIKI/Terminology

Scoping The Distribution Problem

There are many existing technologies that deal with managing compute nodes and job scheduling. So it’s probably safe to start by ignoring the issue of how the system picks processors on which runtime nodes will run. So the runtime management of the nodes themselves is out of scope.

There are also many technologies that provide scalable, robust and/or high performance service hosting solutions. So we can also ignore the issue of how component instances are actually constructed as the runtime representation of components deployed to a runtime. For example if a JVM clustering solution is chosen to implement a node then we assume that local method calls within that cluster will be handled by the clustering technology and no special action is required. If higher level clustering technology is in operation then integration with the runtime is required. In this case, where each node in the cluster runs part of the domain a component can be mapped to multiple nodes, the most natural integration point is the SCA binding which must interact with the clustering technology in order locate target component services.

So the initial area of consideration is how the components of a domain are associated with runtime nodes (2).

Cardinality
In the non-distributed case a single runtime node loads all contributions and runs all components.

In the distributed case, a Domain may span many nodes.

Each component must be associated (through specific configuration or some matching algorithm) with one or more nodes. If the same component appears in more than one node, the runtime is responsible for deciding how messages are routed to the correct node.

There is no restriction on the number of component instances a node can create, or indeed how these component instances are run. For example, all component instances could run in a single VM or be distributed across a number of VM’s to provide improved performance or failover for example.

Some questions have been raised about cardinality:

Should load balancing or HA type scenarios be able to be described in the topology by allowing components to be assigned to more than one node?

Answer: Yes. However, this may well be best handled by a "layered runtime" approach, where a whole cluster of nodes is presented to the rest of the distributed runtime as a single node.

Should a runtime be able to run more than one domain?

Answer: Yes. Multiple Domains can run on the same runtime. It is up to the runtime implementation to ensure that appropriate partitioning is achieved, since SCA Domains are intended to be isolated (for example, a reference in one domain cannot directly reference a service in another domain through it SCA component & service names).

**Scenario - Simple Distributed Components**
In this basic scenario a number of composites are started across nodes from the command line and once they are all started messages are send through the application

Demonstrates: the sca binding and service resolution within the domain.

**Scenario - Standalone Node**

Composites are added to the node through the node API and the node is started.

Demonstrates: Resolution of wires across composites with a single node

**Scenario - Nodes Connected To A domain**

Composites are added through the node API and each node is started

Demonstrates: Compilation of a domain view of the application as composites are started on nodes

**Scenario - Nodes Running in a Web App**

Nodes started in web apps run the composites from those applications and registers them with the domain

Demonstrated: Compilation of a domain view of the application a web apps are run

**Scenario - Virtual Node**

A node is associated with a domain that doesn’t have a tuscany runtime.

Demonstrates: Ability of Tuscany domain to include components/services that are not running on an SCA runtime.

**Scenario - Domain Adding Nodes**

A node is started and it becomes part of the domain ready to run composites

**Managing The Distributed Domain**
The logical view of how the different parts of the solution communicate is.

**Messages** - the application messages that flow between configured components. Messages will flow over bindings described explicitly in the assembly model or across the default binding used when no explicit binding is specified.

**Configuration** - In the distributed domain configuration is shared across the nodes with which the domain is associated. This includes information about, contributed resources, running components and their endpoints and domain configuration items such as base URLs.

**Events** - as the domain runs interesting events will occur, for example, a node fails and is restarted meaning that a set of endpoints change.

**Components Of The Solution**

Based on the calculator scenario we can take a general view of how the domain organizes running application
However there are a number of specific configurations to consider which affect the way that configuration and events are distributed.

### Domain Driven

- Domain
  - Registration
    - Add/RemoveContribution
    - Add/RemoveComposite
  - start
  - stop

### Node Driven

- Node
  - DomainProxy
APIs

SCADomainFactory

SCADomain

- public void start() throws DomainException;
- public void stop() throws DomainException;
- public String getURI();
- public void addContribution(String uri, URL url) throws DomainException;
- public void removeContribution(String uri) throws DomainException;
- public void addToDomainLevelComposite(QName compositeQName) throws DomainException;
- public void removeFromDomainLevelComposite(QName compositeQName) throws DomainException;
- public void addDeploymentComposite(String composite, CompositeXML) throws DomainException;
- public void startComposite(QName qname) throws DomainException;
- public void stopComposite(QName qname) throws DomainException;
- public <B, R extends CallableReference<B>> R cast(B target) throws IllegalArgumentException;
- public <B> B getService(Class<B> businessInterface, String serviceName);
- public <B> ServiceReference<B> getServiceReference(Class<B> businessInterface, String referenceName);

SCANodeFactory

SCANode

- public String getURI();
- public SCADomain getDomain();
- public void addContribution(String uri, URL url) throws DomainException;
- public void removeContribution(String uri) throws DomainException;
- public void addToDomainLevelComposite(QName compositeQName) throws DomainException;
- public void removeFromDomainLevelComposite(QName compositeQName) throws DomainException;
- public void startComposite(QName composite) throws NodeException;
- public void stopComposite(QName composite) throws NodeException;
- public void start() throws NodeException;
- public void stop() throws NodeException;
- public void destroy() throws NodeException;

SPIs

NodeEvents (node to domain)

- public String registerNode(String nodeURI, String nodeURL);
- public String removeNode(String nodeURI);
- public void registerContribution(String nodeURI, String contributionURI, String contributionURL);
- public void unregisterContribution(String contributionURI);
- public String registerServiceEndpoint(String domainUri, String nodeUri, String serviceName, String bindingName, String URI);
- public String removeServiceEndpoint(String domainUri, String nodeUri, String serviceName, String bindingName);
- public String findServiceEndpoint(String domainUri, String serviceName, String bindingName);

NodeManagement (domain to node)
public String getURI();
public void addContribution(String contributionURI, String contributionURL);
public void deployComposite(String compositeName);
public void start();
public void stop();

## Interactions

<table>
<thead>
<tr>
<th>Action</th>
<th>Domain</th>
<th>Domain Proxy</th>
<th>Node</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Starting Node standalone</td>
<td>SCANodeFactory nodeFactory = SCANodeFactory.newInstance();</td>
<td></td>
<td></td>
<td>use default node URL and don't connect to a domain</td>
</tr>
<tr>
<td>Starting Domain</td>
<td>SCADomainFactory domainFactory = SCADomainFactory.newInstance();</td>
<td></td>
<td></td>
<td>use default domain URL on this machine</td>
</tr>
<tr>
<td>Starting Node to connect to domain</td>
<td>SCADomain node = domainFactory.createSCADomain (null)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Starting Domain proxy standalone</td>
<td>SCANode node = nodeFactory.createSCANode (null, null);</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Add contribution to domain</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Add contribution to node</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Remove Contribution from domain</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Remove Contribution from node</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Update Contribution in domain</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Update Contribution in node</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Start domain</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Start node</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Start node</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Start node</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Stop node</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Stop node</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Stop node</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Start composite at domain</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Start composite at domain</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Start composite at domain</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Get service from domain</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Get service from domain proxy</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Get service from node</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

## Load Balancing

## Reliability and Failover