Distributed Calculator

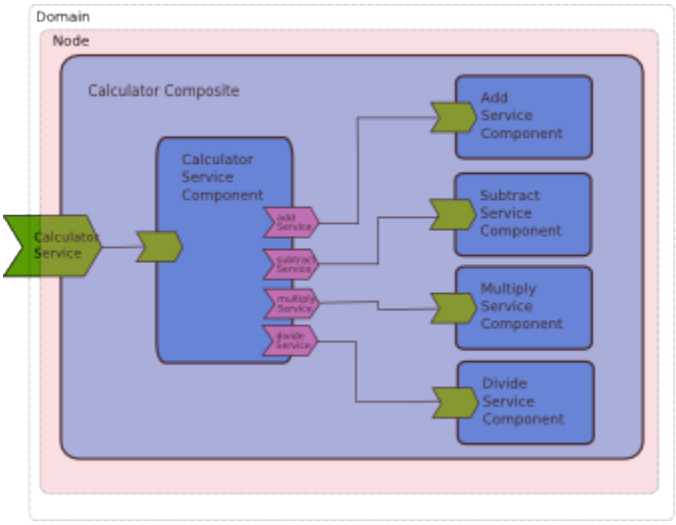
Create A Distributed Application

An SCA assembly can be run in a single or multi-node environment. Here we introduce the Tuscany node and SCA domain and explain how the calculator example can be distributed across multiple nodes.

Nodes

Node is a Tuscany specific term (you won't find it in the OASIS SCA specifications). The node wraps the Tuscany SCA runtime. The runtime comprises the modules that are present in the modules directory of the Tuscany SCA Java distribution. As you might expect there are functions that read XML, create an in memory model of the SCA assembly, create the components and wire them together ready to process incoming messages.

A node runs an SCA assembly described by one or more composite files. A node runs in a single JVM. You can start more than one node in a single JVM if you need to but we only tend to do this for testing purposes. In the first section of the user guide we built a calculator sample that included Calculator, Add, Subtract, Multiply and Divide components included in a single composite.



The composite was run using the following API;

```
        node = SCANodeFactory.newInstance().createSCANodeFromClassLoader("Calculator.composite", getClass().
getClassLoader());
        node.start();
```

This loads the named composite, and any associated resources, and then starts the node to start up all of the components within the composite.

Configuring Nodes

Creating a node in code is straightforward. For example,

```
SCANode2 node = SCANodeFactory.newInstance().createSCANodeFromClassLoader("calculator.composite", null);
```

Here the "null" parameter means that the node factory uses the current class' classloader to locate the named composite file. The location of the composite file is assumed to be the location of the SCA contribution. The assumption here is that only one contribution is required.

If more contributions must be loaded by the node the following interface can be used.

```
SCANode2  node = SCANodeFactory.newInstance().createSCANode("file:/C:/CalculatorContribution1/Calculator.
composite",
                                                    new SCAContribution("CalculatorContribution1",
                                                                        "file:/C:
/CalculatorContribution1"),
                                                    new SCAContribution("CalculatorContribution2",
                                                                        "file:/C:
/CalculatorContribution2"));
```

Where

```
"file:/C:/CalculatorContribution1/Calculator.composite"
```

Is the explicit location of the composite file to be run and

```
new SCAContribution("CalculatorContribution1",
                    "file:/C:/CalculatorContribution1")
```

Shows how to provide contribution details. The first parameter "CalculatorContribution1" gives a name for the contribution. The second parameter tells the node where to find the contribution. As this second parameter is a URL the contribution can be read from disk or directly from the network.

Using multiple contributions can be useful if, for example, you have chosen to separate common artifacts from those specific to this composite. The contribution containing common artifacts can then be used in other SCA applications without change.

Starting a Node

Once the node is created it is configured and ready to be started. It can be started as follows.

```
node.start();
```

Locating Services

If the node has been created from within a piece of Java code using the node fatory interface a local service reference can be retrieved in the following way.

```
calculatorService = ((SCAClient)node).getService(CalculatorService.class, "CalculatorServiceComponent");
```

One a node has been started all of its remote bindings will be active so you can also access the component services that the node contains via any remote bindings that they define.

Stopping a Node

If you are done with the node or need to stop it processing messages use the following.

```
node.stop();
```

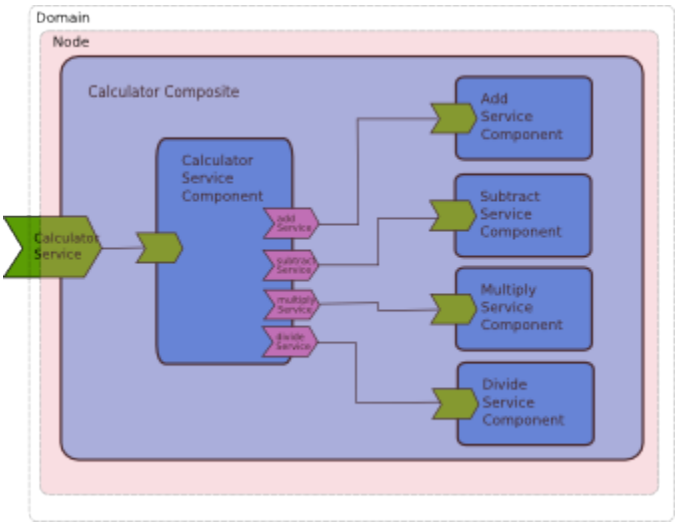
SCA Domain

When running standalone a node defines the scope of component services that references can target by name. SCA defines the term SCA Domain to describe this scope. Section 10 of the SCA Assembly specification actually describes an SCA Domain as defining "the boundary of visibility for all SCA mechanisms".

In a standalone node a reference can only target services that are running within the same node. However nodes running in separate JVMs can be configured to run an SCA assembly in a distributed fashion. In this way components running on one node can reference components running on a separate node. To make this configuration work the nodes must be running as part of the same distributed SCA domain. In Tuscany a distributed SCA domain is represented using a separate domain manager process.

Creating composites to run across multiple nodes.

From the calculator sample you can see that the wires between the component references and services, formed by adding a target component name to a reference, are resolved inside an SCA domain comprising a single node.



For example, this section of the calculator composite file shows how the CalcuatorServiceComponent targets the AddServiceComponent.

```
<composite xmlns="http://www.osoa.org/xmlns/sca/1.0"
           targetNamespace="http://sample"
           xmlns:sample="http://sample"
           name="Calculator">
    <component name="CalculatorServiceComponent">
        <implementation.java class="calculator.CalculatorServiceImpl"/>
        <reference name="addService" target="AddServiceComponent" />
        <reference name="subtractService" target="SubtractServiceComponent" />
        <reference name="multiplyService" target="MultiplyServiceComponent" />
        <reference name="divideService" target="DivideServiceComponent" />
    </component>

    <component name="AddServiceComponent">
        <implementation.java class="calculator.AddServiceImpl"/>
    </component>

    ...
</composite>
```

The target="AddServiceComponent" of the CalculatorServiceComponent's addService reference refers to the AddServiceComponent defined later on in this composite. A single composite must run on a single node so to distribute this application we need to separate it into more than one composite.

In the distributed calculator sample the composite defines the calculator component but not the add component;

```
<composite xmlns="http://www.osoa.org/xmlns/sca/1.0"
           targetNamespace="http://sample"
           xmlns:sample="http://sample"
           name="Calculator">
    <component name="CalculatorServiceComponent">
        <implementation.java class="calculator.CalculatorServiceImpl"/>
        <reference name="addService" target="AddServiceComponent" />
        <reference name="subtractService" target="SubtractServiceComponent" />
        <reference name="multiplyService" target="MultiplyServiceComponent" />
        <reference name="divideService" target="DivideServiceComponent" />
    </component>

    ...
</composite>
```

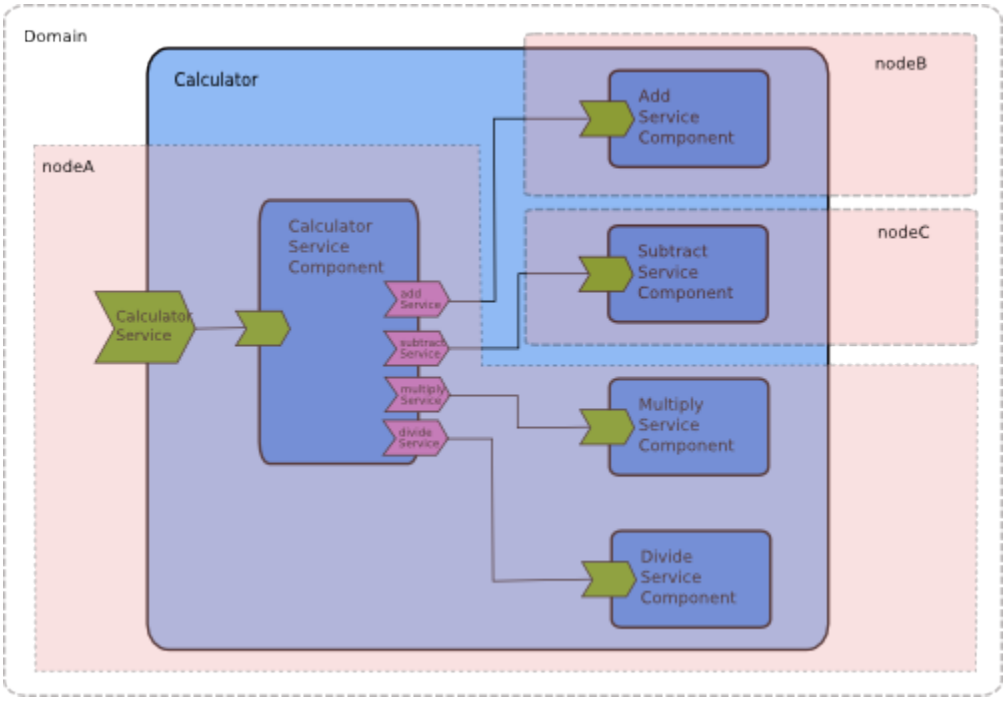
Another composite defines the add component.

```
<composite xmlns="http://www.osoa.org/xmlns/sca/1.0"
           targetNamespace="http://sample"
           xmlns:sample="http://sample"
           name="Add">

    <component name="AddServiceComponent">
        <implementation.java class="calculator.AddServiceImpl"/>
    </component>

    ...
</composite>
```

These two composites are run by separate nodes within the context of a single domain and the calculator addService reference

```
<reference name="addService" target="AddServiceComponent" />
```

is still be able to find the add component running on a separate node.



So a distributed SCA domain can consist of more than one node/composite and wires can run between components defined in the separate composites.

Starting A Domain

There is a launcher that has a mainline for starting the domain. For example,

```
public static void main(String[] args) throws Exception {
    DomainManagerLauncher.main(args);
}
```

In the calculator-distributed sample this code is contained in the class node.LaunchDomain.java. Runing this will run the domain mananger.

This article that is hosted outside of Tuscany provides a step by step instruction for how to use the Tuscany web-based domain manager UI to administer an SCA domain and how to use the domain administrative UI to deploy an SOA solution comprised of SCA components.

Configuring A Domain Through The Domain Manager Web Interface

Once the domain manager is running you can get to the web interface of the default domain by pointing your browser at http://localhost:9990/ui. This provides a number of different pages allowing you to

- Add contributions to the domain
- Add node configurations to the domain
- Configure composites to run on specific nodes
- Start local node instances to test the configuration

When you start the domain manager for the distributed calculator sample you will see the pages already populated with the information needed to execture the sample.

Configuring a Domain Through The File System

Behind the domain manager web application there are three files where the configuration information is stored.

workspace.xml - stores the ID and location of each contribution that has been added to the domain.

domain.composite - the virtual domain composite. This is an SCA composite that represents the virtual domain which includes all of the composites that will run in the domain. A URI is provided which indicates which contribution each composite comes from. Ultimately an SCANode instance will be started for each composite included in the virtual domain. Each SCANode instance can be running on separate/distributed processors. So the virtual domain is a consolidated description of you distributed SCA application

cloud.composite - describes the compute cloud (the set of distributed SCANode instances) being used to run the SCA application. Each node is assigned a composite and also has details of the configuration of bindings when they run on that node, for example, the root URI is set. It's possibly slightly confusing that this file is in SCA composite format also but this is a convenience because firstly we didn't have to create a new file format and secondly there may be some benefit in the future of representing this network of nodes as a network of SCA services for management purposes although we don't do that yet.

These files are updated when you make changes to the domain through the domain manager interface but they have a fairly simple XML format and can be edited manually or new utilities could be constructed to edit them automatically. In the distributed calculator sample you can find these files in the root directory.

Connection to non-SCA services

To connect to services outside of the SCA Domain (whether they be services provided by SCA or by other means) you can still configure an explicit binding, for example, lets now assume that the DivideServiceComponent is a non-sca web service out there on the network somewhere. As this is outside the SCA domain we can use an explicit remote binding to talk to it.

```
<component name="CalculatorServiceComponent">
                <implementation.java class="calculator.CalculatorServiceImpl"/>
        <reference name="addService" ></reference>
        <reference name="subtractService" target="SubtractServiceComponent"></reference>
        <reference name="multiplyService" target="MultiplyServiceComponent"></reference>
        <reference name="divideService" target="DivideServiceComponent">
            <binding.ws uri="http://localhost:8080/sample-calculator-ws-webapp/AddServiceComponent"/>
        </reference>
    </component>
```

Starting Nodes In A Distributed Domain

The domain manager is used to pre-process each composited prior to the composites being deployed to nodes for running. Contributions (containing the composites) are added to the domain manager. Configuration must be provided which tells the domain manager where the nodes are that are going to run the composites and which composites they will run. Based on this information the domain manager makes an atom feed available for each composite that is deployed to a node. The node to which the composite is assigned must read the correct atom feed in order to configure itself by downloading the composite to run and the contributions required provide the artifacts required by the composite. There is a Node API that takes as input the URL of an atom feed exposed by the domain manager as follows.

```
SCANode2 node = SCANodeFactory.newInstance().createSCANodeFromURL("http://localhost:9990/node-config/NodeA");
```

The URL http://localhost:9990/node-config/NodeA is the URL of the atom feed that the domain manager provides.

The domain manager processes all the relationships between all composite components and provides a fully configured composite to each node, i.e. all of the binding URIs are filled out correctly so it's basically a pre-processing step.

In the calculator-distributed sample java classes are provided which are set up ready to start the required nodes. Look at the files.

```
node.LaunchCalculatorNodeA
node.LaunchCalculatorNodeB
node.LaunchCalculatorNodeC
```

The README provided with the samples gives more detailed instruction on how to run the sample.