BAE SYSTEMS
INSPIRED WORK

Manager Module Example

Introduction

This document describes an ECOA® client-server example named "*Manager Module Example*".

The client-server model (ref. [2]) is one of the most basic data, task, or workload, distribution mechanisms in computing. Clients and servers may be distributed across a network, or they may reside on the same computing system. Service oriented concepts, which form a basis behind the ECOA, naturally fit with the client-server model, the clients referencing (using) the services provided by the server. Service orientation, and therefore the ECOA, goes on a step extra, in that a component can be a client (service user) to one or more other components, whilst simultaneously being a server (service provider) to others.

This document presents the principal user generated artefacts required to create the "*Manager Module Example*" client-server example using the ECOA. It is assumed that the reader is conversant with the ECOA Architecture Specification (ref. [1]) and the process of defining and declaring ECOA Assemblies, ASCs (components), Modules, and deployments in XML, and then using code generation to produce Module framework (stub) code units and ECOA Container and Platform code.

Aims

This ECOA "*Manager Module Example*" client-server example is based upon the "*Service Availability Example*" (ref. [3]), and is intended to demonstrate one design patterns which can be used by an ECOA system designer in order to implement a functional manager module within a Component.

ECOA Features Exhibited

- Composition of an ECOA Assembly of multiple ECOA ASCs (components).
- Contention-free resource sharing within an ECOA Assembly.
- Use of the ECOA runtime logging API.
- Management of services using a "functional availability"
- Management of multi-module Component using a "manager module"

Design and Definition

Client-Server Functional Design

The "*Manager Module Example*" client-server example will demonstrate the use of a "manager" module which controls the behaviour of a Component. This includes the functional startup of the Component and the service availability management. The example uses 2 services, each containing a

number of request-response operations to perform simple mathematic functions. Figure 1 shows the behaviour of the example system.

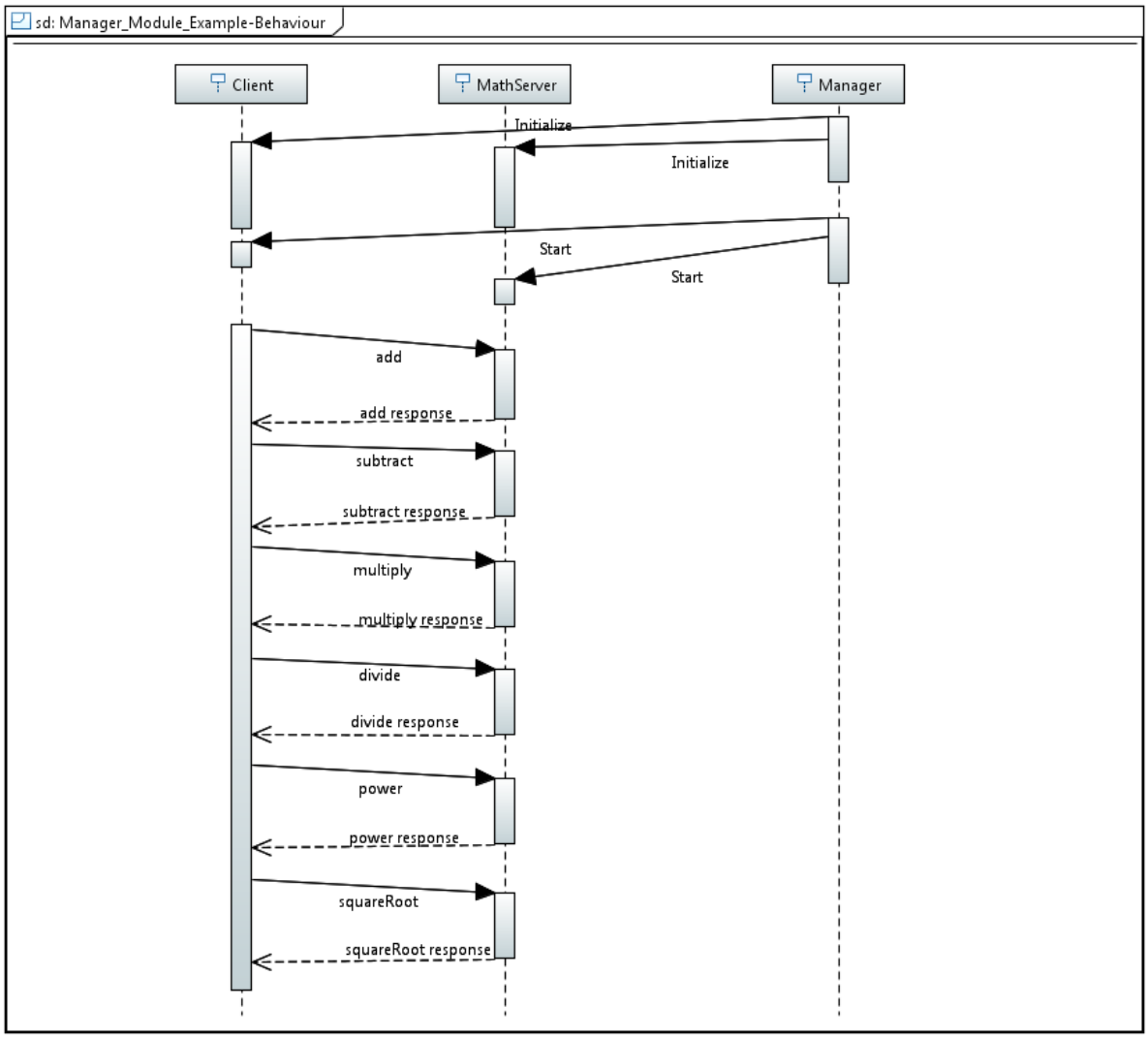

Figure 1 - ECOA "Manager Module Example" Client-Server Behaviour

The Client will perform a number of request operations in order to perform simple mathematic operations:

1. Addition
2. Subtraction
3. Multiplication
4. Division
5. Power

6. Square Root

The Server will be split into 3 modules, a "Manager" module which manages the behaviour of both the "Basic Math" and the "Complex Math" modules.

ECOA Assembly Design and Definition

This ECOA "*Manager Module Example*" client-server example ECOA Assembly comprises two ECOA ASCs named "*Client*" and "*MathServer*". The "*Client*" ASC type is instantiated once within the ECOA Assembly as "*Client_Inst*". The "*Server*" ASC is instantiated once within the ECOA Assembly as "Math*Server_Inst*" and provides the "*ProvidedBasicMath*" and "*ProvidedComplextMath*" ECOA Services, both of which are referenced (used) by the "*Client_Inst*" ASC (Figure 2).

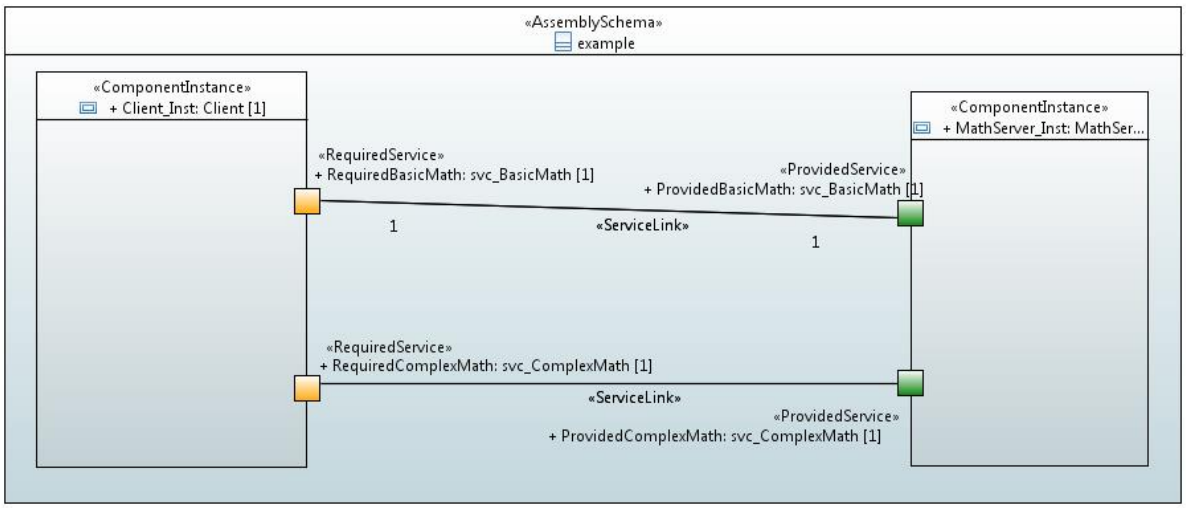

Figure 2 - ECOA "Manager Module" Assembly Diagram

This ECOA Assembly is defined in an Initial Assembly XML file, and declared in a Final Assembly (or Implementation) XML file (which is practically identical). The Final Assembly XML for the ECOA "*Manager Module Example*" Assembly is as follows (file *example.impl.composite*):

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<csa:composite
   xmlns:csa="http://docs.oasis-open.org/ns/opencsa/sca/200912"
   xmlns:ecoa-sca="http://www.ecoa.technology/sca-extension-2.0"
   name="example"
   targetNamespace="http://www.ecoa.technology">


   <csa:component name="Client_Inst">
      <ecoa-sca:instance componentType="Client">
         <ecoa-sca:implementation name="Client_Im"/>
      </ecoa-sca:instance>
```

```
      <csa:reference name="RequiredBasicMath">
         <ecoa-sca:interface syntax="svc_BasicMath"/>
      </csa:reference>

      <csa:reference name="RequiredComplexMath">
         <ecoa-sca:interface syntax="svc_ComplexMath"/>
      </csa:reference>

   </csa:component>

   <csa:component name="MathServer_Inst">
      <ecoa-sca:instance componentType="MathServer">
         <ecoa-sca:implementation name="MathServer_Im"/>
      </ecoa-sca:instance>

      <csa:service name="ProvidedBasicMath">
         <ecoa-sca:interface syntax="svc_BasicMath"/>
      </csa:service>

      <csa:service name="ProvidedComplexMath">
         <ecoa-sca:interface syntax="svc_ComplexMath"/>
      </csa:service>

   </csa:component>

   <csa:wire source="Client_Inst/RequiredBasicMath"
target="MathServer_Inst/ProvidedBasicMath"/>

   <csa:wire source="Client_Inst/RequiredComplexMath"
target="MathServer_Inst/ProvidedComplexMath"/>

</csa:composite>
```

The *MathServer* ASC type is defined in XML as follows (file *MathServer.componentType*):

```
<?xml version="1.0" encoding="UTF-8"?>
<componentType xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200912"
xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:ecoa-sca="http://www.ecoa.technology/sca-extension-2.0">

   <service name="ProvidedBasicMath">
      <ecoa-sca:interface syntax="svc_BasicMath"/>
   </service>

   <service name="ProvidedComplexMath">
      <ecoa-sca:interface syntax="svc_ComplexMath"/>
   </service>

</componentType>
```

The ASC definition (the *<componentType>* XML element) declares the provision (by the ASC) of the *ProvidedBasicMath* and *ProvidedComplexMath* ECOA Services.

The *Client* ASC type is defined in XML as follows (file *Client.componentType*):

```
<?xml version="1.0" encoding="UTF-8"?>
<componentType xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200912"
xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:ecoa-sca="http://www.ecoa.technology/sca-extension-2.0">

   <reference name="RequiredBasicMath">
      <ecoa-sca:interface syntax="svc_BasicMath"/>
   </reference>

   <reference name="RequiredComplexMath">
      <ecoa-sca:interface syntax="svc_ComplexMath"/>
   </reference>

</componentType>
```

This ASC definition (the *<componentType>* XML element) declares a reference (by the ASC) to the *RequiredBasicMath* and *RequiredComplexMath* ECOA Services.

ECOA Service and Types Definition

The *svc_BasicMath* Service, which is provided by the *MathServer* ASC and referenced by the *Client* ASC, is defined in a XML file (*svc_BasicMath.interface.xml*):

```
<?xml version="1.0" encoding="UTF-8"?>
<serviceDefinition xmlns="http://www.ecoa.technology/interface-2.0">

   <use library="BasicMath"/>

   <operations>
      <requestresponse name="add">
         <input name="value1" type="ECOA:int32"/>
         <input name="value2" type="ECOA:int32"/>
         <output name="result" type="ECOA:int32"/>
      </requestresponse>

      <requestresponse name="subtract">
         <input name="value1" type="ECOA:int32"/>
         <input name="value2" type="ECOA:int32"/>
         <output name="result" type="ECOA:int32"/>
      </requestresponse>

      <requestresponse name="multiply">
         <input name="value1" type="ECOA:int32"/>
         <input name="value2" type="ECOA:int32"/>
         <output name="result" type="ECOA:int32"/>
      </requestresponse>

      <requestresponse name="divide">
         <input name="value1" type="ECOA:int32"/>
         <input name="value2" type="ECOA:int32"/>
         <output name="result" type="ECOA:int32"/>
```

```
         <output name="status" type="BasicMath:Divide_Status_Type"/>
      </requestresponse>

      <data name="available" type="ECOA:boolean8"/>

   </operations>
</serviceDefinition>
```

The Service comprises four ECOA Request-Response Operations called *add, subtract, multiply and divide*. In addition, an ECOA Versioned Data Operation called *available* is defined.

The *svc_ComplexMath* Service, which is provided by the *MathServer* ASC and referenced by the *Client* ASC, is defined in a XML file (*svc_ComplexMath.interface.xml*):

```
<?xml version="1.0" encoding="UTF-8"?>
<serviceDefinition xmlns="http://www.ecoa.technology/interface-2.0">


   <operations>
      <requestresponse name="power">
         <input name="base" type="ECOA:int32"/>
         <input name="exponent" type="ECOA:int32"/>
         <output name="result" type="ECOA:int32"/>
      </requestresponse>

      <requestresponse name="squareRoot">
         <input name="value" type="ECOA:int32"/>
         <output name="result" type="ECOA:int32"/>
      </requestresponse>

   </operations>
</serviceDefinition>
```

The Service comprises two ECOA Request-Response Operations called *power* and *squareRoot*.

The data types library (used in the *svc_BasicMath*) is, unsurprisingly, also defined in an XML (file *BasicMath.types.xml*):

```
<?xml version="1.0" encoding="UTF-8"?>
<library xmlns="http://www.ecoa.technology/types-2.0">


   <types>
      <enum name="Divide_Status_Type" type="ECOA:uint32">
         <value name="OK" valnum="0"/>
         <value name="Error" valnum="1"/>
         <value name="Divide_By_Zero" valnum="2"/>
         <value name="Unavailable" valnum="3"/>
      </enum>

   </types>
```

```
</library>
```

The data type *BasicMath:Divide_Status_Type* is therefore an enumeration type, with 4 possible values.

ECOA Module Design and Definition

The *MathServer* ASC (component) is composed of three Modules (Module Implementations *Manager_Im*, *BasicMath_Im* and *ComplexMath_Im* of Module Types *BasicMath_Type*, *BasicMath_Type* and *ComplexMath_Type* respectively) as illustrated in UML in Figure 3.

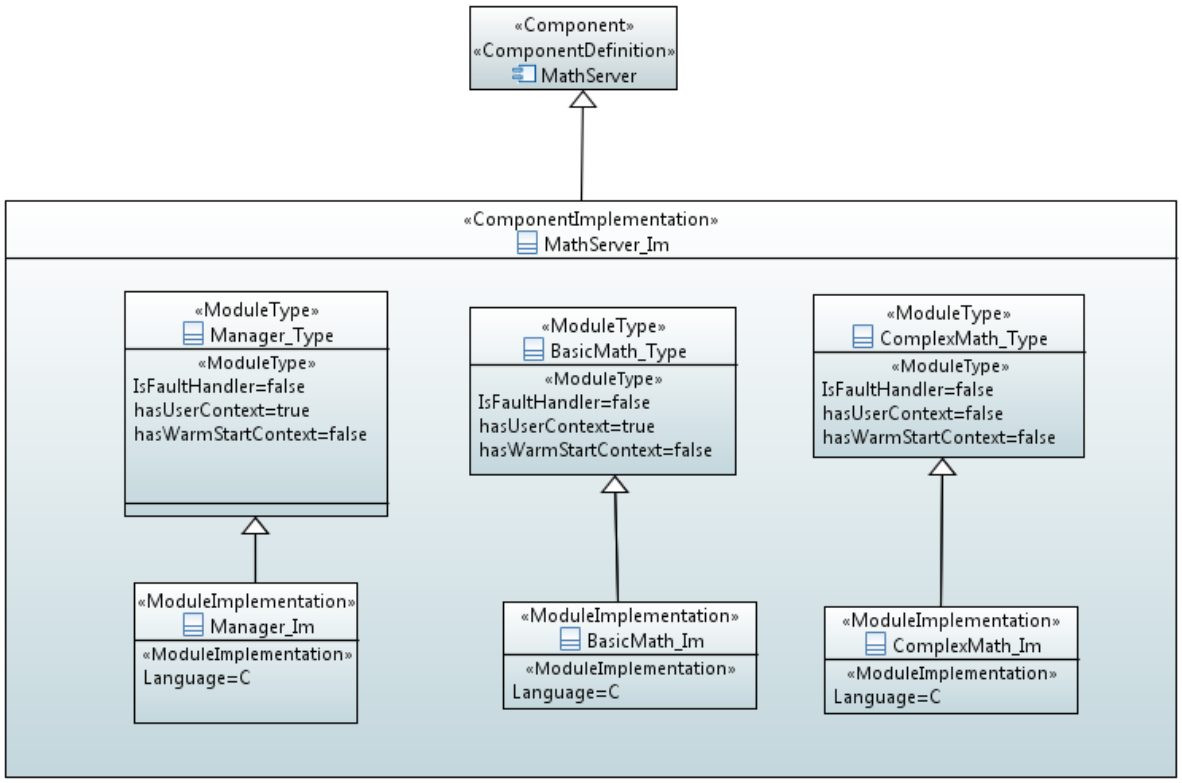

Figure 3 "MathServer" Module Design (as UML Composite Structure Diagram)

The *Client* ASC (component) is composed of a single ECOA Module (Module Implementations *Client_Module_Im* of Module Type *Client_Module_Type*) as illustrated in UML in Figure 4.

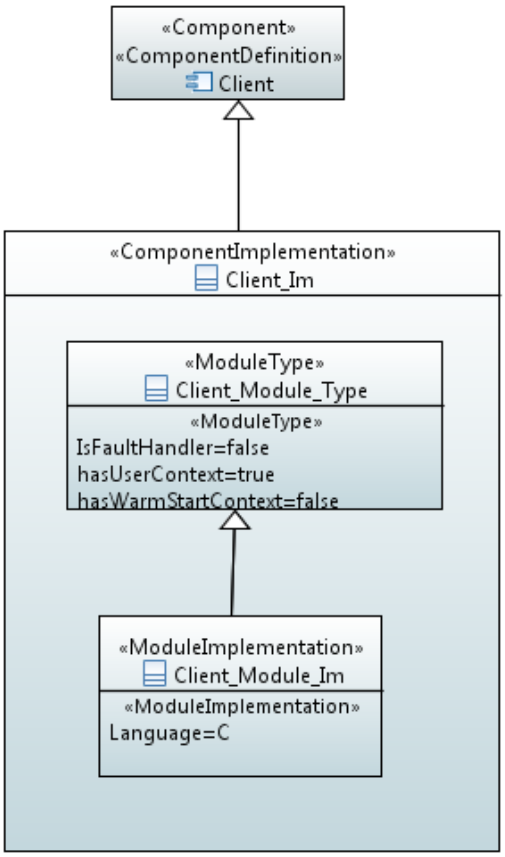

Figure 4 – "Client" Module Design (as UML Composite Structure Diagram)

Figure 5 and Figure 6 depict in UML the internal design of the `MathServer` ASC (component) ***providing*** the `svc_BasicMath` and `svc_ComplexMath` ECOA Services, whilst the `Client` ASC ***references*** the Services. As always in the ECOA, the Module Implementations implement the Module Lifecycle operations defined by the ECOA.

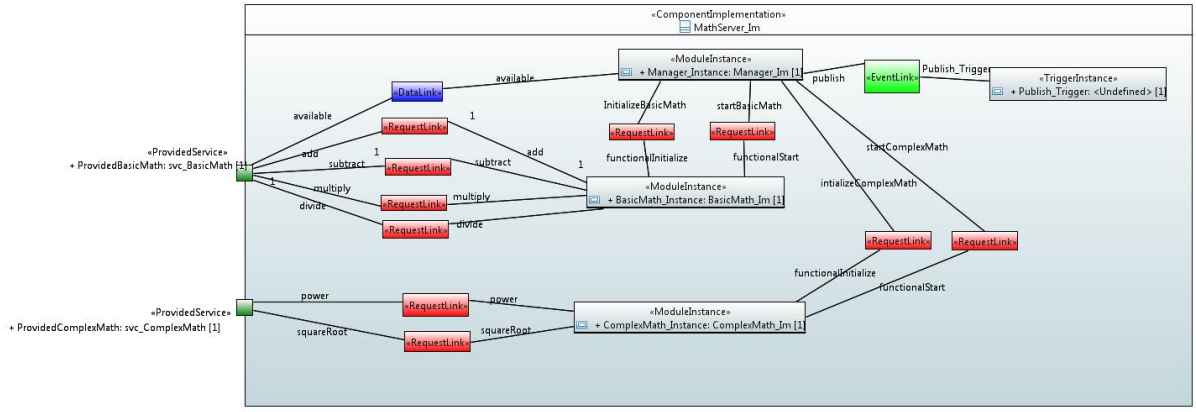

Figure 5 - "MathServer" Component Design (as UML Composite Structure Diagram)

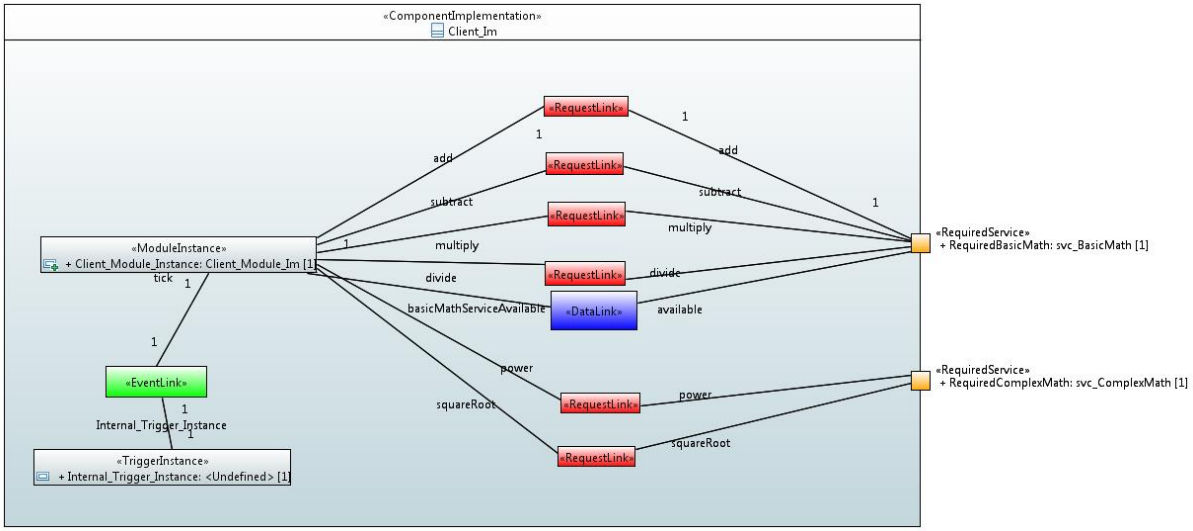

Figure 6 - "Client" Component Design (as UML Composite Structure Diagram)

The MathServer ASC

The *MathServer* ASC is declared in XML as follows (file *MathServer_Im.impl.xml*):

```
<?xml version="1.0" encoding="UTF-8"?>
<componentImplementation xmlns="http://www.ecoa.technology/implementation-2.0"
   componentDefinition="MathServer">

   <use library="BasicMath"/>

   <moduleType name="BasicMath_Type" hasUserContext="true"
hasWarmStartContext="false">
```

```
      <operations>

         <requestReceived name="add" maxConcurrentRequests="10">
            <input name="value1" type="ECOA:int32"/>
            <input name="value2" type="ECOA:int32"/>
            <output name="result" type="ECOA:int32"/>
         </requestReceived>

         <requestReceived name="subtract" maxConcurrentRequests="10">
            <input name="value1" type="ECOA:int32"/>
            <input name="value2" type="ECOA:int32"/>
            <output name="result" type="ECOA:int32"/>
         </requestReceived>

         <requestReceived name="multiply" maxConcurrentRequests="10">
            <input name="value1" type="ECOA:int32"/>
            <input name="value2" type="ECOA:int32"/>
            <output name="result" type="ECOA:int32"/>
         </requestReceived>

         <requestReceived name="divide" maxConcurrentRequests="10">
            <input name="value1" type="ECOA:int32"/>
            <input name="value2" type="ECOA:int32"/>
            <output name="result" type="ECOA:int32"/>
            <output name="status" type="BasicMath:Divide_Status_Type"/>
         </requestReceived>

         <requestReceived name="functionalInitialize" maxConcurrentRequests="1">
         </requestReceived>

         <requestReceived name="functionalStart" maxConcurrentRequests="1">
         </requestReceived>

         <dataRead name="available" type="ECOA:boolean8" notifying="true"/>

      </operations>

   </moduleType>

   <moduleType name="ComplexMath_Type" hasUserContext="false"
hasWarmStartContext="false">


      <operations>

         <requestReceived name="power" maxConcurrentRequests="10">
            <input name="base" type="ECOA:int32"/>
            <input name="exponent" type="ECOA:int32"/>
            <output name="result" type="ECOA:int32"/>
         </requestReceived>

         <requestReceived name="squareRoot" maxConcurrentRequests="10">
            <input name="value" type="ECOA:int32"/>
            <output name="result" type="ECOA:int32"/>
```

```
         </requestReceived>

         <requestReceived name="functionalInitialize" maxConcurrentRequests="1">
         </requestReceived>

         <requestReceived name="functionalStart" maxConcurrentRequests="1">
         </requestReceived>

      </operations>

   </moduleType>

   <moduleType name="Manager_Type" hasUserContext="true"
hasWarmStartContext="false">


      <operations>

         <dataWritten name="available" type="ECOA:boolean8"/>

         <eventReceived name="publish">
         </eventReceived>

         <requestSent name="initializeBasicMath" isSynchronous="false" timeout="-
1.0" maxConcurrentRequests="1">
         </requestSent>

         <requestSent name="initializeComplexMath" isSynchronous="false"
timeout="-1.0" maxConcurrentRequests="1">
         </requestSent>

         <requestSent name="startBasicMath" isSynchronous="false" timeout="-1.0"
maxConcurrentRequests="1">
         </requestSent>

         <requestSent name="startComplexMath" isSynchronous="false" timeout="-1.0"
maxConcurrentRequests="1">
         </requestSent>

      </operations>

   </moduleType>


   <moduleImplementation name="BasicMath_Im" language="C"
moduleType="BasicMath_Type"/>
   <moduleImplementation name="ComplexMath_Im" language="C"
moduleType="ComplexMath_Type"/>
   <moduleImplementation name="Manager_Im" language="C"
moduleType="Manager_Type"/>

   <moduleInstance name="BasicMath_Instance" implementationName="BasicMath_Im"
relativePriority="2">
```

```
   </moduleInstance>
   <moduleInstance name="ComplexMath_Instance" implementationName="ComplexMath_Im"
relativePriority="3">

   </moduleInstance>
   <moduleInstance name="Manager_Instance" implementationName="Manager_Im"
relativePriority="1">

   </moduleInstance>

   <triggerInstance name="Publish_Trigger" relativePriority="2"/>


   <requestLink>

      <clients>
         <service instanceName="ProvidedBasicMath" operationName="add"/>
      </clients>
      <server>
         <moduleInstance instanceName="BasicMath_Instance" operationName="add"/>
      </server>
   </requestLink>

   <requestLink>

      <clients>
         <service instanceName="ProvidedBasicMath" operationName="subtract"/>
      </clients>
      <server>
         <moduleInstance instanceName="BasicMath_Instance"
operationName="subtract"/>
      </server>
   </requestLink>

   <dataLink>
      <writers>
         <moduleInstance instanceName="Manager_Instance"
operationName="available"/>
      </writers>
      <readers>
         <service instanceName="ProvidedBasicMath" operationName="available"/>
         <moduleInstance instanceName="BasicMath_Instance"
operationName="available"/>
      </readers>
   </dataLink>

   <requestLink>

      <clients>
         <service instanceName="ProvidedBasicMath" operationName="multiply"/>
      </clients>
      <server>
         <moduleInstance instanceName="BasicMath_Instance"
operationName="multiply"/>
```

```
      </server>
   </requestLink>

   <requestLink>

      <clients>
         <service instanceName="ProvidedBasicMath" operationName="divide"/>
      </clients>
      <server>
         <moduleInstance instanceName="BasicMath_Instance"
operationName="divide"/>
      </server>
   </requestLink>

   <requestLink>

      <clients>
         <service instanceName="ProvidedComplexMath" operationName="power"/>
      </clients>
      <server>
         <moduleInstance instanceName="ComplexMath_Instance"
operationName="power"/>
      </server>
   </requestLink>

   <requestLink>

      <clients>
         <service instanceName="ProvidedComplexMath" operationName="squareRoot"/>
      </clients>
      <server>
         <moduleInstance instanceName="ComplexMath_Instance"
operationName="squareRoot"/>
      </server>
   </requestLink>

   <eventLink>
      <senders>
         <trigger instanceName="Publish_Trigger" period="1"/>
      </senders>
      <receivers>
         <moduleInstance instanceName="Manager_Instance" operationName="publish"/>
      </receivers>
   </eventLink>

   <requestLink>

      <clients>
         <moduleInstance instanceName="Manager_Instance"
operationName="initializeBasicMath"/>
      </clients>
      <server>
         <moduleInstance instanceName="BasicMath_Instance"
operationName="functionalInitialize"/>
```

```
      </server>
   </requestLink>

   <requestLink>

      <clients>
         <moduleInstance instanceName="Manager_Instance"
operationName="startBasicMath"/>
      </clients>
      <server>
         <moduleInstance instanceName="BasicMath_Instance"
operationName="functionalStart"/>
      </server>
   </requestLink>

   <requestLink>

      <clients>
         <moduleInstance instanceName="Manager_Instance"
operationName="initializeComplexMath"/>
      </clients>
      <server>
         <moduleInstance instanceName="ComplexMath_Instance"
operationName="functionalInitialize"/>
      </server>
   </requestLink>

   <requestLink>

      <clients>
         <moduleInstance instanceName="Manager_Instance"
operationName="startComplexMath"/>
      </clients>
      <server>
         <moduleInstance instanceName="ComplexMath_Instance"
operationName="functionalStart"/>
      </server>
   </requestLink>


</componentImplementation>
```

That is, three Module Types (*Manager_Type*, *BasicMath_Type* and *ComplexMath_Type*) are declared.

The *Publish_Trigger* Trigger Instance is introduced because the Server needs to change its behaviour over time, and this trigger sequences the changes. Once every period (1 second as set in the *<eventLink>* XML) the Trigger will fire and the Module Operation *publish* will be invoked.

Manager_Type is a Module which has six operations specified:

- a *requestSent* operation "*initializeBasicMath*";

- a *requestSent* operation "*initializeComplexMath*";
- a *requestSent* operation "*startBasicMath*";
- a *requestSent* operation "*startComplexMath*";
- a *dataWritten* operation "*available*";
- the *eventReceived* operation "*publish*".

This Module Type is implemented by a concrete Module Implementation *Manager_Im* which in turn is instantiated once as the Module Instance *Manager_Instance*.

BasicMath_Type is a Module which has six operations specified:

- a *requestReceived* operation "*add*";
- a *requestReceived* operation "*subtract*";
- a *requestReceived* operation "*multiply*";
- a *requestReceived* operation "*divide*";
- a *requestReceived* operation "*functionalInitialize*";
- a *requestReceived* operation "*functionalStart*";
- a *dataRead* operation "*available*";

This Module Type is implemented by a concrete Module Implementation *BasicMath_Im* which in turn is instantiated once as the Module Instance *BasicMath_Instance*.

ComplexMath_Type is a Module which has two operations specified:

- a *requestReceived* operation "*power*";
- a *requestReceived* operation "*squareRoot*";
- a *requestReceived* operation "*functionalInitialize*";
- a *requestReceived* operation "*functionalStart*".

This Module Type is implemented by a concrete Module Implementation *ComplexMath_Im* which in turn is instantiated once as the Module Instance *ComplexMath_Instance*.

The operation links XML logically associates the specific concrete operations of the Module Instance with the abstract Service operations.

Three functional code units will be produced by the code generation process, implementing in code the concrete *Manager_Im*, *BasicMath_Im* and *ComplexMath_Im* classes, named "*Manager_Im.c*" "*BasicMath_Im.c*" and "*ComplexMath_Im.c*" respectively (assuming the Module Implementation declaration has set the *language* property to "C").

The Client ASC

The *Client* ASC is declared in XML as follows (file *Client_Im.impl.xml*):

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
<componentImplementation xmlns="http://www.ecoa.technology/implementation-2.0"
   componentDefinition="Client">

   <use library="BasicMath"/>

   <moduleType name="Client_Module_Type" hasUserContext="true"
hasWarmStartContext="false">


      <operations>

         <eventReceived name="tick">
         </eventReceived>

         <requestSent name="add" isSynchronous="true" timeout="-1.0"
maxConcurrentRequests="10">
            <input name="value1" type="ECOA:int32"/>
            <input name="value2" type="ECOA:int32"/>
            <output name="result" type="ECOA:int32"/>
         </requestSent>

         <requestSent name="subtract" isSynchronous="true" timeout="-1.0"
maxConcurrentRequests="10">
            <input name="value1" type="ECOA:int32"/>
            <input name="value2" type="ECOA:int32"/>
            <output name="result" type="ECOA:int32"/>
         </requestSent>

         <requestSent name="multiply" isSynchronous="true" timeout="-1.0"
maxConcurrentRequests="10">
            <input name="value1" type="ECOA:int32"/>
            <input name="value2" type="ECOA:int32"/>
            <output name="result" type="ECOA:int32"/>
         </requestSent>

         <requestSent name="divide" isSynchronous="true" timeout="-1.0"
maxConcurrentRequests="10">
            <input name="value1" type="ECOA:int32"/>
            <input name="value2" type="ECOA:int32"/>
            <output name="result" type="ECOA:int32"/>
            <output name="status" type="BasicMath:Divide_Status_Type"/>
         </requestSent>

         <requestSent name="power" isSynchronous="true" timeout="-1.0"
maxConcurrentRequests="10">
            <input name="base" type="ECOA:int32"/>
            <input name="exponent" type="ECOA:int32"/>
            <output name="result" type="ECOA:int32"/>
         </requestSent>

         <requestSent name="squareRoot" isSynchronous="true" timeout="-1.0"
maxConcurrentRequests="10">
            <input name="value" type="ECOA:int32"/>
            <output name="result" type="ECOA:int32"/>
```

```
         </requestSent>

         <dataRead name="BasicMathServiceAvailable" type="ECOA:boolean8"
notifying="true"/>

      </operations>

   </moduleType>


   <moduleImplementation name="Client_Module_Im" language="C"
moduleType="Client_Module_Type"/>

   <moduleInstance name="Client_Module_Instance"
implementationName="Client_Module_Im" relativePriority="1">

   </moduleInstance>

   <triggerInstance name="Internal_Trigger_Instance" relativePriority="2"/>


   <eventLink>
      <senders>
         <trigger instanceName="Internal_Trigger_Instance" period="2"/>
      </senders>
      <receivers>
         <moduleInstance instanceName="Client_Module_Instance"
operationName="tick"/>
      </receivers>
   </eventLink>

   <requestLink>

      <clients>
         <moduleInstance instanceName="Client_Module_Instance"
operationName="add"/>
      </clients>
      <server>
         <reference instanceName="RequiredBasicMath" operationName="add"/>
      </server>
   </requestLink>

   <requestLink>

      <clients>
         <moduleInstance instanceName="Client_Module_Instance"
operationName="subtract"/>
      </clients>
      <server>
         <reference instanceName="RequiredBasicMath" operationName="subtract"/>
      </server>
   </requestLink>

   <requestLink>
```

```
      <clients>
         <moduleInstance instanceName="Client_Module_Instance"
operationName="multiply"/>
      </clients>
      <server>
         <reference instanceName="RequiredBasicMath" operationName="multiply"/>
      </server>
   </requestLink>

   <requestLink>

      <clients>
         <moduleInstance instanceName="Client_Module_Instance"
operationName="divide"/>
      </clients>
      <server>
         <reference instanceName="RequiredBasicMath" operationName="divide"/>
      </server>
   </requestLink>

   <requestLink>

      <clients>
         <moduleInstance instanceName="Client_Module_Instance"
operationName="power"/>
      </clients>
      <server>
         <reference instanceName="RequiredComplexMath" operationName="power"/>
      </server>
   </requestLink>

   <requestLink>

      <clients>
         <moduleInstance instanceName="Client_Module_Instance"
operationName="squareRoot"/>
      </clients>
      <server>
         <reference instanceName="RequiredComplexMath"
operationName="squareRoot"/>
      </server>
   </requestLink>

   <dataLink>
      <writers>
         <reference instanceName="RequiredBasicMath" operationName="available"/>
      </writers>
      <readers>
         <moduleInstance instanceName="Client_Module_Instance"
operationName="BasicMathServiceAvailable"/>
      </readers>
   </dataLink>
```

```
</componentImplementation>
```

That is, a Module Type (*Client_Module_Type*) is declared which has eight operations:

- An "*add*" *requestSent* operation;
- An "*subtract*" *requestSent* operation;
- An "*multiply*" *requestSent* operation;
- An "*divide*" *requestSent* operation;
- An "*power*" *requestSent* operation;
- An "*squareRoot*" *requestSent* operation;
- A "*BasicMathServiceAvailable*" dataRead operation;
- The *eventReceived* operation "*tick*".

The *Internal_Trigger_Instance* Trigger Instance is introduced because the Client needs to "*periodically request mathematical calculations*" and so an ECOA periodic trigger is required. Once every period (2 seconds as set in the *<eventLink>* XML) the Trigger will fire and the Module Operation *tick* will be invoked.

This Module Type is implemented by a concrete Module Implementation *Client_Module_Im*, which in turn is instantiated once as the Module Instance *Client_Module_Instance*.

The operation links XML logically associates the specific concrete operations of the Module Instance with the abstract Service operations.

A single functional code unit will be produced by the code generation process, implementing in code the concrete *Client_Module_Im* class, and named "*Client_Module_Im.c*" (assuming the Module Implementation declaration has set the *language* property to "C").

ECOA Deployment Definition

The ECOA "*Manager Module Example*" Assembly is deployed (that is, the declared Module and Trigger Instances are allocated to a single ECOA Protection Domain, which is then allocated to a computing node) by the following XML (file *example.deployment.xml*):

```
<deployment xmlns="http://www.ecoa.technology/deployment-2.0"
finalAssembly="example" logicalSystem="example">

   <protectionDomain name="Ex1">
      <executeOn computingPlatform="Example_Platform" computingNode="card1_bae"/>

      <deployedModuleInstance componentName="Client_Inst"
moduleInstanceName="Client_Module_Instance" modulePriority="11"/>
      <deployedTriggerInstance componentName="Client_Inst"
triggerInstanceName="Internal_Trigger_Instance" triggerPriority="12"/>
      <deployedModuleInstance componentName="MathServer_Inst"
moduleInstanceName="ComplexMath_Instance" modulePriority="3"/>
```

```
      <deployedModuleInstance componentName="MathServer_Inst"
moduleInstanceName="BasicMath_Instance" modulePriority="3"/>
      <deployedTriggerInstance componentName="MathServer_Inst"
triggerInstanceName="Publish_Trigger" triggerPriority="12"/>
      <deployedModuleInstance componentName="MathServer_Inst"
moduleInstanceName="Manager_Instance" modulePriority="1"/>
   </protectionDomain>

   <platformConfiguration faultHandlerNotificationMaxNumber="8"
computingPlatform="Example_Platform"></platformConfiguration>

</deployment>
```

Thus in this case, a single ECOA Protection Domain is declared (*Ex1*) executing on an ECOA Computing Node, on a single ECOA Computing Platform.

Implementation

The MathServer ASC

The behaviour of each module of the *MathServer* ASC is described in detail in the following sections.

Manager_Im Module

The *Manager_Im* module is responsible for functionally initialising and starting the "functional" modules *BasicMath_Im* and *ComplexMath_Im*. In addition, the Manager Module is responsible for declaring the *ProvidedBasicMath* service as available when the Component is in the correct functional state to perform its (this is based upon the state of the *BasicMath_Instance* module being "functionally" running.

During initialisation time, the Manager module defaults state data and publishes the (unavailable) state of the *ProvidedBasicMath*. This functionality is implemented by the following (C) code:

```
void Manager_Im__INITIALIZE__received(Manager_Im__context *context)
{
   context->user.basicMathState = UNINITIALIZED;
   context->user.complexMathState = UNINITIALIZED;

   context->user.basicMathServiceAvailable = ECOA__FALSE;
   Publish_Functional_Service_Availability(context);
}
```

The function "Publish_Functional_Service_Availability" is defined as follows:

```
static void Publish_Functional_Service_Availability(Manager_Im__context* context)
{
   ECOA__return_status status;
   Manager_Im_container__available_handle basicMathAvailableHandle;

   status = Manager_Im_container__available__get_write_access(context,
&basicMathAvailableHandle);
```

```
   if (status == ECOA__return_status_OK || status ==
ECOA__return_status_DATA_NOT_INITIALIZED)
   {
      *(basicMathAvailableHandle.data) = context->user.basicMathServiceAvailable;
   }

   ECOA__log log;

   if (context->user.basicMathServiceAvailable)
   {
      log.current_size = sprintf((char *) &log.data, "- - - - Publishing basicMath
service available");
   }
   else
   {

      log.current_size = sprintf((char *) &log.data, "- - - - Publishing basicMath
service unavailable");
   }
   Manager_Im_container__log_info(context, log);

   status = Manager_Im_container__available__publish_write_access(context,
&basicMathAvailableHandle);
}
```

The main logic of the Manager module is implemented on the periodic trigger operation "publish". This function performs different actions depending on the current "count" state data.

The *BasicMath_Instance* module will not handle correctly handle any math operation requests in the until the service is set as functionally available (each operation will act in a different manner to show different ways of managing operations being called in the wrong state). The *ComplexMath_Instance* module has no premise of an availability state, so will respond to operations as soon as they are queued (i.e. once the module is "technically" running – even though it may not be "functionally" running).

When count is equal to 1, the Manager requests both the *BasicMath_Instance* and *ComplexMath_Instance* modules to initialize.

When count is equal to 2, the Manager requests both the *BasicMath_Instance* and *ComplexMath_Instance* modules to start.

At this point, the *ProvidedBasicMath* is set as available and the modules will begin handling requests.

When count is 6, the Manager requests both the *BasicMath_Instance* and *ComplexMath_Instance* modules to re-initialize. This has the effect of setting the *ProvidedBasicMath* service as

unavailable. The count is also reset, at which point the sequence of behaviours described above is repeated.

```
void Manager_Im__publish__received(Manager_Im__context *context)
{
   ECOA__log log;

   context->user.count++;

   ECOA__uint32 ID;

   // Initialize modules on count 1
   if (context->user.count == 1)
   {
      log.current_size = sprintf((char *) &log.data, "1 - - - - requesting
initialisation of modules");
      Manager_Im_container__log_info(context, log);

      Manager_Im_container__initializeBasicMath__request_async(context, &ID);
      Manager_Im_container__initializeComplexMath__request_async(context, &ID);
   }
   // Start modules on count 2
   if (context->user.count == 2)
   {
      log.current_size = sprintf((char *) &log.data, "2 - - - - requesting start
of modules");
      Manager_Im_container__log_info(context, log);

      Manager_Im_container__startBasicMath__request_async(context, &ID);
      Manager_Im_container__startComplexMath__request_async(context, &ID);
   }
   // Reinitialize modules on count 6 (set to not running...)
   if (context->user.count == 6)
   {
      log.current_size = sprintf((char *) &log.data, "4 - - - - requesting
reinitialise of modules");
      Manager_Im_container__log_info(context, log);
      context->user.basicMathState = UNINITIALIZED;
      context->user.complexMathState = UNINITIALIZED;
      Manager_Im_container__initializeBasicMath__request_async(context, &ID);
      Manager_Im_container__initializeComplexMath__request_async(context, &ID);
      // Publish functional availability of service.
      context->user.basicMathServiceAvailable = ECOA__FALSE;
      context->user.count = 0;
   }

   // Always republish the current availability
   Publish_Functional_Service_Availability(context);
}
```

The responses to the requests for modules *BasicMath_Instance* and *ComplexMath_Instance* simply update the current functional state of the modules:

```
void Manager_Im__initializeBasicMath__response_received
   (Manager_Im__context* context,
    const ECOA__uint32 ID, const ECOA__return_status status)
{
   context->user.basicMathState = INITIALIZED;
}
void Manager_Im__initializeComplexMath__response_received
   (Manager_Im__context* context,
    const ECOA__uint32 ID, const ECOA__return_status status)
{
   context->user.complexMathState = INITIALIZED;
}
void Manager_Im__startBasicMath__response_received
   (Manager_Im__context* context,
    const ECOA__uint32 ID, const ECOA__return_status status)
{
   context->user.basicMathState = RUNNING;
   context->user.basicMathServiceAvailable = ECOA__TRUE;
}
void Manager_Im__startComplexMath__response_received
   (Manager_Im__context* context,
    const ECOA__uint32 ID, const ECOA__return_status status)
{
   context->user.complexMathState = RUNNING;
}
```

BasicMath_Im Module

The "ProvidedBasicMath" Service operation request operation handlers are implemented in the (C) code unit *BasicMath_Im.c*. Each operation handler demonstrates a different method of handling functional service availability.

The "*add*" functionality is implemented by the following (C) code:

```
void BasicMath_Im__add__request_received
   (BasicMath_Im__context* context,
    const ECOA__uint32 ID,
    const ECOA__int32 value1,
    const ECOA__int32 value2)
{
   // The behaviour of the add operation is:
   // Undefined if the service is not "functionally" available. The module does
not check the state before responding!

   ECOA__return_status status;

   ECOA__int32 result = value1 + value2;
   status = BasicMath_Im_container__add__response_send(context, ID, result);
}
```

This function performs a simple addition operation on the two input parameter values. A response is then sent immediately to the client containing the result of the addition. Note that this function

does not take into account the functional availability of the service and so the operation will complete successfully even if the service has not been set functionally available.

The "*subtract*" functionality is implemented by the following (C) code:

```
void BasicMath_Im__subtract__request_received
   (BasicMath_Im__context* context,
    const ECOA__uint32 ID,
    const ECOA__int32 value1,
    const ECOA__int32 value2)
{
   // The behaviour of the subtract operation is:
   // Check if the service is available.
   // If it is available, send a response.
   // If it is not available, do not send a response.

   ECOA__return_status status;

   if (context->user.available)
   {
      ECOA__int32 result = value1 - value2;
      status = BasicMath_Im_container__subtract__response_send(context, ID,
result);
   }
}
```

This function checks to ensure that the service has been set as functionally available prior to performing a simple subtraction operation on the two input parameter values. A response is then sent immediately to the client containing the result of the subtraction. This implementation may mean that a client will not receive a response if it has attempted to send a request when the service is set as unavailable. A client using this service operation should use a timeout to ensure the request does not block indefinitely or overflow the maximum concurrent request.

The "*multiply*" functionality is implemented by the following (C) code:

```
void BasicMath_Im__multiply__request_received
   (BasicMath_Im__context* context,
    const ECOA__uint32 ID,
    const ECOA__int32 value1,
    const ECOA__int32 value2)
{
   // The behaviour of the multiply operation is:
   // Check if the service is available.
   // If it is available, send a response.
   // If it is not available, send a response, but with a default value.

   ECOA__return_status status;
   ECOA__int32 result = 0;

   if (context->user.available)
   {
```

```
      result = value1 * value2;
      status = BasicMath_Im_container__multiply__response_send(context, ID,
result);
   }
   else
   {
      status = BasicMath_Im_container__multiply__response_send(context, ID,
result);
   }
}
```

This function checks to ensure that the service has been set as functionally available prior to performing a simple multiplication operation on the two input parameter values. A response is then sent immediately to the client containing the result of the multiplication. If the service is not available, a response is sent with a default value of 0 for the result. Note that this is not a robust solution for this functionality, as a 0 value could be a valid result, but the intent is to show that a default value could be used in appropriate situations.

The "*divide*" functionality is implemented by the following (C) code:

```
void BasicMath_Im__divide__request_received
   (BasicMath_Im__context* context,
    const ECOA__uint32 ID,
    const ECOA__int32 value1,
    const ECOA__int32 value2)
{
   // The behaviour of the divide operation is:
   // Check if the service is available.
   // If it is available, send a response.
   // If it is not available, send a response, but with an "Unavailable" status
and default value.

   ECOA__return_status status;
   ECOA__int32 result = 0;
   BasicMath__Divide_Status_Type divideStatus = BasicMath__Divide_Status_Type_OK;

   if (context->user.available)
   {
      ECOA__int32 result = value1 / value2;
      status = BasicMath_Im_container__divide__response_send(context, ID, result,
divideStatus);
   }
   else
   {
      // Return not available.
      divideStatus = BasicMath__Divide_Status_Type_Unavailable;
      status = BasicMath_Im_container__divide__response_send(context, ID, result,
divideStatus);
   }
}
```

This function checks to ensure that the service has been set as functionally available prior to performing a simple division operation on the two input parameter values. A response is then sent immediately to the client containing the result of the division and an "OK" status. If the service is not available, a response is sent with a default value of 0 for the result and a "Not Available" status.

The "*functionalInitialize*" is implemented by the following (C) code:

```
void BasicMath_Im__functionalInitialize__request_received
   (BasicMath_Im__context* context,
    const ECOA__uint32 ID)
{
   // Add an artificial delay (to mimic a heavy-weight initialisation process).
   int i,j;
   for (i; i <= 500000000; i++)
   {
      while (j <= 500000000)
      {
         j++;
      }
   }
   BasicMath_Im_container__functionalInitialize__response_send(context, ID);
}
```

This function mimics a heavy-weight (or time intensive task). Once this task is completed, a response is sent to the Manager module to confirm initialization is complete.

The "*functionalStart*" is implemented by the following (C) code:

```
void BasicMath_Im__functionalStart__request_received
   (BasicMath_Im__context* context,
    const ECOA__uint32 ID)
{
   BasicMath_Im_container__functionalStart__response_send(context, ID);
}
```

This function does no functional work other than sending a response to the Manager to confirm that the start operation is complete.

The module also gets notified of changes to the service availability state (which is set by the Manager module) in the following (C) code:

```
void BasicMath_Im__available__updated(BasicMath_Im__context* context)
{
   BasicMath_Im_container__available_handle availableHandle;

   ECOA__return_status status =
BasicMath_Im_container__available__get_read_access(context, &availableHandle);

   if (status == ECOA__return_status_OK)
   {
      // Update user context state.
```

```
      context->user.available = *(availableHandle.data);
      status = BasicMath_Im_container__available__release_read_access(context,
&availableHandle);
   }
}
```

This function simply gets the latest available state and stores it in user context for use when handling operations.

ComplexMath_Im Module

The "ProvidedComplexMath" Service operation request operation handlers are implemented in the (C) code unit *ComplexMath_Im.c*. This service does not have the concept of functional availability. A client is free to call the operation at any time.

The "*power*" functionality is implemented by the following (C) code:

```
void ComplexMath_Im__power__request_received
   (ComplexMath_Im__context* context,
    const ECOA__uint32 ID,
    const ECOA__int32 base,
    const ECOA__int32 exponent)
{
   ECOA__return_status status;

   ECOA__int32 result = pow(base, exponent);
   status = ComplexMath_Im_container__power__response_send(context,ID, result);
}
```

The "*squareRoot*" functionality is implemented by the following (C) code:

```
void ComplexMath_Im__squareRoot__request_received
   (ComplexMath_Im__context* context,
    const ECOA__uint32 ID,
    const ECOA__int32 value)
{
   ECOA__return_status status;

   ECOA__int32 result = sqrt(value);
   status = ComplexMath_Im_container__squareRoot__response_send(context,ID,
result);
}
```

The "*functionalInitialize*" is implemented by the following (C) code:

```
void ComplexMath_Im__functionalInitialize__request_received
   (ComplexMath_Im__context* context,
    const ECOA__uint32 ID)
{
   // Add an artificial delay (to mimic a heavy-weight initialisation process).
   int i,j;
   for (i; i <= 500000000; i++)
   {
```

```
      while (j <= 500000000)
      {
         j++;
      }
   }

   ComplexMath_Im_container__functionalInitialize__response_send(context, ID);
}
```

This function mimics a heavy-weight (or time intensive task). Once this task is completed, a response is sent to the Manager module to confirm initialization is complete.

The "*functionalStart*" is implemented by the following (C) code:

```
void ComplexMath_Im__functionalStart__request_received
   (ComplexMath_Im__context* context,
    const ECOA__uint32 ID)
{
   ComplexMath_Im_container__functionalStart__response_send(context, ID);
}
```

This function does no functional work other than sending a response to the Manager to confirm that the start operation is complete.

The Client ASC

All we need to do is program what to do when the *Internal_Trigger_Instance* Trigger Instance fires, i.e. to populate the *Client_Module_Im__tick__received* function stub.

```
void Client_Module_Im__tick__received(Client_Module_Im__context *context)
{
   ECOA__return_status status;

   // Basic math should only be used if the service has been set as functionally
available...
   // However, we can use any except subtract as the other service operations are
designed to send a response regardless.
   testAddition(context);
   testSubtraction(context);
   testMultiplication(context);
   testDivision(context);

   // The exponential math can be used anytime...
   testPower(context);
   testSquareRoot(context);

   ECOA__log log;
   log.current_size = sprintf((char *) &log.data, "-------------------------------
--------------------");
   Client_Module_Im_container__log_info(context, log);

}
```

At each period, a synchronous Request-Response call is made to each of the math operations available in "RequiredBasicMath" and "RequiredComplexMath". This is done by the invocation of a number of user-written methods which are detailed below. In each method, a log is made before invoking the respective container operation. In this example, the functional availability of the "RequiredBasicMath" is not taken into consideration with the exception of the subtract operation ("*testSubtraction()*").

Before sending the subtract request, the "*BasicMathServiceAvailable*" versioned data is interrogated to check if the service has been set as functionally available. This is due to the fact that the server requirement for this operation is to not send a response. If the request is made regardless, the module would become blocked indefinitely. Note that there is a race-condition which means the service could be available when the request is made, but unavailable when it reaches the server; it is therefore always advisable to set a timeout on the client request operation to cater for this scenario.

```
static void testAddition(Client_Module_Im__context *context)
{
   ECOA__log log;
   ECOA__return_status status;

   ECOA__int32 value1 = 5;
   ECOA__int32 value2 = 10;
   ECOA__int32 result = 0;

   log.current_size = sprintf((char *) &log.data, "requesting addition of %d and
%d", value1, value2);
   Client_Module_Im_container__log_info(context, log);

   status = Client_Module_Im_container__add__request_sync(context, value1, value2,
&result);

   log.current_size = sprintf((char *) &log.data, "result of addition of %d and %d
= %d", value1, value2, result);
   Client_Module_Im_container__log_info(context, log);
}

static void testSubtraction(Client_Module_Im__context *context)
{
   ECOA__log log;
   ECOA__return_status status;
   Client_Module_Im_container__BasicMathServiceAvailable_handle availableHandle;

   ECOA__int32 value1 = 50;
   ECOA__int32 value2 = 10;
   ECOA__int32 result = 0;

   // The division operation should check if the service is functionally
available, as the server
```

```
   // behaviour is defined to not respond if a request is received when
functionally unavailable.
   // This could lead to the client module being blocked indefinitely if no
timeout is specified!
   status =
Client_Module_Im_container__BasicMathServiceAvailable__get_read_access(context,
&availableHandle);

   if (status == ECOA__return_status_OK)
   {
      if (*(availableHandle.data) == ECOA__TRUE)
      {
         log.current_size = sprintf((char *) &log.data, "requesting subtraction of
%d minus %d", value1, value2);
         Client_Module_Im_container__log_info(context, log);

         status = Client_Module_Im_container__subtract__request_sync(context,
value1, value2, &result);

         log.current_size = sprintf((char *) &log.data, "result of subtraction of
%d minus %d = %d", value1, value2, result);
         Client_Module_Im_container__log_info(context, log);
      }
      else
      {
         log.current_size = sprintf((char *) &log.data, "cannot perform
subtraction as service unavailable");
         Client_Module_Im_container__log_info(context, log);
      }

      status =
Client_Module_Im_container__BasicMathServiceAvailable__release_read_access(context
, &availableHandle);
   }
}
```

Before sending the multiplication request, the "*BasicMathServiceAvailable*" versioned data is interrogated to check if the state data has been updated since the last time (by checking the '*stamp*'). The request will be sent irrespective of the actual state of the service availability, but will only be sent if the state has not become stale. Since initially the service is set as unavailable but the state is being continually published, then the server will respond with a default value. Once the service is set as available, then the correct multiplication result will be returned. Finally, when the server stops publishing the state, the client will detect it has become stale and no-longer send the request.

```
static void testMultiplication(Client_Module_Im__context *context)
{
   ECOA__log log;
   ECOA__return_status status;
   Client_Module_Im_container__BasicMathServiceAvailable_handle availableHandle;

   ECOA__int32 value1 = 7;
```

```
   ECOA__int32 value2 = 8;
   ECOA__int32 result = 0;

   // When the server is declaring its service as unavailable, then it will return
a default value for the multiplication.
   // When the server stops periodically publishing its availability then assume
the service is not available
   status =
Client_Module_Im_container__BasicMathServiceAvailable__get_read_access(context,
&availableHandle);

   if (status == ECOA__return_status_OK)
   {
      if(availableHandle.stamp != context->user.previousStamp)
      {
         context->user.previousStamp = availableHandle.stamp;
         log.current_size = sprintf((char *) &log.data, "requesting multiplication
of %d by %d", value1, value2);
         Client_Module_Im_container__log_info(context, log);

         status = Client_Module_Im_container__multiply__request_sync(context,
value1, value2, &result);

         log.current_size = sprintf((char *) &log.data, "result of multiplication
of %d by %d = %d", value1, value2, result);
         Client_Module_Im_container__log_info(context, log);

      }
      else
      {
         log.current_size = sprintf((char *) &log.data, "server availability is
stale - not requesting multiplication");
         Client_Module_Im_container__log_info(context, log);
      }
   }
}

static void testDivision(Client_Module_Im__context *context)
{
   ECOA__log log;
   ECOA__return_status status;

   ECOA__int32 value1 = 1000;
   ECOA__int32 value2 = 20;
   ECOA__int32 result = 0;
   BasicMath__Divide_Status_Type divideStatus;

   log.current_size = sprintf((char *) &log.data, "requesting division of %d by
%d", value1, value2);
   Client_Module_Im_container__log_info(context, log);

   status = Client_Module_Im_container__divide__request_sync(context, value1,
value2, &result, &divideStatus);
```

```
   if (divideStatus == BasicMath__Divide_Status_Type_OK)
   {
      log.current_size = sprintf((char *) &log.data, "result of division of %d by
%d = %d", value1, value2, result);
      Client_Module_Im_container__log_info(context, log);
   }
   else
   {
      log.current_size = sprintf((char *) &log.data, "Failed to divide - status =
%d", divideStatus);
      Client_Module_Im_container__log_info(context, log);
   }
}

static void testPower(Client_Module_Im__context *context)
{
   ECOA__log log;
   ECOA__return_status status;

   ECOA__int32 base = 3;
   ECOA__int32 exponent = 4;
   ECOA__int32 result = 0;

   log.current_size = sprintf((char *) &log.data, "requesting %d raised to the
power %d", base, exponent);
   Client_Module_Im_container__log_info(context, log);

   status = Client_Module_Im_container__power__request_sync(context, base,
exponent, &result);

   log.current_size = sprintf((char *) &log.data, "result of %d raised to the
power %d = %d", base, exponent, result);
   Client_Module_Im_container__log_info(context, log);
}

static void testSquareRoot(Client_Module_Im__context *context)
{
   ECOA__log log;
   ECOA__return_status status;

   ECOA__int32 value1 = 25;
   ECOA__int32 result = 0;

   log.current_size = sprintf((char *) &log.data, "requesting square root of %d",
value1);
   Client_Module_Im_container__log_info(context, log);

   status = Client_Module_Im_container__squareRoot__request_sync(context, value1,
&result);

   log.current_size = sprintf((char *) &log.data, "result of square root of %d =
%d", value1, result);
   Client_Module_Im_container__log_info(context, log);
}
```

Program Output

When the ECOA "*Manager Module Example*" Assembly is built and run (in a single Node deployment), an output similar to Figure 7 should be achieved. The `Client` ASC outputs, at each iteration, the values before sending each request message, and the value after receiving the corresponding response.

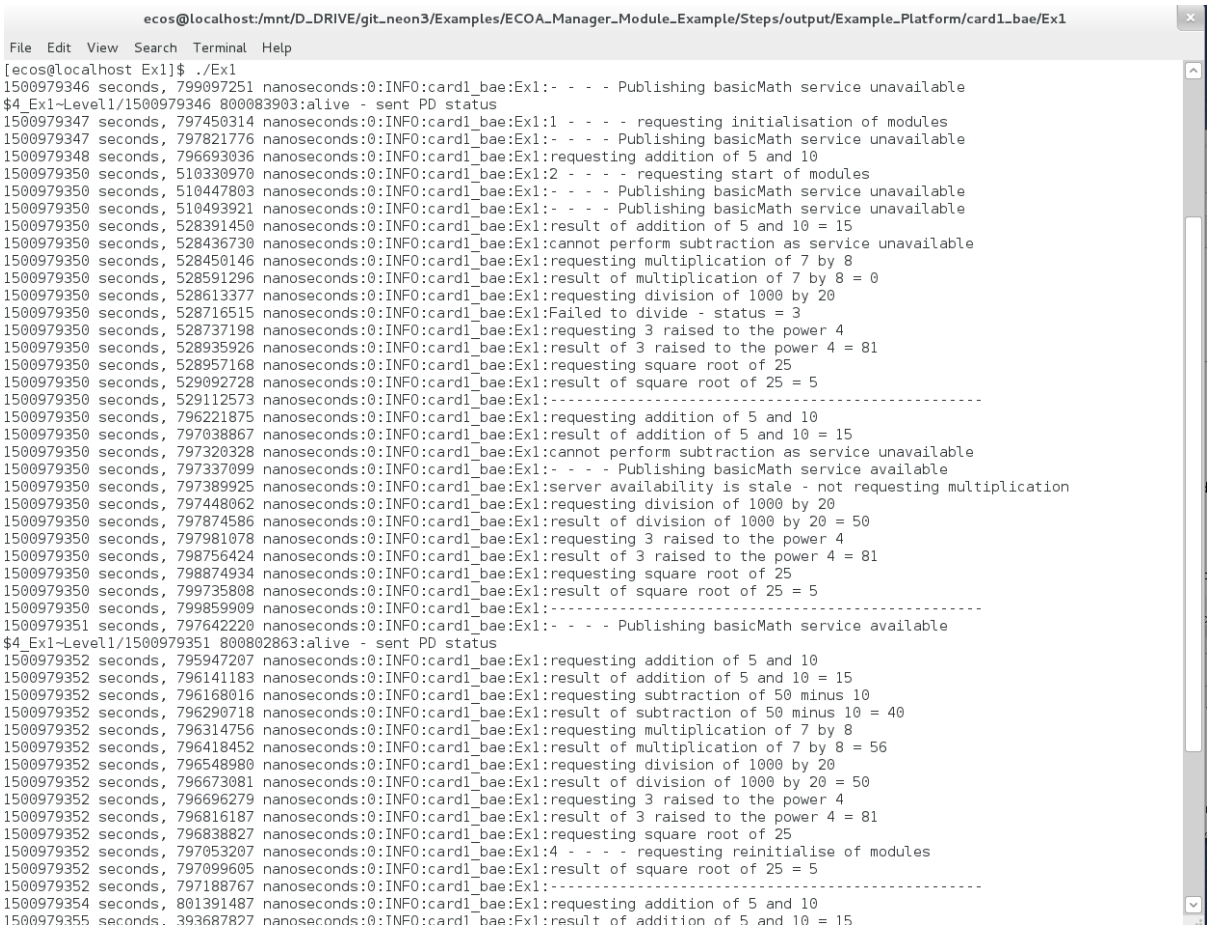

Figure 7 - ECOA "*Manager Module Example*" in Execution

References

1	European Component Oriented Architecture (ECOA) Collaboration Programme: Architecture Specification (Parts 1 to 11) "ECOA" is a registered trade mark.
2	Client-server model *https://en.wikipedia.org/wiki/Client%E2%80%93server_model*
3	Service Availability Example *http://www.ecoa.technology/tutorials.html*