Wire Switch Example

Introduction

This document describes an ECOA® wire switch (Service Link switch) example named “*Wire Switch Example*”.

It is based upon the concepts introduced in the ECOA “*Simple Example*” (ref. [2]), with extensions showing two design patterns which could be used to implement a service switching mechanism in ECOA.

This document presents the principal user generated artefacts required to create the “*Wire Switch Example*” 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 “*Wire Switch Example*” example is intended to demonstrate a number of design patters which can be used by an ECOA system designer in order to provide a wire switching mechanism. An example of when this sort of mechanism may be useful is when multiple service providers are required (e.g. for redundancy) and there is a preference to use one provider over another in normal operation (e.g. it has a better quality of service, response time, update rate or more accurate data).

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.
- Use of a “broker” component to manage service connectivity.
- Use of a “client” component with multiple service instances.

Design and Definition

Wire Switch Functional Design

The “Wire Switch” example will demonstrate two methods of implementing a service switching mechanism.

The first method will involve a “brokered” client component which will periodically perform a request, from a server and will receive a data item in return. This “brokered” client component is the connected to a “broker” component, which is responsible for routing the request to the preferred server if available, or the backup server if not. In this scenario, the “brokered” client is fully isolated from the knowledge that multiple servers even exist in the system. This is advantageous, as the same Component may be deployed into systems where there is only a single Server, two servers, or in fact any number of servers without requiring modification. The disadvantage of this approach is that extra latency will be incurred as any operations will pass through the “broker” component.

The second method will involve a “non-brokered” client component which will periodically perform a request, from a server and will receive a data item in return. This “non-brokered” client is “aware” that there are multiple servers available and is responsible for choosing the preferred provider itself. This approach is advantageous as there are no latency penalties. However, the approach limits the reusability of the Component, as it will only be able to be deployed into systems with the same number of Servers available (i.e. 2 in this case).

In each case, the data content of the request will be the current absolute time and the response will be of a user defined type.

Both clients will set a local variable to zero and output this to the log prior to performing the request. The result will be returned into this variable and logged.

Both clients will be periodically activated at a rate of 0.5Hz (once every 2 seconds).

ECOA Assembly Design and Definition

This ECOA “*Wire Switch Example*” example ECOA Assembly comprises five ECOA ASCs named “*BrokeredClient*”, “*NonBrokeredClient*”, “*ServiceBroker*” and “*Server*”. The “*BrokeredClient*” ASC type is instantiated once within the ECOA Assembly as “*BrokeredClient_Inst*”. The “*NonBrokeredClient*” ASC type is instantiated once within the ECOA Assembly as “*NonBrokeredClient_Inst*”. The “*ServiceBroker*” ASC type is instantiated once within the ECOA Assembly as “*ServiceBroker_Inst*”. The “*Server*” ASC is instantiated twice (different implementations) within the ECOA Assembly as “*PreferredServer_Inst*” and “*BackupServer_Inst*” as depicted in Figure 1.

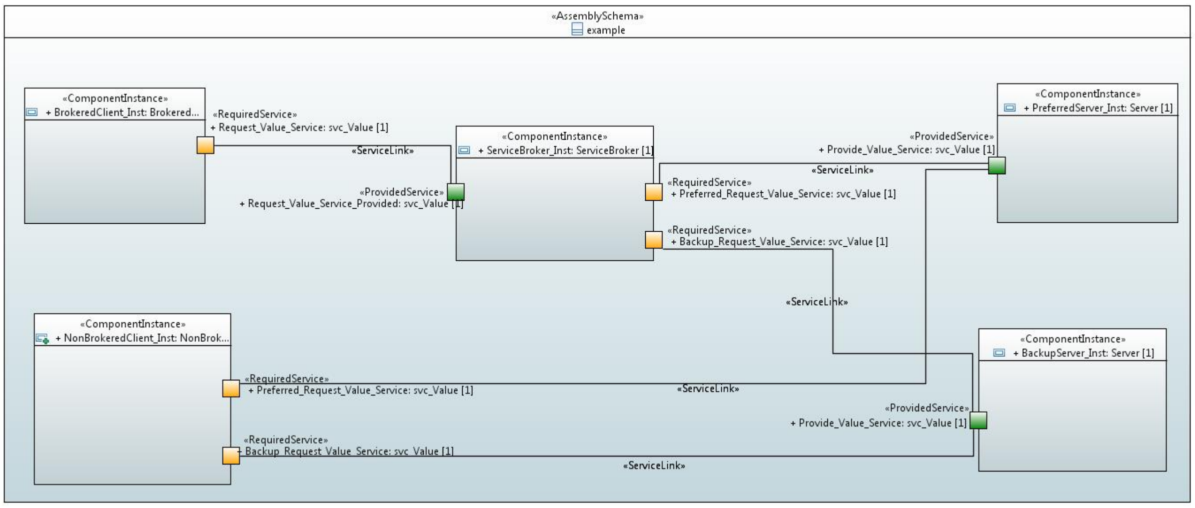

«AssemblySchema»
example
«ComponentInstance»
+ BrokeredClient_Inst: Brokered...
«RequiredService»
+ Request_Value_Service: svc_Value [1]
«ServiceLink»
«ProvidedService»
+ Request_Value_Service_Provided: svc_Value [1]
«ComponentInstance»
+ ServiceBroker_Inst: ServiceBroker [1]
«RequiredService»
+ Preferred_Request_Value_Service: svc_Value [1]
«RequiredService»
+ Backup_Request_Value_Service: svc_Value [1]
«ComponentInstance»
+ PreferredServer_Inst: Server [1]
«ProvidedService»
+ Provide_Value_Service: svc_Value [1]
«ServiceLink»
«ServiceLink»
«ComponentInstance»
+ NonBrokeredClient_Inst: NonBrok...
«RequiredService»
+ Preferred_Request_Value_Service: svc_Value [1]
«ServiceLink»
«RequiredService»
+ Backup_Request_Value_Service: svc_Value [1]
«ServiceLink»
«ComponentInstance»
+ BackupServer_Inst: Server [1]
«ProvidedService»
+ Provide_Value_Service: svc_Value [1]

Figure 1 - ECOA "Wire Switch" 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 "*Wire Switch Example*" Assembly is as follows (file *example.impl.composite*):

```xml
<?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="BrokeredClient_Inst">
      <ecoa-sca:instance componentType="BrokeredClient"/>

      <csa:reference name="Request_Value_Service">
         <ecoa-sca:interface syntax="svc_Value"/>
      </csa:reference>

   </csa:component>

   <csa:component name="PreferredServer_Inst">
      <ecoa-sca:instance componentType="Server"/>

      <csa:service name="Provide_Value_Service">
         <ecoa-sca:interface syntax="svc_Value"/>
      </csa:service>

   </csa:component>

   <csa:component name="BackupServer_Inst">
      <ecoa-sca:instance componentType="Server"/>
      <csa:service name="Provide_Value_Service">
         <ecoa-sca:interface syntax="svc_Value"/>
      </csa:service>

   </csa:component>

   <csa:component name="ServiceBroker_Inst">
      <ecoa-sca:instance componentType="ServiceBroker"/>

      <csa:service name="Request_Value_Service_Provided">
         <ecoa-sca:interface syntax="svc_Value"/>
      </csa:service>

      <csa:reference name="Preferred_Request_Value_Service">
         <ecoa-sca:interface syntax="svc_Value"/>
      </csa:reference>

      <csa:reference name="Backup_Request_Value_Service">
         <ecoa-sca:interface syntax="svc_Value"/>
      </csa:reference>

   </csa:component>

   <csa:component name="NonBrokeredClient_Inst">
      <ecoa-sca:instance componentType="NonBrokeredClient"/>

      <csa:reference name="Preferred_Request_Value_Service">
         <ecoa-sca:interface syntax="svc_Value"/>
      </csa:reference>

      <csa:reference name="Backup_Request_Value_Service">
         <ecoa-sca:interface syntax="svc_Value"/>
      </csa:reference>

   </csa:component>

   <csa:wire source="NonBrokeredClient_Inst/Preferred_Request_Value_Service"
target="PreferredServer_Inst/Provide_Value_Service"/>

   <csa:wire source="NonBrokeredClient_Inst/Backup_Request_Value_Service"
target="BackupServer_Inst/Provide_Value_Service"/>

   <csa:wire source="BrokeredClient_Inst/Request_Value_Service"
target="ServiceBroker_Inst/Request_Value_Service_Provided"/>

   <csa:wire source="ServiceBroker_Inst/Preferred_Request_Value_Service"
target="PreferredServer_Inst/Provide_Value_Service"/>

   <csa:wire source="ServiceBroker_Inst/Backup_Request_Value_Service"
target="BackupServer_Inst/Provide_Value_Service"/>

</csa:composite>
```

The *Server* ASC type is defined in XML as follows (file *Server.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="Provide_Value_Service">
      <ecoa-sca:interface syntax="svc_Value"/>
   </service>

</componentType>
```

The ASC definition (the *<componentType>* XML element) declares the provision (by the ASC) of the *Provide_Value_Service* ECOA Service.

The *BrokeredClient* ASC type is defined in XML as follows (file *BrokeredClient.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="Request_Value_Service">
      <ecoa-sca:interface syntax="svc_Value"/>
   </reference>

</componentType>
```

This ASC definition (the *<componentType>* XML element) declares a reference (by the ASC) to the *Request_Value_Service* ECOA Service.

The *NonBrokeredClient* ASC type is defined in XML as follows (file *NonBrokeredClient.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="Preferred_Request_Value_Service">
      <ecoa-sca:interface syntax="svc_Value"/>
   </reference>

   <reference name="Backup_Request_Value_Service">
      <ecoa-sca:interface syntax="svc_Value"/>
   </reference>

</componentType>
```

This ASC definition (the *<componentType>* XML element) declares two references (by the ASC) to the *Request_Value_Service* ECOA Service.

The *ServiceBroker* ASC type is defined in XML as follows (file *ServiceBroker.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="Request_Value_Service_Provided">
      <ecoa-sca:interface syntax="svc_Value"/>
   </service>

   <reference name="Preferred_Request_Value_Service">
      <ecoa-sca:interface syntax="svc_Value"/>
   </reference>

   <reference name="Backup_Request_Value_Service">
      <ecoa-sca:interface syntax="svc_Value"/>
   </reference>

</componentType>
```

This ASC definition (the *<componentType>* XML element) declares two references (by the ASC) to the *Request_Value_Service* ECOA Service and also one provision (by the ASC) of the *Request_Value_Service* ECOA Service.

ECOA Service and Types Definition

The *svc_Value* Service is defined in a XML file (*svc_Value.interface.xml*):

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

   <use library="example"/>

   <operations>
      <requestresponse name="Request_Value">
         <input name="Time" type="ECOA:global_time"/>
         <output name="Value" type="example:value_type"/>
      </requestresponse>

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

   </operations>
</serviceDefinition>
```

The Service comprises an ECOA Request-Response Operation called *Request_Value* which has one input parameter (*Time* which is passed from the requesting client to the server), and one output parameter (*Value* which is the response from the server to the client). The first parameter is defined as being of type *global_time*, which is a pre-defined ECOA type. The second parameter is defined as being of type *example:value_type*, where *example* names a data types library *used* by the service definition. The data types library is, unsurprisingly, also defined in XML (file *example.types.xml*):

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

        <types>
                <simple name="value_type" type="uint32" />
        </types>

</library>
```

The data type *example:value_type* is therefore an unsigned 32 bit integer type.

In addition, the service defines an ECOA Versioned-Data Operation called *Available* which is of type *boolean8*. This operation is used to control the functional availability of the service.

ECOA Module Design and Definition

The implementations of each of the Components are composed of a single ECOA Module. This is illustrated in UML in Figure 2, Figure 3, **Error! Reference source not found.**Figure 4, Figure 5 and Figure 6.

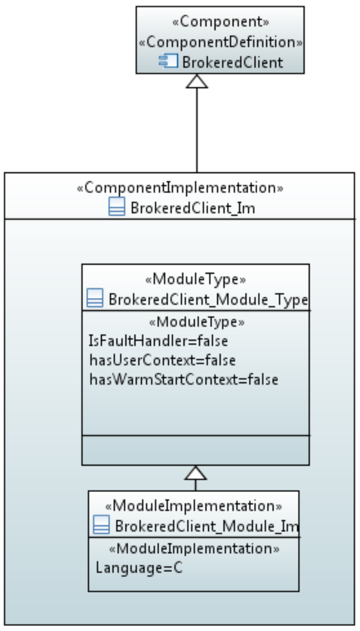


Figure 2 "BrokeredClient_Im" Module Design (as UML Composite Structure Diagram)

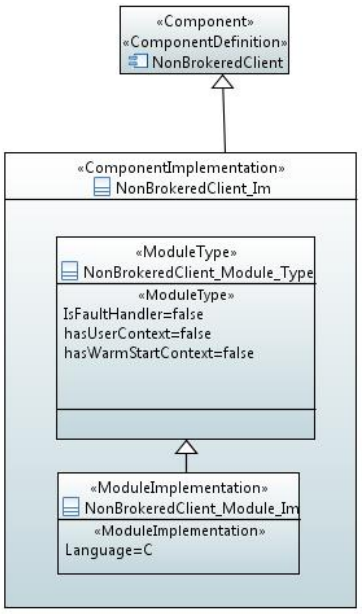


Figure 3 – "NonBrokeredClient_Im" Module Design (as UML Composite Structure Diagram)

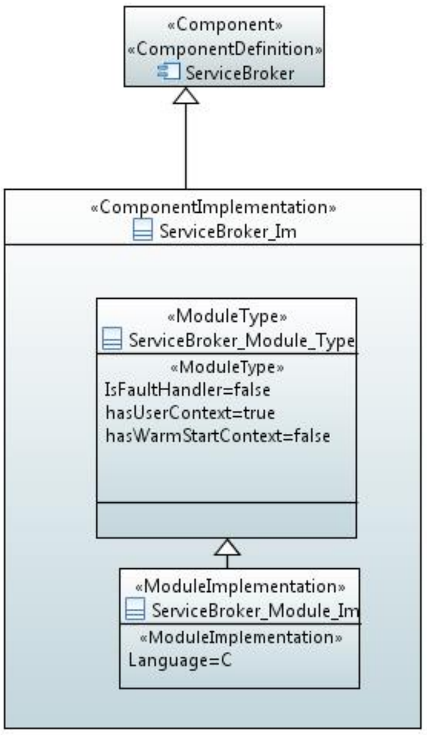


Figure 4 – "ServiceBroker_Im" Module Design (as UML Composite Structure Diagram)

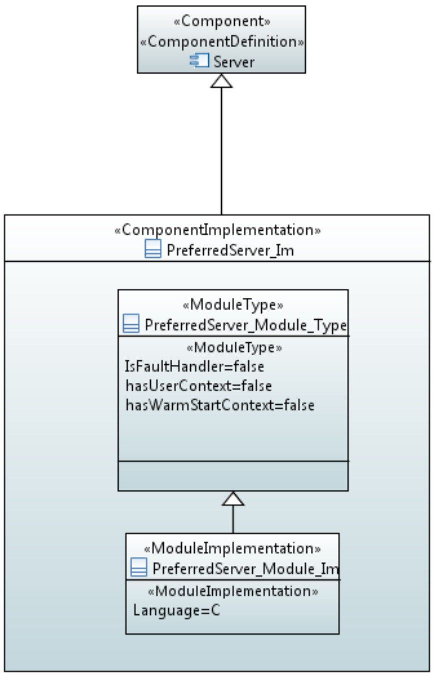


Figure 5 – "PreferredServer_Im" Module Design (as UML Composite Structure Diagram)

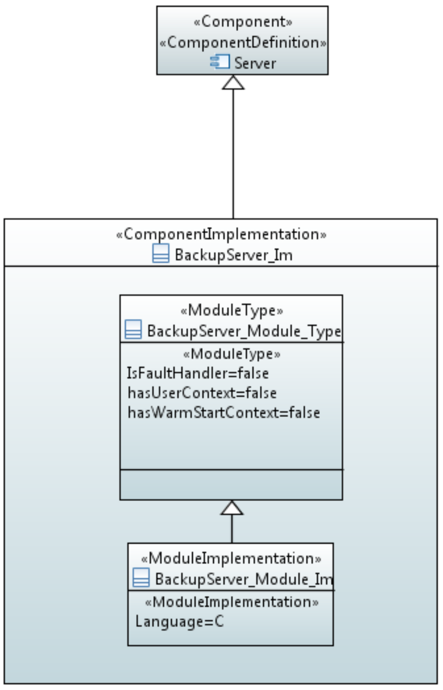


Figure 6 – "BackupServer_Im" Module Design (as UML Composite Structure Diagram)

Figure 7, Figure 8 depict in UML the internal design of each of the components.

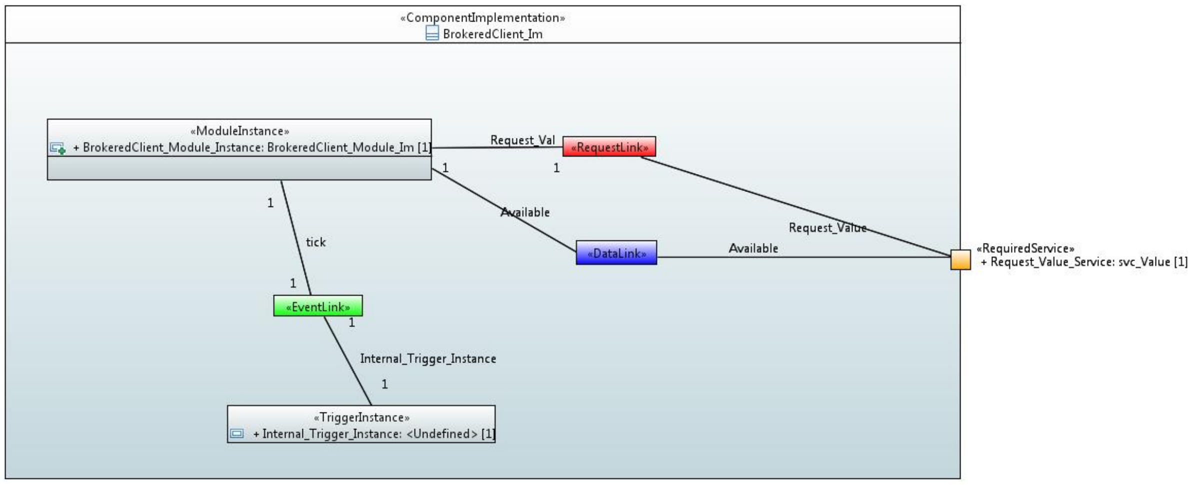

«ComponentImplementation»
BrokeredClient_Im
«ModuleInstance»
+ BrokeredClient_Module_Instance: BrokeredClient_Module_Im [1]
Request_Val
«RequestLink»
Available
«DataLink»
Request_Value
Available
«RequiredService»
+ Request_Value_Service: svc_Value [1]
tick
«EventLink»
Internal_Trigger_Instance
«TriggerInstance»
+ Internal_Trigger_Instance: <Undefined> [1]

Figure 7 - "BrokeredClient_Im" Component Design (as UML Composite Structure Diagram)

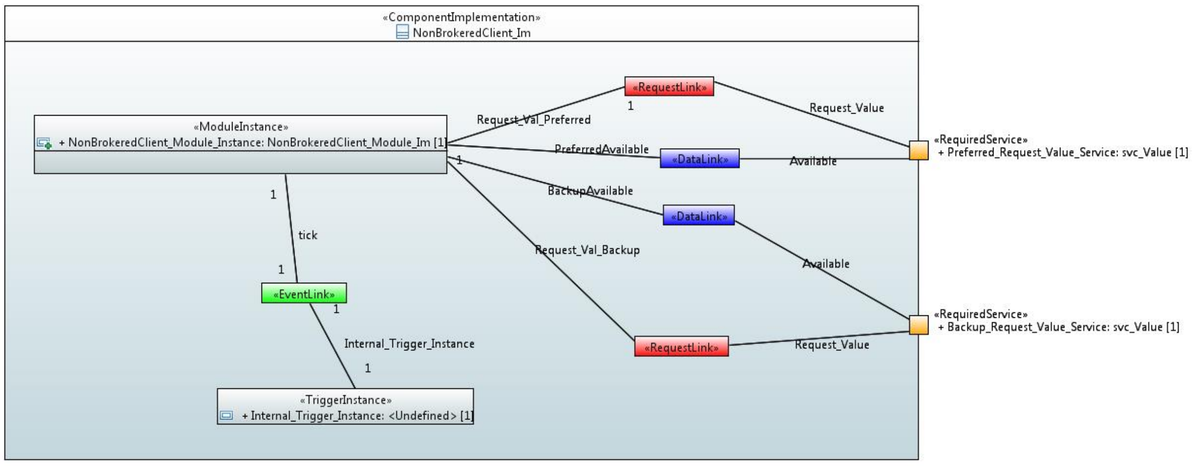


Figure 8 - "NonBrokeredClient_Im" Component Design (as UML Composite Structure Diagram)

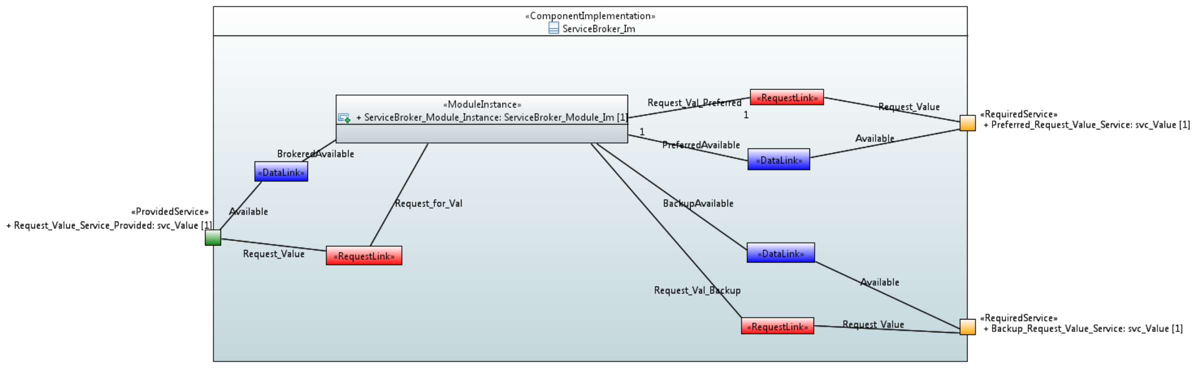


Figure 9 - "ServiceBroker_Im” Component Design (as UML Composite Structure Diagram)

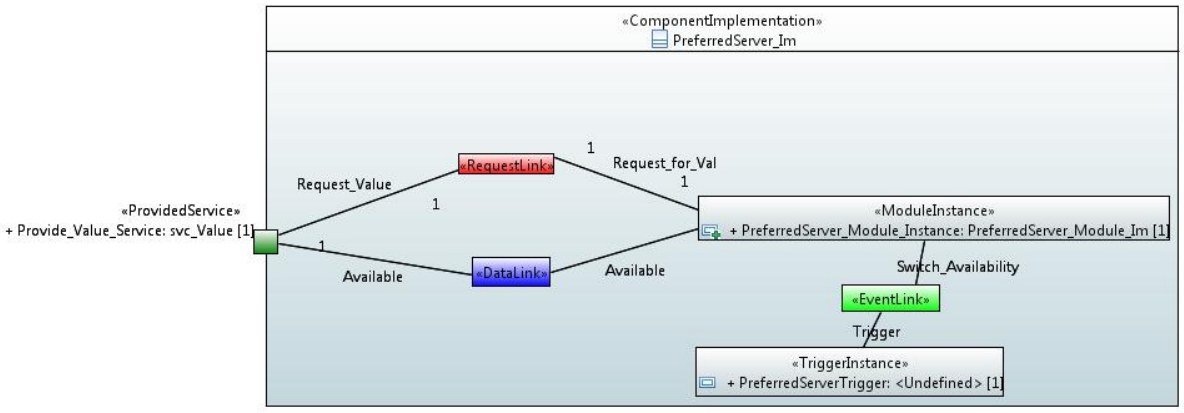


Figure 10 - "PreferredServer_Im” Component Design (as UML Composite Structure Diagram)

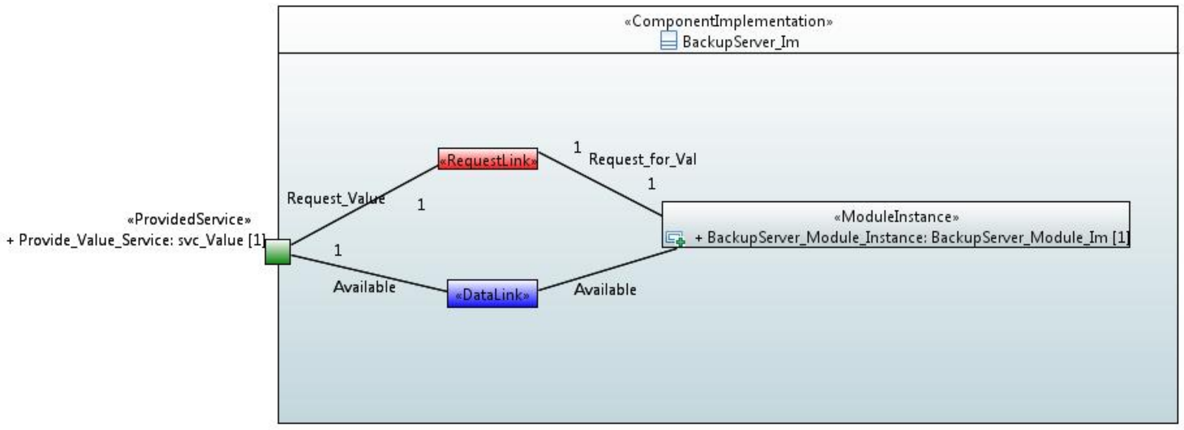

«ComponentImplementation»
BackupServer_Im
«ProvidedService»
+ Provide_Value_Service: svc_Value [1]
Request_Value
1
«RequestLink»
1
Request_for_Val
1
1
Available
«DataLink»
Available
«ModuleInstance»
+ BackupServer_Module_Instance: BackupServer_Module_Im [1]

Figure 11 - "BackupServer_Im” Component Design (as UML Composite Structure Diagram)

The Brokered Client ASC

The `BrokeredClient` ASC is declared in XML as follows (file `BrokeredClient_Im.impl.xml`):

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

   <use library="example"/>

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


      <operations>

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

         <requestSent name="Request_Val" isSynchronous="true" timeout="-1.0"
maxConcurrentRequests="10">
            <input name="time" type="ECOA:global_time"/>
            <output name="val" type="example:value_type"/>
         </requestSent>

         <dataRead name="Available" type="ECOA:boolean8" notifying="false"/>

      </operations>

   </moduleType>


   <moduleImplementation name="BrokeredClient_Module_Im" language="C"
moduleType="BrokeredClient_Module_Type"/>

   <moduleInstance name="BrokeredClient_Module_Instance"
implementationName="BrokeredClient_Module_Im" relativePriority="1">

   </moduleInstance>

   <triggerInstance name="Internal_Trigger_Instance" relativePriority="0"/>


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

      <clients>
         <moduleInstance instanceName="BrokeredClient_Module_Instance"
operationName="Request_Val"/>
      </clients>
      <server>
         <reference instanceName="Request_Value_Service"
operationName="Request_Value"/>
      </server>
   </requestLink>

   <dataLink>
      <writers>
         <reference instanceName="Request_Value_Service"
operationName="Available"/>
      </writers>
      <readers>
         <moduleInstance instanceName="BrokeredClient_Module_Instance"
operationName="Available"/>
      </readers>
   </dataLink>


</componentImplementation>
```

That is, a Module Type (*BrokeredClient_Module_Type*) is declared which has three operations:

- A "*Request_Val*" *requestSent* operation;
- The *eventReceived* operation "*tick*";
- The *dataRead* operation "*Available*".

The *Internal_Trigger_Instance* Trigger Instance is introduced because the Brokered Client needs to periodically request a data item 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 *BrokeredClient_Module_Im*, which in turn is instantiated once as the Module Instance *BrokeredClient_Module_Instance*.

The *<requestLink>* XML logically associates the specific concrete operations of the Module Instance with the abstract Service operations. In this example, the "*Request_Val*" module operation is connected to the "*Request_Value*" service operation of the "*Request_Value_Service*" service instance.

The *<dataLink>* XML logically associates the specific concrete operations of the Module Instance with the abstract Service operations. In this example, the "*Available*" module operation is connected to the "*Available*" service operation of the "*Request_Value_Service*" service instance.

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

The Non Brokered Client ASC

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

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

   <use library="example"/>

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


      <operations>

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

         <requestSent name="Request_Val_Preferred" isSynchronous="true" timeout="-
1.0" maxConcurrentRequests="10">
            <input name="time" type="ECOA:global_time"/>
            <output name="val" type="example:value_type"/>
         </requestSent>

         <requestSent name="Request_Val_Backup" isSynchronous="true" timeout="-
1.0" maxConcurrentRequests="10">
            <input name="time" type="ECOA:global_time"/>
            <output name="val" type="example:value_type"/>
         </requestSent>

         <dataRead name="PreferredAvailable" type="ECOA:boolean8"
notifying="false"/>

         <dataRead name="BackupAvailable" type="ECOA:boolean8" notifying="false"/>

      </operations>

   </moduleType>


   <moduleImplementation name="NonBrokeredClient_Module_Im" language="C"
moduleType="NonBrokeredClient_Module_Type"/>
```

```
   <moduleInstance name="NonBrokeredClient_Module_Instance"
implementationName="NonBrokeredClient_Module_Im" relativePriority="1">

   </moduleInstance>

   <triggerInstance name="Internal_Trigger_Instance" relativePriority="0"/>


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

   <requestLink>

      <clients>
         <moduleInstance instanceName="NonBrokeredClient_Module_Instance"
operationName="Request_Val_Preferred"/>
      </clients>
      <server>
         <reference instanceName="Preferred_Request_Value_Service"
operationName="Request_Value"/>
      </server>
   </requestLink>

   <requestLink>

      <clients>
         <moduleInstance instanceName="NonBrokeredClient_Module_Instance"
operationName="Request_Val_Backup"/>
      </clients>
      <server>
         <reference instanceName="Backup_Request_Value_Service"
operationName="Request_Value"/>
      </server>
   </requestLink>

   <dataLink>
      <writers>
         <reference instanceName="Preferred_Request_Value_Service"
operationName="Available"/>
      </writers>
      <readers>
         <moduleInstance instanceName="NonBrokeredClient_Module_Instance"
operationName="PreferredAvailable"/>
      </readers>
   </dataLink>

   <dataLink>
```

```
      <writers>
         <reference instanceName="Backup_Request_Value_Service"
operationName="Available"/>
      </writers>
      <readers>
         <moduleInstance instanceName="NonBrokeredClient_Module_Instance"
operationName="BackupAvailable"/>
      </readers>
   </dataLink>


</componentImplementation>
```

That is, a Module Type (*NonBrokeredClient_Module_Type*) is declared which has five operations:

- A "*Request_Val_Preferred*" *requestSent* operation;
- A "*Request_Val_Backup*" *requestSent* operation;
- The *eventReceived* operation "*tick*";
- The *dataRead* operation "*PreferredAvailable*";
- The *dataRead* operation "*BackupAvailable*".

The *Internal_Trigger_Instance* Trigger Instance is introduced because the Non-Brokered Client needs to periodically request a data item 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 *NonBrokeredClient_Module_Im*, which in turn is instantiated once as the Module Instance *NonBrokeredClient_Module_Instance*.

The *<requestLink>* XML logically associates the specific concrete operations of the Module Instance with the abstract Service operations. In this example, the "*Request_Val_Preferred*" module operation is connected to the "*Request_Value*" service operation of the "*Preferred_Request_Value_Service*" service instance and the "*Request_Val_Backup*" module operation is connected to the "*Request_Value*" service operation of the "*Backup_Request_Value_Service*" service instance.

The *<dataLink>* XML logically associates the specific concrete operations of the Module Instance with the abstract Service operations. In this example, the "*PreferredAvailable*" module operation is connected to the "*Available*" service operation of the "*Preferred_Request_Value_Service*" service instance and the "*BackupAvailable*" module operation is connected to the "*Available*" service operation of the "*Backup_Request_Value_Service*" service instance.

A single functional code unit will be produced by the code generation process, implementing in code the concrete *NonBrokeredClient_Module_Im* class, and named

"*NonBrokeredClient_Module_Im.c*" (assuming the Module Implementation declaration has set the *language* property to "C").

The Service Broker ASC

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

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

   <use library="example"/>

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


      <operations>

         <requestSent name="Request_Val_Preferred" isSynchronous="true" timeout="-
1.0" maxConcurrentRequests="10">
            <input name="time" type="ECOA:global_time"/>
            <output name="val" type="example:value_type"/>
         </requestSent>

         <requestSent name="Request_Val_Backup" isSynchronous="true" timeout="-
1.0" maxConcurrentRequests="10">
            <input name="time" type="ECOA:global_time"/>
            <output name="val" type="example:value_type"/>
         </requestSent>

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

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

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

         <requestReceived name="Request_for_Val" maxConcurrentRequests="10">
            <input name="time" type="ECOA:global_time"/>
            <output name="val" type="example:value_type"/>
         </requestReceived>

      </operations>

   </moduleType>


   <moduleImplementation name="ServiceBroker_Module_Im" language="C"
moduleType="ServiceBroker_Module_Type"/>

   <moduleInstance name="ServiceBroker_Module_Instance"
implementationName="ServiceBroker_Module_Im" relativePriority="1">
```

```
   </moduleInstance>



   <requestLink>

      <clients>
         <moduleInstance instanceName="ServiceBroker_Module_Instance"
operationName="Request_Val_Preferred"/>
      </clients>
      <server>
         <reference instanceName="Preferred_Request_Value_Service"
operationName="Request_Value"/>
      </server>
   </requestLink>

   <dataLink>
      <writers>
         <reference instanceName="Preferred_Request_Value_Service"
operationName="Available"/>
      </writers>
      <readers>
         <moduleInstance instanceName="ServiceBroker_Module_Instance"
operationName="PreferredAvailable"/>
      </readers>
   </dataLink>

   <requestLink>

      <clients>
         <moduleInstance instanceName="ServiceBroker_Module_Instance"
operationName="Request_Val_Backup"/>
      </clients>
      <server>
         <reference instanceName="Backup_Request_Value_Service"
operationName="Request_Value"/>
      </server>
   </requestLink>

   <dataLink>
      <writers>
         <reference instanceName="Backup_Request_Value_Service"
operationName="Available"/>
      </writers>
      <readers>
         <moduleInstance instanceName="ServiceBroker_Module_Instance"
operationName="BackupAvailable"/>
      </readers>
   </dataLink>

   <dataLink>
      <writers>
         <moduleInstance instanceName="ServiceBroker_Module_Instance"
operationName="BrokeredAvailable"/>
```

```
      </writers>
      <readers>
         <service instanceName="Request_Value_Service_Provided"
operationName="Available"/>
      </readers>
   </dataLink>

   <requestLink>

      <clients>
         <service instanceName="Request_Value_Service_Provided"
operationName="Request_Value"/>
      </clients>
      <server>
         <moduleInstance instanceName="ServiceBroker_Module_Instance"
operationName="Request_for_Val"/>
      </server>
   </requestLink>


</componentImplementation>
```

That is, a Module Type (*ServiceBroker_Module_Type*) is declared which has six operations:

- A "*Request_Val_Preferred*" *requestSent* operation;
- A "*Request_Val_Backup*" *requestSent* operation;
- The *dataRead* operation "*PreferredAvailable*";
- The *dataRead* operation "*BackupAvailable*";
- The *dataWritten* operation "*BrokeredAvailable*";
- The *requestReceived* operation "*Request_for_Val*".

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

The *<requestLink>* XML logically associates the specific concrete operations of the Module Instance with the abstract Service operations. In this example, the "*Request_Val_Preferred*" module operation is connected to the "*Request_Value*" service operation of the "*Preferred_Request_Value_Service*" service instance and the "*Request_Val_Backup*" module operation is connected to the "*Request_Value*" service operation of the "*Backup_Request_Value_Service*" service instance.

The *<dataLink>* XML logically associates the specific concrete operations of the Module Instance with the abstract Service operations. In this example, the "*PreferredAvailable*" module operation is connected to the "*Available*" service operation of the "*Preferred_Request_Value_Service*" service instance and the "*BackupAvailable*" module operation is connected to the "*Available*" service operation of the "*Backup_Request_Value_Service*" service instance.

In addition, the “*Request_for_Val*” module operation is connected to the “*Request_Value*” service operation of the “*Request_Value_Service_Provided*” and the “*BrokeredAvailable*” module operation is connected to the “*Available*” service operation of the “*Request_Value_Service_Provided*”.

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

The Preferred Server ASC

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

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

   <use library="example"/>

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


      <operations>

         <requestReceived name="Request_for_Val" maxConcurrentRequests="10">
            <input name="time" type="ECOA:global_time"/>
            <output name="val" type="example:value_type"/>
         </requestReceived>

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

         <eventReceived name="Switch_Availability">
         </eventReceived>

      </operations>

   </moduleType>


   <moduleImplementation name="PreferredServer_Module_Im" language="C"
moduleType="PreferredServer_Module_Type"/>

   <moduleInstance name="PreferredServer_Module_Instance"
implementationName="PreferredServer_Module_Im" relativePriority="1">

   </moduleInstance>

   <triggerInstance name="PreferredServerTrigger" relativePriority="3"/>


   <requestLink>
```

```
      <clients>
         <service instanceName="Provide_Value_Service"
operationName="Request_Value"/>
      </clients>
      <server>
         <moduleInstance instanceName="PreferredServer_Module_Instance"
operationName="Request_for_Val"/>
      </server>
   </requestLink>

   <dataLink>
      <writers>
         <moduleInstance instanceName="PreferredServer_Module_Instance"
operationName="Available"/>
      </writers>
      <readers>
         <service instanceName="Provide_Value_Service" operationName="Available"/>
      </readers>
   </dataLink>

   <eventLink>
      <senders>
         <trigger instanceName="PreferredServerTrigger" period="5"/>
      </senders>
      <receivers>
         <moduleInstance instanceName="PreferredServer_Module_Instance"
operationName="Switch_Availability"/>
      </receivers>
   </eventLink>


</componentImplementation>
```

That is, a Module Type (`PreferredServer_Module_Type`) is declared which has three operations:

- A `requestReceived` operation "`Request_for_Val`";
- A `dataWritten` operation "`Available`";
- An `eventReceived` operation "`Switch_Availability`".

This Module Type is implemented by a concrete Module Implementation `PreferredServer_Module_Im` which in turn is instantiated once as the Module Instance `PreferredServer_Module_Instance`.

The `PreferredServerTrigger` Trigger Instance is introduced in order for the Preferred Server to periodically switch its provided service between the available and unavailable state. Once every period (5 seconds as set in the `<eventLink>` XML) the Trigger will fire and the Module Operation `Switch_Availability` will be invoked.

The *<requestLink>* XML logically associates the specific concrete operations of the Module Instance with the abstract Service operations. In this example, the “*Request_for_Val*” module operation is connected to the “*Request_Value*” service operation of the “*Provide_Value_Service*” service instance.

The *<dataLink>* XML logically associates the specific concrete operations of the Module Instance with the abstract Service operations. In this example, the “*Available*” module operation is connected to the “*Available*” service operation of the “*Provide_Value_Service*” service instance.

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

The Backup Server ASC

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

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

   <use library="example"/>

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


      <operations>

         <requestReceived name="Request_for_Val" maxConcurrentRequests="10">
            <input name="time" type="ECOA:global_time"/>
            <output name="val" type="example:value_type"/>
         </requestReceived>

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

      </operations>

   </moduleType>


   <moduleImplementation name="BackupServer_Module_Im" language="C"
moduleType="BackupServer_Module_Type"/>

   <moduleInstance name="BackupServer_Module_Instance"
implementationName="BackupServer_Module_Im" relativePriority="0">

   </moduleInstance>
   <requestLink>

      <clients>
         <service instanceName="Provide_Value_Service"
operationName="Request_Value"/>
      </clients>
      <server>
         <moduleInstance instanceName="BackupServer_Module_Instance"
operationName="Request_for_Val"/>
      </server>
   </requestLink>

   <dataLink>
      <writers>
         <moduleInstance instanceName="BackupServer_Module_Instance"
operationName="Available"/>
      </writers>
      <readers>
         <service instanceName="Provide_Value_Service" operationName="Available"/>
      </readers>
   </dataLink>


</componentImplementation>
```

That is, a Module Type (*BackupServer_Module_Type*) is declared which has three operations:

- A *requestReceived* operation "*Request_for_Val*";
- A *dataWritten* operation "*Available*";

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

The *<requestLink>* XML logically associates the specific concrete operations of the Module Instance with the abstract Service operations. In this example, the "*Request_for_Val*" module operation is connected to the "*Request_Value*" service operation of the "*Provide_Value_Service*" service instance.

The *<dataLink>* XML logically associates the specific concrete operations of the Module Instance with the abstract Service operations. In this example, the "*Available*" module operation is connected to the "*Available*" service operation of the "*Provide_Value_Service*" service instance.

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

ECOA Deployment Definition

The ECOA "*Wire Switch 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="BrokeredClient_Inst"
moduleInstanceName="BrokeredClient_Module_Instance" modulePriority="11"/>
      <deployedTriggerInstance componentName="BrokeredClient_Inst"
triggerInstanceName="Internal_Trigger_Instance" triggerPriority="12"/>
      <deployedModuleInstance componentName="PreferredServer_Inst"
moduleInstanceName="PreferredServer_Module_Instance" modulePriority="11"/>
      <deployedModuleInstance componentName="NonBrokeredClient_Inst"
moduleInstanceName="NonBrokeredClient_Module_Instance" modulePriority="11"/>
      <deployedTriggerInstance componentName="NonBrokeredClient_Inst"
triggerInstanceName="Internal_Trigger_Instance" triggerPriority="12"/>
      <deployedModuleInstance componentName="BackupServer_Inst"
moduleInstanceName="BackupServer_Module_Instance" modulePriority="11"/>
      <deployedModuleInstance componentName="ServiceBroker_Inst"
moduleInstanceName="ServiceBroker_Module_Instance" modulePriority="11"/>
      <deployedTriggerInstance componentName="PreferredServer_Inst"
triggerInstanceName="PreferredServerTrigger" triggerPriority="0"/>
   </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 Brokered Client ASC

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

```
void BrokeredClient_Module_Im__tick__received(BrokeredClient_Module_Im__context
*context)
{
   BrokeredClient_Module_Im_container__Available_handle availableHandle;
   // First check the service is available.
   ECOA__return_status return_status =
BrokeredClient_Module_Im_container__Available__get_read_access(context,
&availableHandle);

   if (return_status == ECOA__return_status_OK)
   {
      if (*(availableHandle.data) == ECOA__TRUE)
      {
         ECOA__global_time time;
         example__value_type val;
         ECOA__log log;

         return_status =
BrokeredClient_Module_Im_container__get_absolute_system_time(context, &time);

         val = 0;

         log.current_size = sprintf((char *) &log.data, "BrokeredClient - val
before request = %d", val);
         BrokeredClient_Module_Im_container__log_info(context, log);

         return_status =
BrokeredClient_Module_Im_container__Request_Val__request_sync(context, &time,
&val);

         log.current_size = sprintf((char *) &log.data, "BrokeredClient - val from
response = %d", val);
         BrokeredClient_Module_Im_container__log_info(context, log);
      }

      return_status =
BrokeredClient_Module_Im_container__Available__release_read_access(context,
&availableHandle);
   }

}
```

That is, the availability of the required service is interrogated. If it is available, the *Client_Module_Im_container__Request_Val__request_sync* operation is invoked. A log is performed prior to invoking the operation and after the operation containing the value of "*val*".

The Non-Brokered Client ASC

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

```
void
NonBrokeredClient_Module_Im__tick__received(NonBrokeredClient_Module_Im__context
*context)
{
   ECOA__return_status return_status;
   ECOA__uint32 requestID;
   example__value_type val = 0;
   ECOA__global_time time;
   ECOA__log log;
   ECOA__boolean8 preferredAvailable = ECOA__FALSE;

   return_status =
NonBrokeredClient_Module_Im_container__get_absolute_system_time(context, &time);

   // Use the preferred service provide if available.
   NonBrokeredClient_Module_Im_container__PreferredAvailable_handle
preferredAvailableHandle;
   return_status =
NonBrokeredClient_Module_Im_container__PreferredAvailable__get_read_access(context
, &preferredAvailableHandle);

   if (return_status == ECOA__return_status_OK)
   {
      if (*(preferredAvailableHandle.data) == ECOA__TRUE)
      {
         preferredAvailable = ECOA__TRUE;
         log.current_size = sprintf((char *) &log.data, "***NonBrokeredClient -
using preferred -  val before request = %d", val);
         NonBrokeredClient_Module_Im_container__log_info(context, log);

         return_status =
NonBrokeredClient_Module_Im_container__Request_Val_Preferred__request_sync(context
, &time, &val);

         log.current_size = sprintf((char *) &log.data, "***NonBrokeredClient -
using preferred - val from response = %d", val);
         NonBrokeredClient_Module_Im_container__log_info(context, log);
      }

      return_status =
NonBrokeredClient_Module_Im_container__PreferredAvailable__release_read_access(con
text, &preferredAvailableHandle);
   }

   // otherwise if the backup is available, use that.
   if (!preferredAvailable)
   {
      NonBrokeredClient_Module_Im_container__BackupAvailable_handle
backupAvailableHandle;
      return_status =
NonBrokeredClient_Module_Im_container__BackupAvailable__get_read_access(context,
&backupAvailableHandle);

      if (return_status == ECOA__return_status_OK)
      {
         if (*(backupAvailableHandle.data) == ECOA__TRUE)
         {
            log.current_size = sprintf((char *) &log.data, "***NonBrokeredClient - 
using backup - val before request = %d", val);
            NonBrokeredClient_Module_Im_container__log_info(context, log);

            return_status = 
NonBrokeredClient_Module_Im_container__Request_Val_Backup__request_sync(context, 
&time, &val);

            log.current_size = sprintf((char *) &log.data, "***NonBrokeredClient - 
using backup - val from response = %d", val);
            NonBrokeredClient_Module_Im_container__log_info(context, log);
         }

         return_status = 
NonBrokeredClient_Module_Im_container__BackupAvailable__release_read_access(contex
t, &backupAvailableHandle);
      }
   }
}
```

That is, the availability of the preferred required service is interrogated. If it is available, the *NonBrokeredClient_Module_Im_container__Request_Val_Preferred__request_sync* operation is invoked. A log is performed prior to invoking the operation and after the operation containing the value of "*val*".

If the preferred Service is not available, the availability of the backup required service is interrogated. If it is available, the *NonBrokeredClient_Module_Im_container__Request_Val_Backup__request_sync* operation is invoked. A log is performed prior to invoking the operation and after the operation containing the value of "*val*".

The Service Broker ASC

The Service Broker ASC checks the availability of both the preferred and backup required service at start up. If either of the required services are available, the module sets its provided service as available. This is done in the START module operation:

```
void ServiceBroker_Module_Im__START__received(ServiceBroker_Module_Im__context 
*context)
{
   ServiceBroker_Module_Im_container__PreferredAvailable_handle 
preferredAvailableHandle;
   ServiceBroker_Module_Im_container__BackupAvailable_handle 
backupAvailableHandle;
   ECOA__return_status status;

   // Check if either the preferred or backup services are available.

   status = 
ServiceBroker_Module_Im_container__PreferredAvailable__get_read_access(context, 
&preferredAvailableHandle);
```

```
   if (status == ECOA__return_status_OK)
   {
      context->user.preferredAvailable = *(preferredAvailableHandle.data);

      status =
ServiceBroker_Module_Im_container__PreferredAvailable__release_read_access(context
, &preferredAvailableHandle);
   }

   status =
ServiceBroker_Module_Im_container__BackupAvailable__get_read_access(context,
&backupAvailableHandle);
   if (status == ECOA__return_status_OK)
   {
      context->user.backupAvailable = *(backupAvailableHandle.data);

      status =
ServiceBroker_Module_Im_container__BackupAvailable__release_read_access(context,
&backupAvailableHandle);
   }

   setProvidedServiceAvailable(context);
}
```

It makes use of the user function setProvidedServiceAvailable:

```
void setProvidedServiceAvailable(ServiceBroker_Module_Im__context *context)
{
   ServiceBroker_Module_Im_container__BrokeredAvailable_handle availableHandle;

   // Set the provided service available if either required service instance is
available.
   ECOA__return_status status =
ServiceBroker_Module_Im_container__BrokeredAvailable__get_write_access(context,
&availableHandle);

   if (status == ECOA__return_status_OK || status ==
ECOA__return_status_DATA_NOT_INITIALIZED)
   {
      if (context->user.preferredAvailable == ECOA__TRUE || context-
>user.backupAvailable == ECOA__TRUE)
      {
         *(availableHandle.data) = ECOA__TRUE;
      }
      else
      {
         *(availableHandle.data) = ECOA__FALSE;
      }

      ECOA__return_status status =
ServiceBroker_Module_Im_container__BrokeredAvailable__publish_write_access(context
, &availableHandle);
   }
}
```

The module also receives notifications if the service availability of either service is changed. Whenever a required service availability is changed, the availability of the provided service is updated:

```
void
ServiceBroker_Module_Im__PreferredAvailable__updated(ServiceBroker_Module_Im__cont
ext* context)
{
   ServiceBroker_Module_Im_container__PreferredAvailable_handle
preferredAvailableHandle;
   ECOA__return_status status;

   // Check if either the preferred or backup services are available.

   status =
ServiceBroker_Module_Im_container__PreferredAvailable__get_read_access(context,
&preferredAvailableHandle);
   if (status == ECOA__return_status_OK)
   {
      context->user.preferredAvailable = *(preferredAvailableHandle.data);

      status =
ServiceBroker_Module_Im_container__PreferredAvailable__release_read_access(context
, &preferredAvailableHandle);
   }

   setProvidedServiceAvailable(context);
}

void
ServiceBroker_Module_Im__BackupAvailable__updated(ServiceBroker_Module_Im__context
* context)
{
   ServiceBroker_Module_Im_container__BackupAvailable_handle
backupAvailableHandle;
   ECOA__return_status status;

   status =
ServiceBroker_Module_Im_container__BackupAvailable__get_read_access(context,
&backupAvailableHandle);
   if (status == ECOA__return_status_OK)
   {
      context->user.backupAvailable = *(backupAvailableHandle.data);

      status =
ServiceBroker_Module_Im_container__BackupAvailable__release_read_access(context,
&backupAvailableHandle);
   }

   setProvidedServiceAvailable(context);
}
```

The request handler is implemented in the Request_for_Val entry point. If the preferred server is available, the broker ASC "forwards" the request on to the preferred server. If only the backup server is available, the broker ASC "forwards" the request on to the backup server:

```
void ServiceBroker_Module_Im__Request_for_Val__request_received
   (ServiceBroker_Module_Im__context* context,
    const ECOA__uint32 ID,
    const ECOA__global_time* time)
{
   ECOA__return_status return_status;
   example__value_type val;
   ECOA__log log;

   // Forward the request to the preferred if it's available.
   if (context->user.preferredAvailable)
   {
      log.current_size = sprintf((char *) &log.data, "***ServiceBrokered - using 
preferred -  val before request = %d", val);
      ServiceBroker_Module_Im_container__log_info(context, log);

      return_status = 
ServiceBroker_Module_Im_container__Request_Val_Preferred__request_sync(context, 
time, &val);

      log.current_size = sprintf((char *) &log.data, "***ServiceBrokered - using 
backup - val from response = %d", val);
      ServiceBroker_Module_Im_container__log_info(context, log);
   }
   else if (context->user.backupAvailable)
   {
      // otherwise if the backup is available, use that.
      log.current_size = sprintf((char *) &log.data, "***ServiceBrokered - using 
backup -  val before request = %d", val);
      ServiceBroker_Module_Im_container__log_info(context, log);

      return_status = 
ServiceBroker_Module_Im_container__Request_Val_Backup__request_sync(context, time, 
&val);

      log.current_size = sprintf((char *) &log.data, "***ServiceBrokered - using 
backup - val from response = %d", val);
      ServiceBroker_Module_Im_container__log_info(context, log);
   }

   // Always respond.
   return_status = 
ServiceBroker_Module_Im_container__Request_for_Val__response_send(context, ID, 
val);
}
```

The Preferred Server ASC

The Preferred Server sets its provided service as available at start up. This is done in the START module lifecycle entry point:

```
void PreferredServer_Module_Im__START__received(PreferredServer_Module_Im__context
*context)
{
   // Set the service as functionally available.
   PreferredServer_Module_Im_container__Available_handle availableHandle;

   ECOA__return_status status =
PreferredServer_Module_Im_container__Available__get_write_access(context,
&availableHandle);

   if (status == ECOA__return_status_OK || status ==
ECOA__return_status_DATA_NOT_INITIALIZED)
   {
      *(availableHandle.data) = ECOA__TRUE;
      ECOA__return_status status =
PreferredServer_Module_Im_container__Available__publish_write_access(context,
&availableHandle);
   }
}
```

The request handler is implemented in the Request_for_Val entrypoint:

```
void PreferredServer_Module_Im__Request_for_Val__request_received
   (PreferredServer_Module_Im__context* context,
    const ECOA__uint32 ID,
    const ECOA__global_time* time)
{
   ECOA__return_status return_status;

   // Preferred server responds immediately.
   return_status =
PreferredServer_Module_Im_container__Request_for_Val__response_send(context, ID,
1);
}
```

This function replies to the request with a data value of *1* by invoking the ECOA Container API function *PreferredServer_Module_Im_container__Request_for_Val__response_send*.

In addition, the preferred server periodically changes the availability of the provided service. This is done in the operation attached to the trigger instance "*Switch_Availability*":

```
void
PreferredServer_Module_Im__Switch_Availability__received(PreferredServer_Module_Im
__context *context)
{
   // Set the service as functionally available.
   ECOA__log log;
   PreferredServer_Module_Im_container__Available_handle availableHandle;
   ECOA__return_status status =
PreferredServer_Module_Im_container__Available__get_write_access(context,
&availableHandle);

   if (status == ECOA__return_status_OK)
   {
      if (*(availableHandle.data) == ECOA__TRUE)
      {
         log.current_size = sprintf((char *) &log.data, "Preferred server setting
service UNAVAILABLE");
         PreferredServer_Module_Im_container__log_info(context, log);
         *(availableHandle.data) = ECOA__FALSE;
      }
      else
      {
         log.current_size = sprintf((char *) &log.data, "Preferred server setting
service AVAILABLE");
         PreferredServer_Module_Im_container__log_info(context, log);
         *(availableHandle.data) = ECOA__TRUE;
      }
      ECOA__return_status status =
PreferredServer_Module_Im_container__Available__publish_write_access(context,
&availableHandle);
   }
}
```

The Backup Server ASC

The Backup Server sets its provided service as available at start up. This is done in the START module lifecycle entry point:

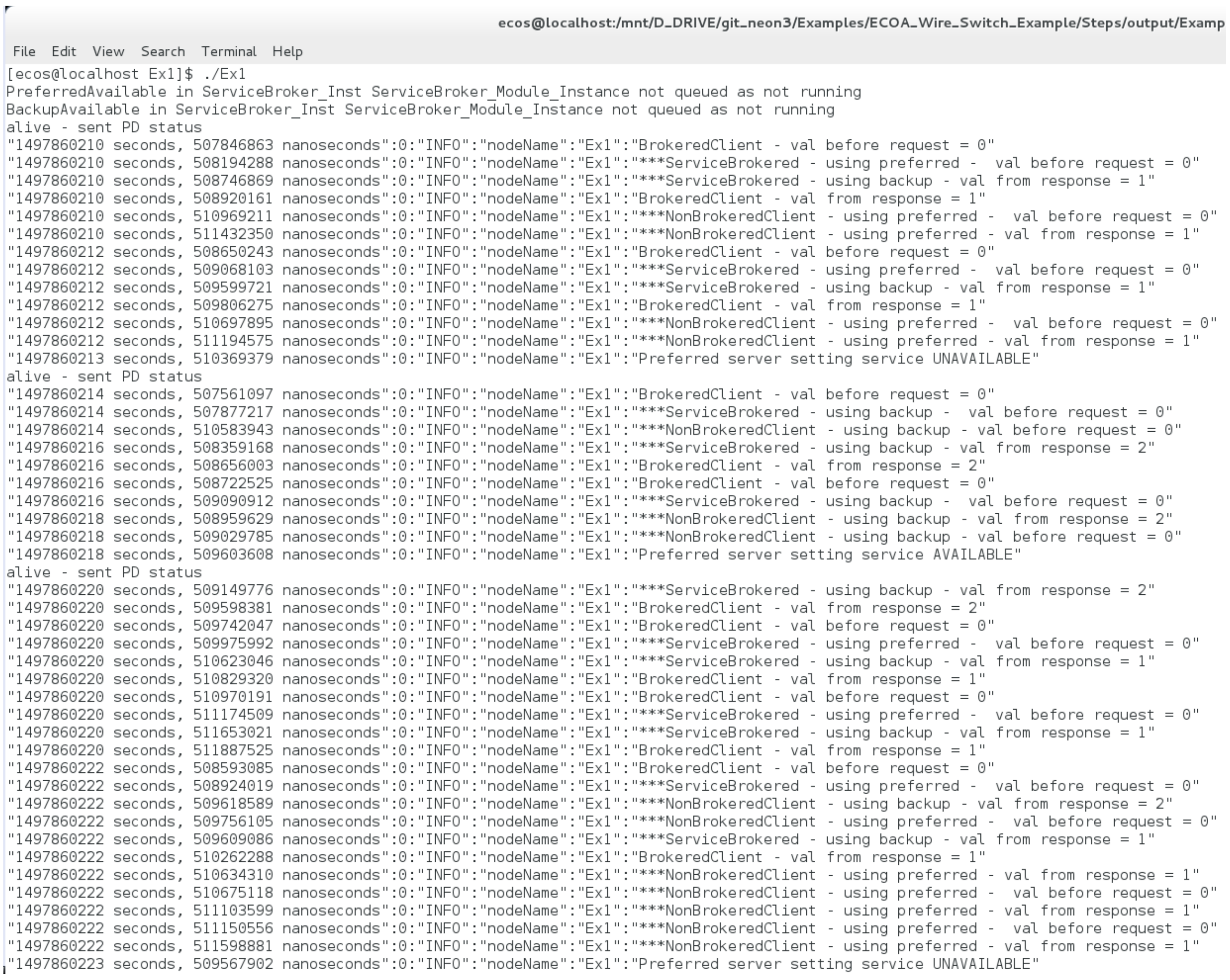
```
void BackupServer_Module_Im__START__received(BackupServer_Module_Im__context
*context)
{
   // Set the service as functionally available.
   BackupServer_Module_Im_container__Available_handle availableHandle;

   ECOA__return_status status =
BackupServer_Module_Im_container__Available__get_write_access(context,
&availableHandle);

   if (status == ECOA__return_status_OK || status ==
ECOA__return_status_DATA_NOT_INITIALIZED)
   {
      *(availableHandle.data) = ECOA__TRUE;
      ECOA__return_status status =
BackupServer_Module_Im_container__Available__publish_write_access(context,
&availableHandle);
   }
}

The request handler is implemented in the Request_for_Val entrypoint:
```

```
void BackupServer_Module_Im__Request_for_Val__request_received
   (BackupServer_Module_Im__context* context,
    const ECOA__uint32 ID,
    const ECOA__global_time* time)
{
   ECOA__return_status return_status;

   // Add an artificial delay in the backup server (to mimic a slower response
time in the non-preferred server).
   int i,j;
   for (i; i <= 500000000; i++)
   {
      while (j <= 500000000)
      {
         j++;
      }
   }

   return_status =
BackupServer_Module_Im_container__Request_for_Val__response_send(context, ID, 2);
}
```

This function replies to the request with a data value of *2* by invoking the ECOA Container API function *BackupServer_Module_Im_container__Request_for_Val__response_send*. An artificial delay has been added before the response to simulate a "worse response time" for the backup server.

Program Output

When the ECOA "*Wire Switch Example*" Assembly is built and run (in a single Node deployment), an output similar to Figure 12 should be achieved. Both *Client* ASCs output, at each iteration, the value before sending the request message, and the value after receiving the response. In addition, the preferred server logs when it sets the functional availability of its provided service. The value returned to the request is 1 when the request is handled by the preferred server, and 2 when the request is handled by the backup server.


ecos@localhost:/mnt/D_DRIVE/git_neon3/Examples/ECOA_Wire_Switch_Example/Steps/output/Examp
File Edit View Search Terminal Help

```
[ecos@localhost Ex1]$ ./Ex1
PreferredAvailable in ServiceBroker_Inst ServiceBroker_Module_Instance not queued as not running
BackupAvailable in ServiceBroker_Inst ServiceBroker_Module_Instance not queued as not running
alive - sent PD status
"1497860210 seconds, 507846863 nanoseconds":0:"INFO":"nodeName":"Ex1":"BrokeredClient - val before request = 0"
"1497860210 seconds, 508194288 nanoseconds":0:"INFO":"nodeName":"Ex1":"***ServiceBrokered - using preferred -  val before request = 0"
"1497860210 seconds, 508746869 nanoseconds":0:"INFO":"nodeName":"Ex1":"***ServiceBrokered - using backup - val from response = 1"
"1497860210 seconds, 508920161 nanoseconds":0:"INFO":"nodeName":"Ex1":"BrokeredClient - val from response = 1"
"1497860210 seconds, 510969211 nanoseconds":0:"INFO":"nodeName":"Ex1":"***NonBrokeredClient - using preferred -  val before request = 0"
"1497860210 seconds, 511432350 nanoseconds":0:"INFO":"nodeName":"Ex1":"***NonBrokeredClient - using preferred - val from response = 1"
"1497860212 seconds, 508650243 nanoseconds":0:"INFO":"nodeName":"Ex1":"BrokeredClient - val before request = 0"
"1497860212 seconds, 509068103 nanoseconds":0:"INFO":"nodeName":"Ex1":"***ServiceBrokered - using preferred -  val before request = 0"
"1497860212 seconds, 509599721 nanoseconds":0:"INFO":"nodeName":"Ex1":"***ServiceBrokered - using backup - val from response = 1"
"1497860212 seconds, 509806275 nanoseconds":0:"INFO":"nodeName":"Ex1":"BrokeredClient - val from response = 1"
"1497860212 seconds, 510697895 nanoseconds":0:"INFO":"nodeName":"Ex1":"***NonBrokeredClient - using preferred -  val before request = 0"
"1497860212 seconds, 511194575 nanoseconds":0:"INFO":"nodeName":"Ex1":"***NonBrokeredClient - using preferred - val from response = 1"
"1497860213 seconds, 510369379 nanoseconds":0:"INFO":"nodeName":"Ex1":"Preferred server setting service UNAVAILABLE"
alive - sent PD status
"1497860214 seconds, 507561097 nanoseconds":0:"INFO":"nodeName":"Ex1":"BrokeredClient - val before request = 0"
"1497860214 seconds, 507877217 nanoseconds":0:"INFO":"nodeName":"Ex1":"***ServiceBrokered - using backup -  val before request = 0"
"1497860214 seconds, 510583943 nanoseconds":0:"INFO":"nodeName":"Ex1":"***NonBrokeredClient - using backup - val before request = 0"
"1497860216 seconds, 508359168 nanoseconds":0:"INFO":"nodeName":"Ex1":"***ServiceBrokered - using backup - val from response = 2"
"1497860216 seconds, 508656003 nanoseconds":0:"INFO":"nodeName":"Ex1":"BrokeredClient - val from response = 2"
"1497860216 seconds, 508722525 nanoseconds":0:"INFO":"nodeName":"Ex1":"BrokeredClient - val before request = 0"
"1497860216 seconds, 509090912 nanoseconds":0:"INFO":"nodeName":"Ex1":"***ServiceBrokered - using backup -  val before request = 0"
"1497860218 seconds, 508959629 nanoseconds":0:"INFO":"nodeName":"Ex1":"***NonBrokeredClient - using backup - val from response = 2"
"1497860218 seconds, 509029785 nanoseconds":0:"INFO":"nodeName":"Ex1":"***NonBrokeredClient - using backup - val before request = 0"
"1497860218 seconds, 509603608 nanoseconds":0:"INFO":"nodeName":"Ex1":"Preferred server setting service AVAILABLE"
alive - sent PD status
"1497860220 seconds, 509149776 nanoseconds":0:"INFO":"nodeName":"Ex1":"***ServiceBrokered - using backup - val from response = 2"
"1497860220 seconds, 509598381 nanoseconds":0:"INFO":"nodeName":"Ex1":"BrokeredClient - val from response = 2"
"1497860220 seconds, 509742047 nanoseconds":0:"INFO":"nodeName":"Ex1":"BrokeredClient - val before request = 0"
"1497860220 seconds, 509975992 nanoseconds":0:"INFO":"nodeName":"Ex1":"***ServiceBrokered - using preferred -  val before request = 0"
"1497860220 seconds, 510623046 nanoseconds":0:"INFO":"nodeName":"Ex1":"***ServiceBrokered - using backup - val from response = 1"
"1497860220 seconds, 510829320 nanoseconds":0:"INFO":"nodeName":"Ex1":"BrokeredClient - val from response = 1"
"1497860220 seconds, 510970191 nanoseconds":0:"INFO":"nodeName":"Ex1":"BrokeredClient - val before request = 0"
"1497860220 seconds, 511174509 nanoseconds":0:"INFO":"nodeName":"Ex1":"***ServiceBrokered - using preferred -  val before request = 0"
"1497860220 seconds, 511653021 nanoseconds":0:"INFO":"nodeName":"Ex1":"***ServiceBrokered - using backup - val from response = 1"
"1497860220 seconds, 511887525 nanoseconds":0:"INFO":"nodeName":"Ex1":"BrokeredClient - val from response = 1"
"1497860222 seconds, 508593085 nanoseconds":0:"INFO":"nodeName":"Ex1":"BrokeredClient - val before request = 0"
"1497860222 seconds, 508924019 nanoseconds":0:"INFO":"nodeName":"Ex1":"***ServiceBrokered - using preferred -  val before request = 0"
"1497860222 seconds, 509618589 nanoseconds":0:"INFO":"nodeName":"Ex1":"***NonBrokeredClient - using backup - val from response = 2"
"1497860222 seconds, 509756105 nanoseconds":0:"INFO":"nodeName":"Ex1":"***NonBrokeredClient - using preferred -  val before request = 0"
"1497860222 seconds, 509609086 nanoseconds":0:"INFO":"nodeName":"Ex1":"***ServiceBrokered - using backup - val from response = 1"
"1497860222 seconds, 510262288 nanoseconds":0:"INFO":"nodeName":"Ex1":"BrokeredClient - val from response = 1"
"1497860222 seconds, 510634310 nanoseconds":0:"INFO":"nodeName":"Ex1":"***NonBrokeredClient - using preferred - val from response = 1"
"1497860222 seconds, 510675118 nanoseconds":0:"INFO":"nodeName":"Ex1":"***NonBrokeredClient - using preferred -  val before request = 0"
"1497860222 seconds, 511103599 nanoseconds":0:"INFO":"nodeName":"Ex1":"***NonBrokeredClient - using preferred - val from response = 1"
"1497860222 seconds, 511150556 nanoseconds":0:"INFO":"nodeName":"Ex1":"***NonBrokeredClient - using preferred -  val before request = 0"
"1497860222 seconds, 511598881 nanoseconds":0:"INFO":"nodeName":"Ex1":"***NonBrokeredClient - using preferred - val from response = 1"
"1497860223 seconds, 509567902 nanoseconds":0:"INFO":"nodeName":"Ex1":"Preferred server setting service UNAVAILABLE"
```

Figure 12 - ECOA "*Wire Switch 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	Simple Example *http://www.ecoa.technology/tutorials.html*