

Dining Philosophers

Introduction

This document describes an ECOA® implementation example of the famous “Dining Philosophers” problem (ref.[2]).

“Dining Philosophers” is an often used example in concurrent programming design, addressing resource contention and synchronization issues.

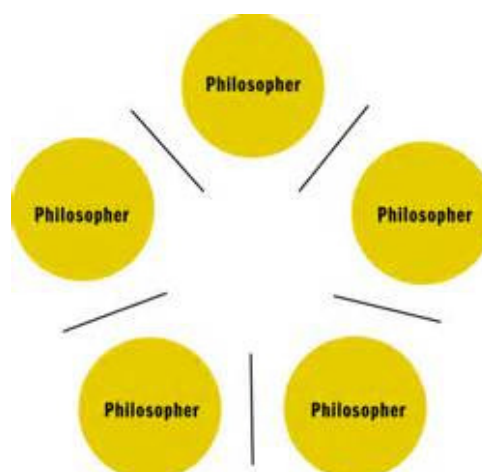
Five silent¹ philosophers sit at a round table with bowls of noodles. A chopstick is placed between each pair of adjacent philosophers.

Each philosopher must alternately think and eat. However, a philosopher can only eat noodles when he has both “left” and “right” chopsticks. Each chopstick can be held by only one philosopher and so a philosopher can use the chopstick only if it is not being used by another philosopher. After he finishes eating, he needs to put down both chopsticks so they become available to others. A philosopher can take the chopstick on his right or the one on his left as they become available, but cannot start eating before getting both of them. Nor can he take a chopstick that is not immediately on his left or right.

Eating is not limited by the remaining amounts of noodle or stomach space; an infinite supply is assumed!

The problem is how to design a discipline of behaviour (a *concurrent algorithm*) such that no philosopher will starve; i.e. each can forever continue to alternate between eating and thinking, assuming that no philosopher can know when others may want to eat or think.

Figure 1 The Dining Philosophers



¹ The philosophers cannot communicate with each other.

ECO A Examples: *Dining Philosophers*

This document presents the principal user generated artefacts required to create a “Dining Philosophers” 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 “*Dining Philosophers*” example is intended to demonstrate how ECOA concepts of concurrency and inversion of control (see ref.[1]) ease and facilitate the design and implementation of multi-threaded, multi-processing, applications.

ECO A Features Exhibited

- Composition of an ECOA Assembly of multiple ECOA ASCs (components).
- Contention-free resource sharing within an ECOA Assembly.
- Multiple cooperating ECOA Protection Domains.
- Service Availability.
- Use of the ECOA runtime logging API.

Design and Definition

Resource Hierarchy Solution

The method of solving the resource contention issue in the Dining Philosophers problem, i.e. how to ensure that no philosopher starves because he cannot get both chopsticks at once, is Dijkstra’s original “resource hierarchy” solution.

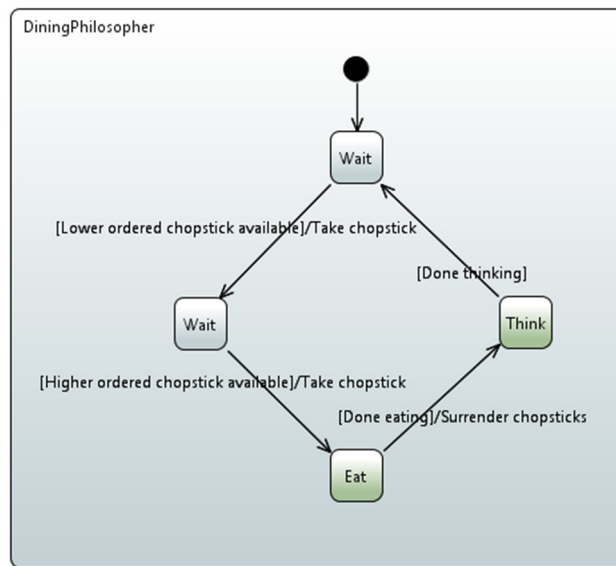
Each chopstick is assigned a “partial order” value (“0” to “4”) (with no duplication) so each philosopher has a “lower ordered” chopstick on one side and a “higher ordered” chopstick on the other side. The solution derives from imposing the rule that the lower ordered chopstick must be picked up first.

So after a philosopher has finished thinking, rather than pick up the first chopstick to become available (of those on his left or right), he must wait until the lower ordered of those chopsticks is available, pick that up, then possibly wait again until the higher ordered chopstick becomes available.

Once he has both chopsticks, he eats, and when finished, surrenders each chopstick. He then thinks for a while, before getting hungry and starting over.

This solution is depicted (for one philosopher) in the UML Activity Diagram of Figure 2.

Figure 2 Resource Hierarchy Solution Applied to a Dining Philosopher

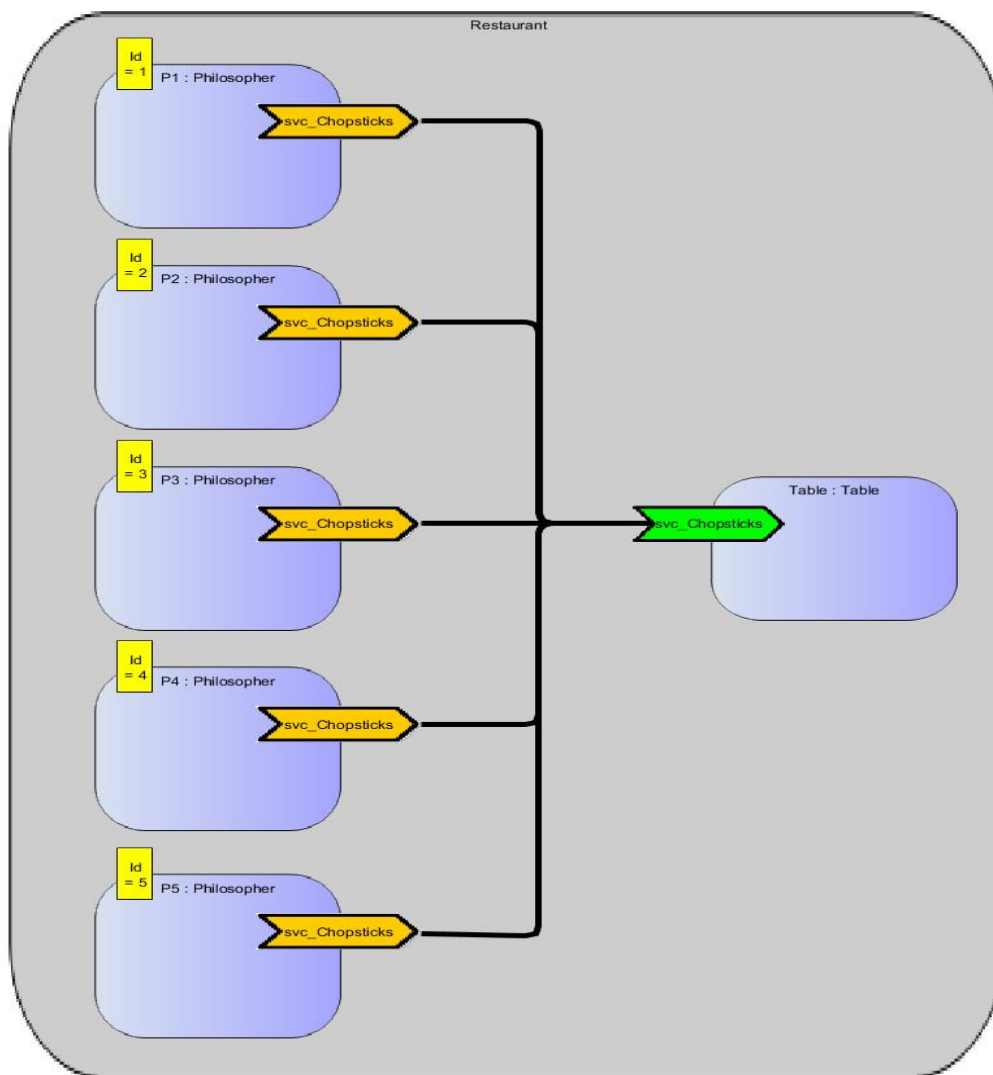


ECO A Assembly Design and Definition

This ECO A “*Dining Philosophers*” example is realized as an ECO A Assembly named “*Restaurant*” comprising five ECO A ASCs named “*P1*” to “*P5*” of the ASC type “*Philosopher*”, and one ASC named “*Table*” of ASC type “*Table*”. The “*Table*” ASC provides a “*svc_Chopsticks*” ECO A Service, which is referenced by the “*Philosopher*” ASCs, and by which each “*Philosopher*” can take and surrender chopsticks.

The ECO A “*Dining Philosophers*” (*Restaurant*) Assembly is depicted in Figure 3.

Figure 3 ECOA "Dining Philosophers" 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 "Dining Philosophers" (Restaurant) Assembly is as follows (file *Restaurant.impl.composite*), reflecting the Assembly diagram above:

```
<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="Restaurant">
  <csa:component name="Table">
    <ecoa-sca:instance componentType="Table">
      <ecoa-sca:implementation name="Table"/>
    </ecoa-sca:instance>
    <csa:service name="svc_Chopsticks"/>
  </csa:component>
```

This document is developed for and on behalf of BAE Systems (Operations) Limited, Dassault Aviation, Bull SAS, Thales Systèmes Aéroportés, GE Aviation Systems Limited, General Dynamics United Kingdom Limited and Leonardo MW Ltd, and the copyright is owned by BAE Systems (Operations) Limited, Dassault Aviation, Bull SAS, Thales Systèmes Aéroportés, GE Aviation Systems Limited, General Dynamics United Kingdom Limited and Leonardo MW Ltd. This document is developed by BAE Systems (Operations) Limited, Electronic Systems, and is the Intellectual Property of BAE Systems (Operations) Limited, Electronic Systems. The information set out in this document is provided solely on an 'as is' basis and the co-developers of this software make no warranties expressed or implied, including no warranties as to completeness, accuracy or fitness for purpose, with respect to any of the information.

```

<!-- -->
<csa:component name="P1">
  <ecoa-sca:instance componentType="Philosopher">
    <ecoa-sca:implementation name="Philosopher"/>
  </ecoa-sca:instance>
  <csa:reference name="svc_Chopsticks"/>
  <csa:property name="Id"><csa:value>1</csa:value></csa:property>
</csa:component>
:
: components P2 to P4 repeat
:
<csa:component name="P5">
  <ecoa-sca:instance componentType="Philosopher">
    <ecoa-sca:implementation name="Philosopher"/>
  </ecoa-sca:instance>
  <csa:reference name="svc_Chopsticks"/>
  <csa:property name="Id"><csa:value>5</csa:value></csa:property>
</csa:component>
<!-- -->
<!-- System Wiring... -->
<csa:wire source="P1/svc_Chopsticks" target="Table/svc_Chopsticks" ecoa-
sca:rank="1"/>
<csa:wire source="P2/svc_Chopsticks" target="Table/svc_Chopsticks" ecoa-
sca:rank="1"/>
<csa:wire source="P3/svc_Chopsticks" target="Table/svc_Chopsticks" ecoa-
sca:rank="1"/>
<csa:wire source="P4/svc_Chopsticks" target="Table/svc_Chopsticks" ecoa-
sca:rank="1"/>
<csa:wire source="P5/svc_Chopsticks" target="Table/svc_Chopsticks" ecoa-
sca:rank="1"/>
</csa:composite>

```

The *Table* ASC type is defined in XML as follows (file *Table.componcomponentType*):

```

<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="svc_Chopsticks">
    <ecoa-sca:interface syntax="svc_Chopsticks"/>
  </service>
</componentType>

```

That is, the ASC provides a single Service (*svc_Chopsticks*).

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

```

<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="svc_Chopsticks">
    <ecoa-sca:interface syntax="svc_Chopsticks"/>
  </reference>
  <property name="Id" type="xs:string" ecoa-sca:type="int32"/>
</componentType>

```

That is, in addition to declaring a *reference* to the *svc_Chopsticks* Service, the ASC defines an ECOA Property (*Id*) . Note that the Property *value* is given for each instance of the ASC in the *Restaurant* Assembly declaration (above), not here in the ASC type definition.

ECOA Service Definition

The *svc_Chopsticks* Service, which is provided by the *Table* ASC and referenced by the *Philosophers* ASCs, is defined in a XML file (*svc_Chopsticks.interface.xml*):

```
<serviceDefinition xmlns="http://www.ecoa.technology/interface-2.0"><!--
  name="svc_Chopsticks" -->
  <operations>
    <requestresponse name="take">
      <input name="which" type="int32"/>
      <input name="who" type="int32"/>
      <output name="taken" type="boolean8"/>
    </requestresponse>
    <requestresponse name="surrender">
      <input name="which" type="int32"/>
      <input name="who" type="int32"/>
    </requestresponse>
    <data name="ready" type="boolean8"/>
  </operations>
</serviceDefinition>
```

The Service comprise two ECOA Request-Response Operations, *take* and *surrender*, each of which takes two input parameters (*which* and *who*), whilst *take* also has a return parameter (*taken*). The parameter *which* specifies which chopstick the request is for, and is the partial order number for the chopstick ("0" to "4"). The *who* parameter specifies the philosopher's identity ("1" to "5") as given by its *Id* ECOA Property. The *taken* parameter will be *true* if the requested chopstick is available, or *false* if not.

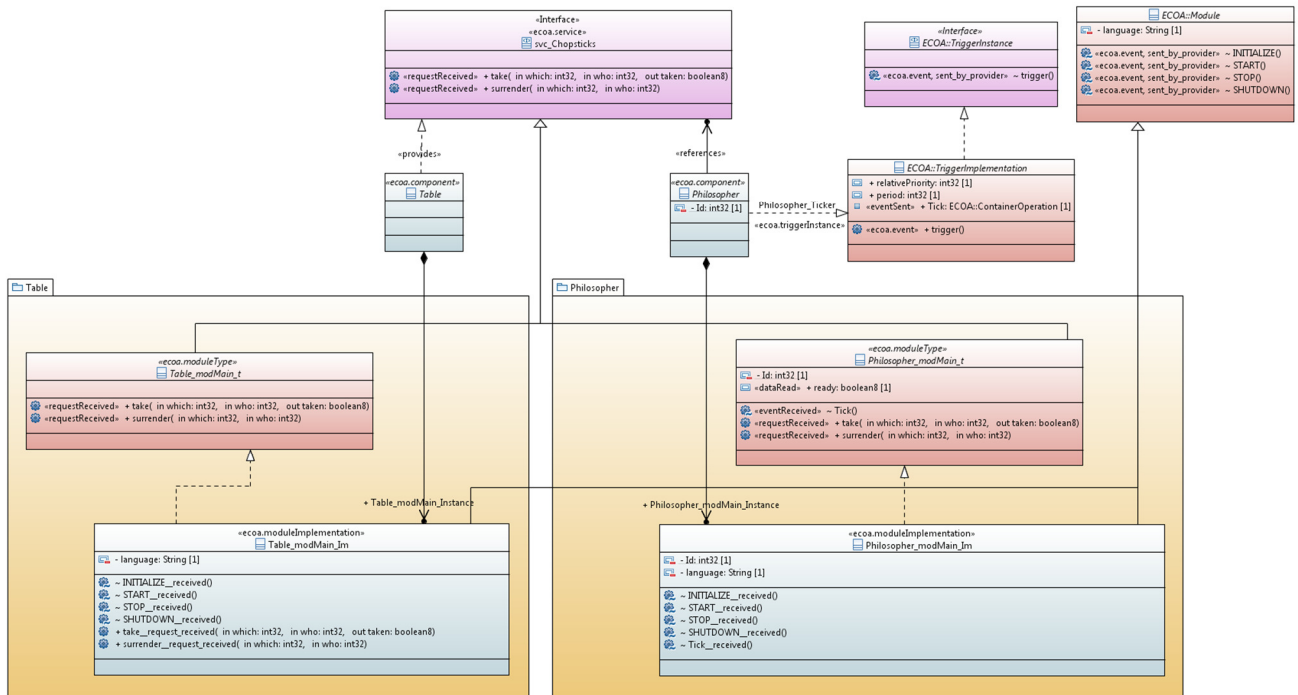
The Service also includes an ECOA Versioned-Data Operation, *ready*, which will be used to indicate to the Service clients (the Philosophers) that the Service is ready to accept operation requests – that is, to indicate "Service Availability", which will be discussed later.

Note that there is no mention or imposition at this declarative stage of the Resource Hierarchy algorithm, except to note that the chopsticks are identified by their numerical order value.

ECOA Module Design and Definition

The *Table* and *Philosopher* ASC (component) types are composed of a single ECOA Module each (Module Implementations *Table_modMain_Im* and *Philosopher_modMain_Im* of Module Types *Table_modMain_t* and *Philosopher_modMain_t* respectively) as illustrated in UML in Figure 4. Here is depicted in UML the *Table* ASC (component) **providing** the *svc_Chopsticks* ECOA Service, whilst the *Philosopher* ASC **references** the Service, and possesses the *Id* ECOA Property. As always in the ECOA, the Module Implementations implement the Module Lifecycle operations defined by the ECOA (as represented in UML by the abstract class *ECOA::Module*).

Figure 4 ECOA "Dining Philosophers" Module Design (as UML Class Diagram)



The Table ASC

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

```
<componentImplementation xmlns=http://www.ecoa.technology/implementation-2.0
  componentDefinition="Table">
  <use library="ECOA" />
  <!-- -->
  <moduleType name="Table_modMain_t" hasUserContext="true"
    hasWarmStartContext="false">
    <operations>
    <requestReceived name="take">
    <input name="which" type="int32" />
    <input name="who" type="int32"/>
    <output name="taken" type="boolean8"/>
    </requestReceived>
    <requestReceived name="surrender">
    <input name="which" type="int32" />
    <input name="who" type="int32"/>
    </requestReceived>
    </operations>
  </moduleType>
  <!-- -->
  <moduleImplementation name="Table_modMain_Im" moduleType="Table_modMain_t"
    language="C" />
  <!-- -->
  <moduleInstance name="Table_modMain_Instance"
    implementationName="Table_modMain_Im" relativePriority="1">
  </moduleInstance>
```

This document is developed for and on behalf of BAE Systems (Operations) Limited, Dassault Aviation, Bull SAS, Thales Systèmes Aéroportés, GE Aviation Systems Limited, General Dynamics United Kingdom Limited and Leonardo MW Ltd, and the copyright is owned by BAE Systems (Operations) Limited, Dassault Aviation, Bull SAS, Thales Systèmes Aéroportés, GE Aviation Systems Limited, General Dynamics United Kingdom Limited and Leonardo MW Ltd. This document is developed by BAE Systems (Operations) Limited, Electronic Systems, and is the Intellectual Property of BAE Systems (Operations) Limited, Electronic Systems. The information set out in this document is provided solely on an 'as is' basis and the co-developers of this software make no warranties expressed or implied, including no warranties as to completeness, accuracy or fitness for purpose, with respect to any of the information.

EOCA Examples: *Dining Philosophers*

```

<!-- -->
<requestLink>
  <clients>
    <service instanceName="svc_Chopsticks" operationName="take"/>
  </clients>
  <server>
    <moduleInstance instanceName="Table_modMain_Instance"
      operationName="take"/>
  </server>
</requestLink>
<!-- -->
<requestLink>
  <clients>
    <service instanceName="svc_Chopsticks" operationName="surrender"/>
  </clients>
  <server>
    <moduleInstance instanceName="Table_modMain_Instance"
      operationName="surrender"/>
  </server>
</requestLink>
/componentImplementation>

```

That is, a Module Type (*Table_modMain_t*) is declared which has two *requestReceived* operations, “take” and “surrender”, inherited from the *svc_Chopsticks* ECOA Service (depicted by the UML *generalization* association). This Module Type is implemented by a concrete Module Implementation *Table_modMain_Im* (depicted in the UML expanded in the form of the code class produced by the code generation process), which in turn is instantiated at runtime as the Module Instance *Table_modMain_Instance*.

The *<requestLink>* XML segments logically associate the specific concrete operations of the runtime 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 *Table_modMain_Im* class, and named “*Table_modMain_Im.c*” (assuming the Module Implementation declaration has set the *Language* property to “C”).

The Philosopher ASC

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

```

<componentImplementation xmlns="http://www.ecoa.technology/implementation-
  2.0" componentDefinition="Philosopher">
  <use library="ECOA" />
  <moduleType name="Philosopher_modMain_t" hasUserContext="true"
    hasWarmStartContext="false">
    <properties>
      <property name="Id" type="uint32"/>
    </properties>
    <operations>
      <eventReceived name="Tick" />
      <requestSent name="take" isSynchronous="true" timeout="-1.0">
        <input name="which" type="int32" />
        <input name="who" type="int32"/>
        <output name="taken" type="boolean8"/>
      </requestSent>
    </operations>
  </moduleType>
</componentImplementation>

```

This document is developed for and on behalf of BAE Systems (Operations) Limited, Dassault Aviation, Bull SAS, Thales Systèmes Aéroportés, GE Aviation Systems Limited, General Dynamics United Kingdom Limited and Leonardo MW Ltd, and the copyright is owned by BAE Systems (Operations) Limited, Dassault Aviation, Bull SAS, Thales Systèmes Aéroportés, GE Aviation Systems Limited, General Dynamics United Kingdom Limited and Leonardo MW Ltd. This document is developed by BAE Systems (Operations) Limited, Electronic Systems, and is the Intellectual Property of BAE Systems (Operations) Limited, Electronic Systems. The information set out in this document is provided solely on an ‘as is’ basis and the co-developers of this software make no warranties expressed or implied, including no warranties as to completeness, accuracy or fitness for purpose, with respect to any of the information.


```

        <requestSent name="surrender" isSynchronous="true"
                                timeout="-1.0">
            <input name="which" type="int32" />
            <input name="who" type="int32"/>
        </requestSent>
    </operations>
</moduleType>
<!-- -->
<moduleImplementation name="Philosopher_modMain_Im"
                        moduleType="Philosopher_modMain_t"
                        language="C" />

<!-- -->
<moduleInstance name="Philosopher_modMain_Instance"
                 implementationName="Philosopher_modMain_Im"
                 relativePriority="1">
    <propertyValues>
        <propertyValue name="Id">$Id</propertyValue>
    </propertyValues>
</moduleInstance>
<!-- -->
<triggerInstance name="Philosopher_Ticker" relativePriority="2"/>
<!-- -->
<requestLink>
    <clients>
        <moduleInstance instanceName="Philosopher_modMain_Instance"
                        operationName="take"/>
    </clients>
    <server>
        <reference instanceName="svc_Chopsticks" operationName="take"/>
    </server>
</requestLink>
<requestLink>
    <clients>
        <moduleInstance instanceName="Philosopher_modMain_Instance"
                        operationName="surrender"/>
    </clients>
    <server>
        <reference instanceName="svc_Chopsticks" operationName="surrender"/>
    </server>
</requestLink>
<eventLink>
    <senders>
        <trigger instanceName="Philosopher_Ticker" period="0.05" />
    </senders>
    <receivers>
        <moduleInstance instanceName="Philosopher_modMain_Instance"
                        operationName="Tick"/>
    </receivers>
</eventLink>
</componentImplementation>

```

That is, a Module Type (*Philosopher_modMain_t*) is declared which has two *requestSent* operations, “*take*” and “*surrender*”, inherited from the *svc_Chopsticks* ECOA Service (depicted by the UML *generalization* association), and an *eventReceived* operation named “*Tick*”. This Module Type is implemented by a concrete Module Implementation *Philosopher_modMain_Im* (depicted in the UML expanded in the form of the code class produced by the code generation

EOCA Examples: *Dining Philosophers*

process), which in turn is instantiated at runtime as the Module Instance *Philosopher_modMain_Instance*.

The *Philosopher_Ticker* Trigger Instance is introduced because a periodic iterative polling behaviour is required to implement the philosopher implementation state machine (of which more later). Once every period (0.05 seconds as set in the `<eventLink>` XML²) the Trigger will fire and the Module Operation *Tick* will be invoked.

The Service Link (`<requestLink>` and `<eventLink>`) XML segments logically associate the specific concrete operations of the runtime Module Instance with the abstract Service operations, or in the case of the “Tick” operation, associates the concrete Module Operation to the Trigger Instance operation.

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

EOCA Deployment Definition

The ECOA “*Dining Philosophers*” (*Restaurant*) Assembly is deployed (that is, the declared Module and Trigger Instances are allocated to ECOA Protection Domains, which are themselves allocated to computing nodes) by the following XML (file *Restaurant.deployment.xml*):

```
<deployment xmlns="http://www.ecoa.technology/deployment-2.0"
             finalAssembly="Restaurant" logicalSystem="hostbased">
  <protectionDomain name="Restaurant">
    <executeOn computingNode="cpu" computingPlatform="host"/>
    <deployedModuleInstance componentName="Table"
                          moduleInstanceName="Table_modMain_Instance"
                          modulePriority="50"/>
    <!-- -->
    <deployedModuleInstance componentName="P1"
                          moduleInstanceName="Philosopher_modMain_Instance"
                          modulePriority="50"/>
    <deployedTriggerInstance componentName="P1"
                          triggerInstanceName="Philosopher_Ticker"
                          triggerPriority="51"/>
    <!-- -->
    <deployedModuleInstance componentName="P2"
                          moduleInstanceName="Philosopher_modMain_Instance"
                          modulePriority="50"/>
    <deployedTriggerInstance componentName="P2"
                          triggerInstanceName="Philosopher_Ticker"
                          triggerPriority="51"/>
    <!-- -->
  </protectionDomain>
</deployment>
```

² The UML does not explicitly depict Service Links. The period attribute is therefore depicted as a UML property of the «*ecoa.triggerInstance*» UML interface class.

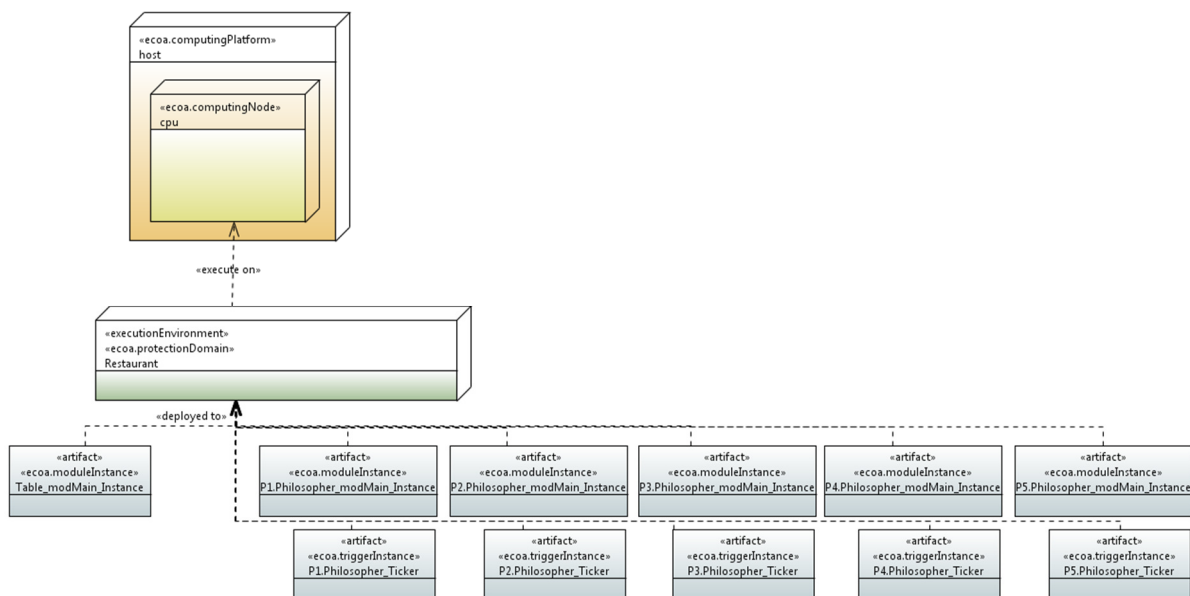
```

<deployedModuleInstance componentName="P3"
    moduleInstanceName="Philosopher_modMain_Instance"
    modulePriority="50"/>
<deployedTriggerInstance componentName="P3"
    triggerInstanceName="Philosopher_Ticker"
    triggerPriority="51"/>
<!-- -->
<deployedModuleInstance componentName="P4"
    moduleInstanceName="Philosopher_modMain_Instance"
    modulePriority="50"/>
<deployedTriggerInstance componentName="P4"
    triggerInstanceName="Philosopher_Ticker"
    triggerPriority="51"/>
<!-- -->
<deployedModuleInstance componentName="P5"
    moduleInstanceName="Philosopher_modMain_Instance"
    modulePriority="50"/>
<deployedTriggerInstance componentName="P5"
    triggerInstanceName="Philosopher_Ticker"
    triggerPriority="51"/>
</protectionDomain>
<platformConfiguration faultHandlerNotificationMaxNumber ="8"
    computingPlatform="host" />
</deployment>

```

Thus in this case, a single ECOA Protection Domain is declared (*Restaurant*) executing on ECOA Computing Node *cpu*, in ECOA Computing Platform *host* (as represented as a UML Deployment Diagram in Figure 5).

Figure 5 ECOA “Dining Philosophers” (Restaurant) Assembly Deployment



This document is developed for and on behalf of BAE Systems (Operations) Limited, Dassault Aviation, Bull SAS, Thales Systèmes Aéroportés, GE Aviation Systems Limited, General Dynamics United Kingdom Limited and Leonardo MW Ltd, and the copyright is owned by BAE Systems (Operations) Limited, Dassault Aviation, Bull SAS, Thales Systèmes Aéroportés, GE Aviation Systems Limited, General Dynamics United Kingdom Limited and Leonardo MW Ltd. This document is developed by BAE Systems (Operations) Limited, Electronic Systems, and is the Intellectual Property of BAE Systems (Operations) Limited, Electronic Systems. The information set out in this document is provided solely on an ‘as is’ basis and the co-developers of this software make no warranties expressed or implied, including no warranties as to completeness, accuracy or fitness for purpose, with respect to any of the information.

Service Availability Considerations³

Since the *Table* ASC provides an ECOA Service (*svc_Chopsticks*) it can be useful that the Service be declared (at runtime) as “available”. Clients of the Service can then check and take alternate action if the Service is not currently being provided. In the present simple example, availability of the *svc_Chopsticks* Service will be indicated using the *ready* ECOA Versioned Data item. *ready* will be set *true* (the Service is “available”) when the *Table* ASC’s Module (Module Implementation *Table_modMain_Im*) receives a **START** Event Operation. No error conditions are defined in this example, so once set the *svc_Chopsticks* Service will always be “available” and the *ready* Versioned Data item will always be *true*.

Implementation

In the implementation, philosophers are numerically identified “1” to “5”. Chopsticks are numerically identified “0” to “4” – the modulus-to-base-5 of which is the partial ordering required for the Resource Hierarchy Solution, i.e.:

$$“0” < “1” < “2” < “3” < “4” < “0”$$

The Table ASC

The *Table* ASC provides the *svc_Chopsticks* Service using a trivial chopstick allocation algorithm:

```
if the requestedChopstick is free
    allocate requestedChopstick to requestingPhilosopher
    return taken Boolean set true
otherwise
    return taken Boolean set false
```

There is no attempt to police whether a *requestingPhilosopher* “may” or “may not” request a particular chopstick. An error is raised if a philosopher tries to surrender a chopstick that is not allocated to him, including a chopstick that is already free.

All chopsticks are initially free, initialized by the *INITIALIZE* operation – implemented by the(C) code function *Table_modMain_Im__INITIALIZE__received* in the code unit *Table_modMain_Im.c*:

```
void Table_modMain_Im__INITIALIZE__received(
    Table_modMain_Im__context* context)
{
    int i;
    for( i = 0; i < 5; i++){
        context->user.Stick[i] = FREE;
    }
}
```

³ Service Availability was a concept and intrinsic functionality included in the ECOA up to issue 5 of the Architecture Specification. It allowed ECOA Service providers to signal to clients when the provider was active and able to accept Service Operation requests.

On invocation of the *START* operation, the *Table* ASC will set the *ready* Versioned Data item (Service Availability) for the *svc_Chopsticks* ECO A Service, and announce its presence. That *START* operation is implemented by the (C) code function *Table_modMain_Im_START_received* in the code unit *Table_modMain_Im.c*:

```
void Table_modMain_Im_START_received(
    Table_modMain_Im_context* context)
{
    ECOA__log msg;
    ECOA__return_status erc;
    //
    msg.current_size = sprintf( msg.data,
        "\n\tThe Table is laid, the chopsticks are available...\n" );
    Table_modMain_Im_container__log_info( context, msg );
    //
    Table_modMain_Im_container__ready__get_write_access( context, &rdyHndl );
    *(rdyHndl.data) = ECOA__TRUE;
    Table_modMain_Im_container__ready__publish_write_access( context, &rdyHndl );
}

```

The “take” Service Operation is handled by the code function *Table_modMain_Im_take_request_received*. The function checks if the requested chopstick (*which*) is free, and if so allocates it to the requesting philosopher (*who*). The taken output parameter is set *true* (not zero) or *false* (zero) respectively. The code function completes the request-response transaction by invoking the *Table_modMain_Im_container_take_response_send* API function:

```
void Table_modMain_Im_take_request_received(
    Table_modMain_Im_context* context,
    const ECOA__uint32 ID,
    const ECOA__int32 which,
    const ECOA__int32 who)
{
    ECOA__boolean8 taken = 0;
    //
    if( context->user.Stick[which] == FREE ){
        taken = ECOA__TRUE;
        context->user.Stick[which] = who;
    }else{
        taken = ECOA__FALSE;
    }
    Table_modMain_Im_container_take_response_send( context, ID, taken );
}

```

The “surrender” Service Operation is handled by the code function *Table_modMain_Im_surrender_request_received*. The function checks if the nominated chopstick (*which*) is allocated to the philosopher (*who*). If it is, then the chopstick becomes free. Otherwise an error is signalled.

ECO A Examples: *Dining Philosophers*

```

void Table_modMain_Im_surrender_request_received( Table_modMain_Im_context*
context,
const ECOA_uint32 ID,
const ECOA_int32 which,
const ECOA_int32 who)
{
    if( context->user.Stick[which] != who ){
        errno = EPERM; perror( "surrender" );
        fprintf( stderr,
            "%d is trying to surrender chopstick %d held by %d...\n",
            who, which, context->user.Stick[which] );
        exit(4);
    }else{
        context->user.Stick[which] = FREE;
    }
    Table_modMain_Im_container_surrender_response_send( context, ID );
}

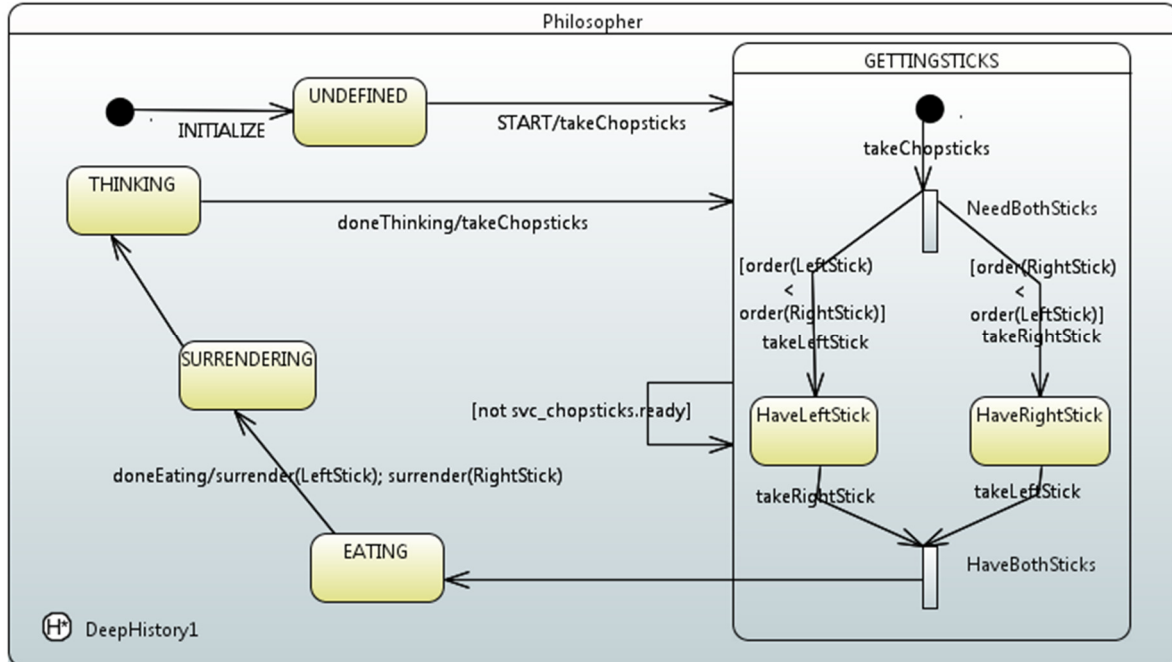
```

The request-response transaction is completed by the code function by invoking the *Table_modMain_Im_container_surrender_response_send* API function. Note that the request-response transaction must be completed even though, in this case, there is no response data.

The Philosopher ASC

The implementation behaviour State Machine of the *Philosopher* ASC, in order to meet the Resource Hierarchy Solution, is depicted in Figure 6.

Figure 6 ECOA “Dining Philosophers” Implementation State Machine



This document is developed for and on behalf of BAE Systems (Operations) Limited, Dassault Aviation, Bull SAS, Thales Systèmes Aéroportés, GE Aviation Systems Limited, General Dynamics United Kingdom Limited and Leonardo MW Ltd, and the copyright is owned by BAE Systems (Operations) Limited, Dassault Aviation, Bull SAS, Thales Systèmes Aéroportés, GE Aviation Systems Limited, General Dynamics United Kingdom Limited and Leonardo MW Ltd. This document is developed by BAE Systems (Operations) Limited, Electronic Systems, and is the Intellectual Property of BAE Systems (Operations) Limited, Electronic Systems. The information set out in this document is provided solely on an ‘as is’ basis and the co-developers of this software make no warranties expressed or implied, including no warranties as to completeness, accuracy or fitness for purpose, with respect to any of the information.

On initialization (receipt of the ECOA Module Lifecycle *INITIALIZE* operation) the philosopher's state is "UNDEFINED". When the ECOA Module Lifecycle *START* operation is received, the philosopher's state is set to "GETTINGSTICKS". From there on, the state machine is free-running, PROVIDED that the *ready* Versioned Data item is *true* (i.e. that the *svc_chopsticks* Service is available):

- when in the "GETTINGSTICKS" state, the *LeftStick* and *RightStick* are taken;
- the philosopher then enters the "EATING" state;
- when the philosopher has finished eating⁴ the chopsticks are surrendered;
- the philosopher then enters the "THINKING" state;
- when the philosopher has finished thinking⁵, he enters the "GETTINGSTICKS" state again.

The *INITIALIZE* Lifecycle operation is implemented by the code function *Philosopher_modMain_Im_INITIALIZE_received* of the (C) code unit *Philosopher_modMain_Im.c*; and simply initializes philosopher the state variables:

```
void Philosopher_modMain_Im_INITIALIZE_received
(Philosopher_modMain_Im_context* context)
{
    context->user.PhiloState = philosopher__State_UNDEFINED;
    context->user.EatUntil =
        context->user.ThinkUntil =
            (ECOA_hr_time){ 0, 0 };
    context->user.HaveLeftStick =
        context->user.HaveRightStick =
            ECOA_FALSE;
}
```

On invocation of the *START* operation, the *Philosopher* ASC will move to (set itself as being "in") the "GETTINGSTICKS" state, find out (by reading the *Id* ECOA Property value) its identity, and announce that it is ready to start eating and thinking. That *START* operation is implemented by the code function *Philosopher_modMain_Im_START_received*:

```
void Philosopher_modMain_Im_START_received
(Philosopher_modMain_Im_context* context)
{
    ECOA_uint32    IAm;
    ECOA_log      msg;
    //
    context->user.PhiloState = philosopher__State_GETTINGSTICKS;
    //
    Philosopher_modMain_Im_container__get_Id_value( context, &IAm );
    msg.current_size = sprintf( msg.data,
        "\n\tPhilosopher %d is ready...\n", IAm );
    Philosopher_modMain_Im_container__log_info( context, msg );
}
```

⁴ In this implementation the philosopher has finished eating after 7 seconds – he is VERY hungry!

⁵ In this implementation the philosopher has finished thinking after just 11 seconds.

ECOAs Examples: *Dining Philosophers*

We just now program the remaining (free-running) part of the philosopher state machine. In order to preserve ECOA Inversion of Control principals, the state machine is implemented by sampling the state periodically, triggered by the *Philosopher_Ticker* ECOA Trigger Instance, and implemented in the code function *Philosopher_modMain_Im_Tick_received* (as on the pages following):

```

void Philosopher_modMain_Im_Tick__received(Philosopher_modMain_Im_context* context)
{
    ECOA_uint32 IAm;
    ECOA_log msg;
    ECOA_return_status_erc;
    ECOA_hr_time timeNow;
    ECOA_boolean8 Taken, TableLaid;
    ECOA_int32 LeftStick, RightStick;
    Philosopher_modMain_Im_container__ready_handle rdyHndl;

    //
    Philosopher_modMain_Im_container__get_Id_value( context, &IAm );
    //
    LeftStick = IAm - 1; // IAm is {1..5} therefore this is {0..4}
    RightStick = IAm % 5; // IAm is {1..5} therefore this is {1..4,0}
    //
    Philosopher_modMain_Im_container__get_relative_local_time( context, &timeNow );
    msg.current_size = sprintf( msg.data, "Philosopher %d Tick. State = %s", IAm, philosopher__State_value( context-
        >user.PhiloState ) );
    //
    Philosopher_modMain_Im_container__log_info( context, msg );
    //
    Philosopher_modMain_Im_container__ready__get_read_access( context, &rdyHndl );
    TableLaid = *(rdyHndl.data);
    Philosopher_modMain_Im_container__ready__release_read_access( context, &rdyHndl );
    if( !TableLaid ){
        // No point in doing anything 'til the Chopsticks are available
        msg.current_size = sprintf( msg.data, "Philosopher %d Tick, but Chopsticks not available...", IAm );
        Philosopher_modMain_Im_container__log_info( context, msg );
        return;
    }
    Philosopher_modMain_Im_container__get_service_availability( context,
        Philosopher_modMain_Im_container__reference_id__svc_Chopsticks, &available);
    if( !TableLaid){
        // No point in doing anything 'til the Chopsticks are available
        msg.current_size = sprintf( msg.data, "Philosopher %d Tick, but Chopsticks not available...", IAm );
        Philosopher_modMain_Im_container__log_info( context, msg );
        return;
    }
    //
}

```

This document is developed for and on behalf of BAE Systems (Operations) Limited, Dassault Aviation, Bull SAS, Thales Systèmes Aéroportés, GE Aviation Systems Limited, General Dynamics United Kingdom Limited and Leonardo MW Ltd, and the copyright is owned by BAE Systems (Operations) Limited, Dassault Aviation, Bull SAS, Thales Systèmes Aéroportés, GE Aviation Systems Limited, General Dynamics United Kingdom Limited and Leonardo MW Ltd. This document is developed by BAE Systems (Operations) Limited, Electronic Systems, and is the Intellectual Property of BAE Systems (Operations) Limited, Electronic Systems. The information set out in this document is provided solely on an 'as is' basis and the co-developers of this software make no warranties expressed or implied, including no warranties as to completeness, accuracy or fitness for purpose, with respect to any of the information.

```

if( context->user.PhiloState == philosopher__State_GETTINGSTICKS ){
// *****
// * Dijkstra's Resource Hierarchy Solution... *
// *****
// The lower numbered chopstick is always taken first...
/**/
if( RightStick < LeftStick ){
    if( !context->user.HaveRightStick ){
        if(( erc = Philosopher_modMain_Im_container__take__request_sync( context, RightStick, IAM, &Taken ))
            != ECOA__return_status_OK ){
            msg.current_size = sprintf( msg.data, "take__request( RightStick=>%d, by=>%d ) failed with %d",
                RightStick, IAM, erc );
            Philosopher_modMain_Im_container__log_info( context, msg );
        }
    }
    if( !Taken ){
        return; // We'll have another go on the next tick...
    }
    }else{
        context->user.HaveRightStick = ECOA__TRUE;
    }
}
}
if( !context->user.HaveLeftStick ){
    if(( erc = Philosopher_modMain_Im_container__take__request_sync( context, LeftStick, IAM, &Taken )) !=
        ECOA__return_status_OK ){
        msg.current_size = sprintf( msg.data, "take__request( LeftStick=>%d, by=>%d ) failed with %d",
            LeftStick, IAM, erc );
        Philosopher_modMain_Im_container__log_info( context, msg );
    }
    if( !Taken ){
        return; // We'll have another go on the next tick...
    }
    }else{
        context->user.HaveLeftStick = ECOA__TRUE;
    }
}
if( RightStick > LeftStick ){
    if( !context->user.HaveRightStick ){
        if(( erc = Philosopher_modMain_Im_container__take__request_sync( context, RightStick, IAM, &Taken ))
            != ECOA__return_status_OK ){
            msg.current_size = sprintf( msg.data, "take__request( RightStick=>%d, by=>%d ) failed with %d",
                RightStick, IAM, erc );
            Philosopher_modMain_Im_container__log_info( context, msg );
        }
    }
}
}

```

This document is developed for and on behalf of BAE Systems (Operations) Limited, Dassault Aviation, Bull SAS, Thales Systèmes Aéroportés, GE Aviation Systems Limited, General Dynamics United Kingdom Limited and Leonardo MW Ltd, and the copyright is owned by BAE Systems (Operations) Limited, Dassault Aviation, Bull SAS, Thales Systèmes Aéroportés, GE Aviation Systems Limited, General Dynamics United Kingdom Limited and Leonardo MW Ltd. This document is developed by BAE Systems (Operations) Limited, Electronic Systems, and is the Intellectual Property of BAE Systems (Operations) Limited, Electronic Systems. The information set out in this document is provided solely on an 'as is' basis and the co-developers of this software make no warranties expressed or implied, including no warranties as to completeness, accuracy or fitness for purpose, with respect to any of the information.

```

if( !Taken ){
    return; // We'll have another go on the next tick...
}else{
    context->user.HaveRightStick = ECOA__TRUE;
}
}
}
printf( "%d begin eating.\n", IAm );
context->user.PhiloState = philosopher__State_EATING;
context->user.EatUntil = timeAdd( timeNow, EatPeriod );
}
//
if( context->user.PhiloState == philosopher__State_EATING ){
    if( timecmp( timeNow, context->user.EatUntil ) < 0 ){
        return; // Still chomping...
    }else{
        printf( "%d finished eating.\n", IAm );
        context->user.PhiloState = philosopher__State_SURRENDERING;
    }
}
//

```

This document is developed for and on behalf of BAE Systems (Operations) Limited, Dassault Aviation, Bull SAS, Thales Systèmes Aéroportés, GE Aviation Systems Limited, General Dynamics United Kingdom Limited and Leonardo MW Ltd, and the copyright is owned by BAE Systems (Operations) Limited, Dassault Aviation, Bull SAS, Thales Systèmes Aéroportés, GE Aviation Systems Limited, General Dynamics United Kingdom Limited and Leonardo MW Ltd. This document is developed by BAE Systems (Operations) Limited, Electronic Systems, and is the Intellectual Property of BAE Systems (Operations) Limited, Electronic Systems. The information set out in this document is provided solely on an 'as is' basis and the co-developers of this software make no warranties expressed or implied, including no warranties as to completeness, accuracy or fitness for purpose, with respect to any of the information.

```

if( context->user.PhiloState == philosopher__State_SURRENDERING ){
    if(( ERC = Philosopher_modMain_Im_container__surrender__request_sync( context, LeftStick, IAM )) !=
        ECOA__return_status_OK ){
        msg.current_size = sprintf( msg.data, "\n\tPhilosopher %d failed to surrender chopstick %d...\n", IAM,
            LeftStick );
        Philosopher_modMain_Im_container__log_warning( context, msg );
    }
    if(( ERC = Philosopher_modMain_Im_container__surrender__request_sync( context, RightStick, IAM )) !=
        ECOA__return_status_OK ){
        msg.current_size = sprintf( msg.data, "\n\tPhilosopher %d failed to surrender chopstick %d...\n", IAM,
            RightStick );
        Philosopher_modMain_Im_container__log_warning( context, msg );
    }
    context->user.HaveLeftStick = context->user.HaveRightStick = ECOA__FALSE;
    printf( "%d begin thinking.\n", IAM );
    context->user.PhiloState = philosopher__State_THINKING;
    context->user.ThinkUntil = timeAdd( timeNow, ThinkPeriod );
}
//
if( context->user.PhiloState == philosopher__State_THINKING ){
    if( timecmp( timeNow, context->user.ThinkUntil ) < 0 ){
        return; // Still cogitating...
    }
    printf( "%d finished thinking.\n", IAM );
    context->user.PhiloState = philosopher__State_GETTINGSTICKS;
}
//
if( context->user.PhiloState <= philosopher__State_UNDEFINED ||
    context->user.PhiloState > philosopher__State_THINKING ){
    msg.current_size = sprintf( msg.data, "\n\tPhilosopher %d has illegal state %s...\n", IAM, philosopher__State_value(
        context->user.PhiloState ) );
    Philosopher_modMain_Im_container__log_info( context, msg );
    return;
}
}
}

```

This document is developed for and on behalf of BAE Systems (Operations) Limited, Dassault Aviation, Bull SAS, Thales Systèmes Aéroportés, GE Aviation Systems Limited, General Dynamics United Kingdom Limited and Leonardo MW Ltd, and the copyright is owned by BAE Systems (Operations) Limited, Dassault Aviation, Bull SAS, Thales Systèmes Aéroportés, GE Aviation Systems Limited, General Dynamics United Kingdom Limited and Leonardo MW Ltd. This document is developed by BAE Systems (Operations) Limited, Electronic Systems, and is the Intellectual Property of BAE Systems (Operations) Limited, Electronic Systems. The information set out in this document is provided solely on an 'as is' basis and the co-developers of this software make no warranties expressed or implied, including no warranties as to completeness, accuracy or fitness for purpose, with respect to any of the information.

The code function *Philosopher_modMain_Im_Tick_received* requires two constant values that are used to determine when the philosopher has had enough eating or thinking.

```
static const ECOA_hr_time EatPeriod = { 7, 0 };
static const ECOA_hr_time ThinkPeriod = { 11, 0 };
```

In each case, the function records the time that the state is entered (“EATING” or “THINKING”) and each time it is triggered it checks the current time against the recorded time. Only when the period set by the constant has passed will the state machine progress (either to the “SURRENDERING” or “GETTINGSTICKS” states of Figure 6).

Program Output

When the ECOA “Dining Philosophers” (Restaurant) Assembly is built and run, an output similar to Figure 7 should be achieved. The *PhiLosopher* ASC start-up messages are output to the system console, prefixed by miscellaneous logging data (time stamp, logging type, etc.) (one of which is shown from “Philosopher 5” (i.e. ASC *P5* of the Assembly)). Each philosopher then outputs state changes, reporting his *Id* number and whether he is beginning or finishing eating or thinking. These outputs are interleaved with any other ECOA Platform logging messages (such as the 10 second periodic “alive” messages in the example shown):

Figure 7 ECOA “Dining Philosophers” (Restaurant) Assembly in Execution

```
D:\Projects\ECOA\Samples\Samples\Philosophers\Steps\output\host\cpu\Restaurant\Restaurant.e...
1506406721 seconds, 531613400 nanoseconds:0:INFO:cpu:Restaurant:
  Philosopher 5 is ready...

$4_Restaurant~Level1/1506406721 532613400:alive - sent PD status
1 begin eating.
3 begin eating.
$4_Restaurant~Level1/1506406726 534613400:alive - sent PD status
1 finished eating.
1 begin thinking.
3 finished eating.
3 begin thinking.
5 begin eating.
2 begin eating.
$4_Restaurant~Level1/1506406731 535613400:alive - sent PD status
5 finished eating.
5 begin thinking.
2 finished eating.
2 begin thinking.
4 begin eating.
$4_Restaurant~Level1/1506406736 536613400:alive - sent PD status
1 finished thinking.
1 begin eating.
3 finished thinking.
$4_Restaurant~Level1/1506406741 537613400:alive - sent PD status
4 finished eating.
```

References

1	European Component Oriented Architecture (ECO A) Collaboration Programme: Architecture Specification (Parts 1 to 11) "ECO A" is a registered trade mark.
2	Dining philosophers problem Dijkstra, Hoare https://en.wikipedia.org/wiki/Dining_philosophers_problem