Ping pong example

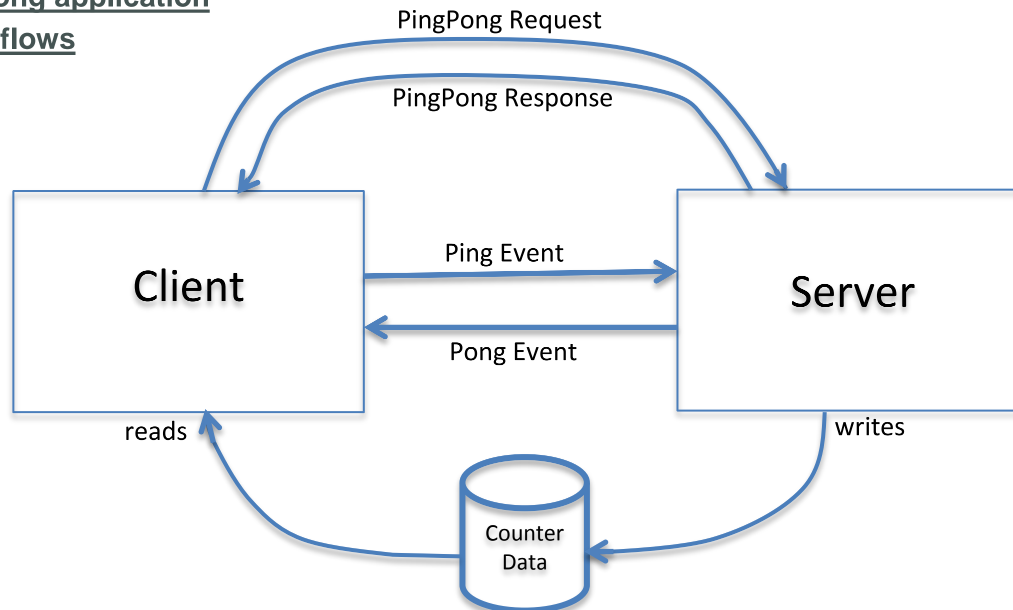
Basic example to introduce ECOA concepts – 2017-11-21

Introduction

This presentation shows how to create, step by step, a basic example using some ECOA concepts :

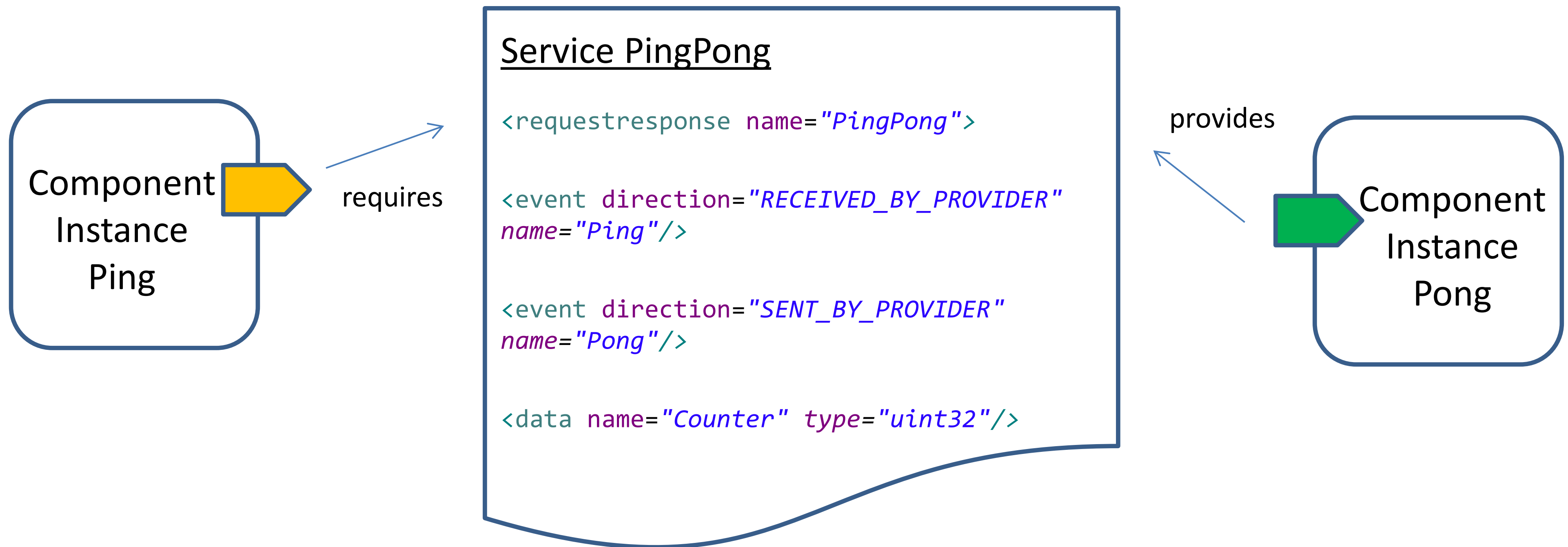
- **Components** (ASC) as functional « bricks » to build an application,
- **Services** that are provided or required by components, and which are composed of elementary operations (three kinds of operations used in the example : **RequestResponse**, **Event**, **Data**),
- **Modules** that implement components as technical monothreaded sequences of treatments,
- Different levels of assembly schemas (**composites**) to define system architectures, or internal component architectures,
- **Deployment** of modules onto platform resources.

Overview of the PingPong application with its functional dataflows



This ECOA tutorial represents the output of a research programme and is provided solely on an ‘as is’ basis and co-authors of this tutorial make no warranties expressed or implied, including no warranties as to completeness, accuracy or fitness for purpose, with respect to any of the information.

Pingpong example : ECOA view



This ECOA tutorial represents the output of a research programme and is provided solely on an 'as is' basis and co-authors of this tutorial make no warranties expressed or implied, including no warranties as to completeness, accuracy or fitness for purpose, with respect to any of the information.

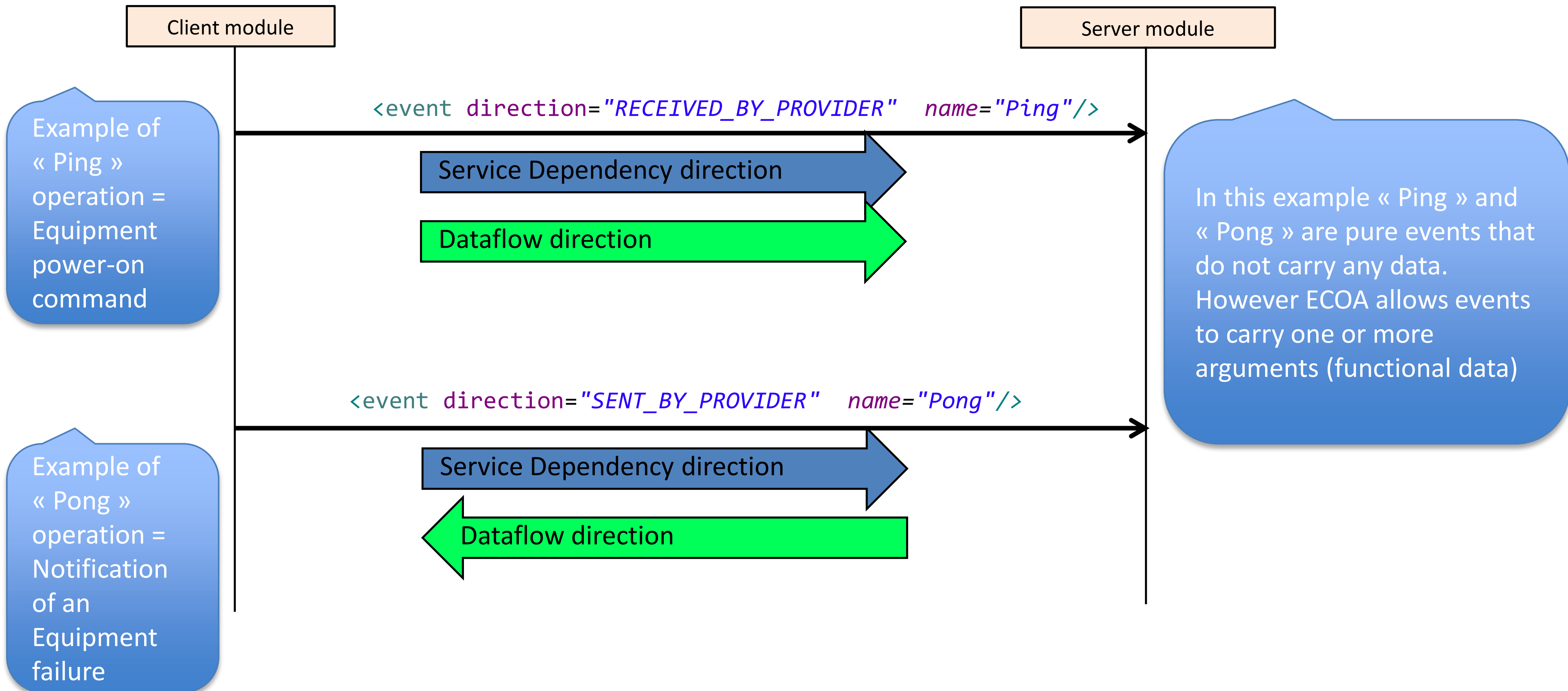
PingPong Example

Predefined Workspace

Folder	Content	
0-Types 1-Services 2-ComponentDefinitions 3-InitialAssembly	*.types.xml - Type definitions used by operations *.interface.xml - Service definitions used to functionally link together components *.componentType - Component contracts (with QoS) *.composite - Initial wiring of components	System Design
4-ComponentImplementations	*.impl.xml - Component implementations (XML, source, binary)	Component Supply
5-Integration	*.impl.composite, logical-system.xml, deployment.xml – Link between component instances and component implementations, Logical system, mapping of modules onto nodes	System Integration

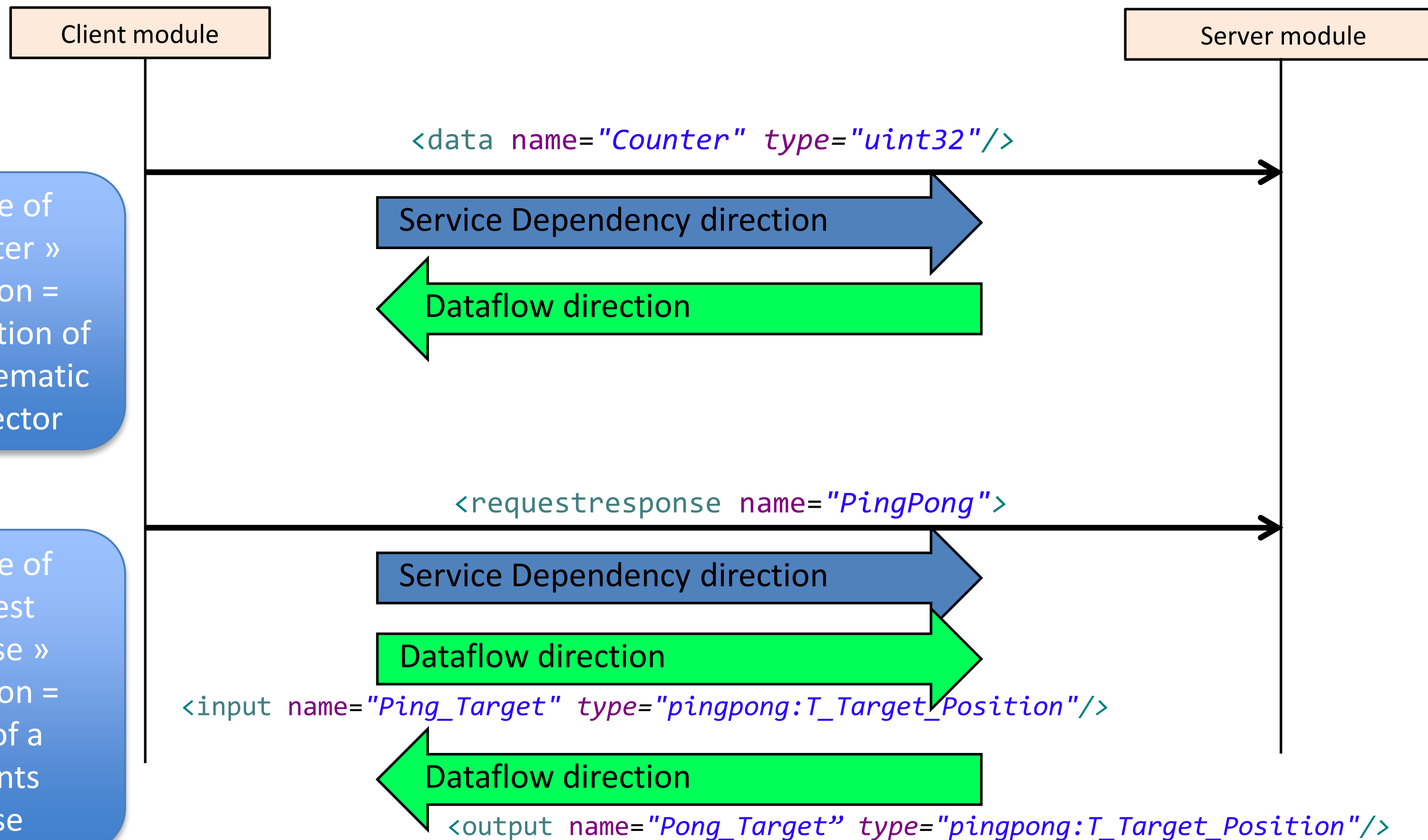
This ECOA tutorial represents the output of a research programme and is provided solely on an 'as is' basis and co-authors of this tutorial make no warranties expressed or implied, including no warranties as to completeness, accuracy or fitness for purpose, with respect to any of the information.

Reminder – services operations (1/2)



This ECOA tutorial represents the output of a research programme and is provided solely on an ‘as is’ basis and co-authors of this tutorial make no warranties expressed or implied, including no warranties as to completeness, accuracy or fitness for purpose, with respect to any of the information.

Reminder – services operations (2/2)



This ECOA tutorial represents the output of a research programme and is provided solely on an 'as is' basis and co-authors of this tutorial make no warranties expressed or implied, including no warranties as to completeness, accuracy or fitness for purpose, with respect to any of the information.

File « pingpong.types.xml »

The name of the library is « pingpong ».

0-Types
1-Services
2-ComponentDefinitions
3-InitialAssembly
4-ComponentImplementations
5-Integration

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

  <types>
    <enum name="T_Side" type="uint8">
      <value name="PING"/>
      <value name="PONG"/>
    </enum>

    <simple name="T_Tactical_Item_ID" type="uint32"/>
    <simple name="T_Angle" type="float32" unit="radian"/>

    <record name="T_2D_Position">
      <field name="Latitude" type="T_Angle"/>
      <field name="Longitude" type="T_Angle"/>
    </record>
    <simple name="T_Time" type="int64" unit="nanoseconds"/>

    <record name="T_Target_Position">
      <field name="Tactical_Item_ID" type="T_Tactical_Item_ID"/>
      <field name="Location" type="T_2D_Position"/>
      <field name="Is_Valid" type="boolean8"/>
    </record>

  </types>
</library>
```

This xml file allows declaring functional datatypes that are being exchanged through service operations.
Each such xml file is a « library » of datatypes that each ECOA xml file may reference in order to use these datatypes when declaring service operations.

Defined by the **system** architect

This ECOA tutorial represents the output of a research programme and is provided solely on an 'as is' basis and co-authors of this tutorial make no warranties expressed or implied, including no warranties as to completeness, accuracy or fitness for purpose, with respect to any of the information.

File « svc_PingPong.interface.xml »

The name of the service is « svc_PingPong ».

0-Types
1-Services
2-ComponentDefinitions
3-InitialAssembly
4-ComponentImplementations
5-Integration

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

  <use library="pingpong"/>

  <operations>

    <requestresponse name="PingPong">
      <input name="Ping_Target" type="pingpong:T_Target_Position"/>
      <output name="Pong_Target" type="pingpong:T_Target_Position"/>
    </requestresponse>

    <event direction="RECEIVED_BY_PROVIDER"  name="Ping"/>
    <event direction="SENT_BY_PROVIDER"  name="Pong"/>

    <data name="Counter" type="uint32"/>

  </operations>

</serviceDefinition>
```

Reference to « pingpong » library which contains datatypes used in this example.

These XML files allow declaring ECOA services.
There is one XML file per ECOA service.
Defining a service consists in defining the prototype of operations provided by this service.
At this stage, ECOA services are not yet instantiated onto provider/user ECOA components.

Defined by the **system** architect

This ECOA tutorial represents the output of a research programme and is provided solely on an ‘as is’ basis and co-authors of this tutorial make no warranties expressed or implied, including no warranties as to completeness, accuracy or fitness for purpose, with respect to any of the information.

File « Ping.componentType »

The name of the component type is « Ping ».

0-Types
1-Services
2-ComponentDefinitions
3-InitialAssembly
4-ComponentImplementations
5-Integration

« reference » = this means that the component requires that service. Syntax parameter must correspond to a service name as defined by the name of an *.interface.xml file.

```
<?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="svc_PingPong">
    <ecoa-sca:interface syntax="svc_PingPong" qos=« Required-svc_PingPong"/>
  </reference>

</componentType>
```

As the component only uses one service typed « svc_PingPong », the same name is chosen for the instance name of the service (but it might have been different).

These XML files allow declaring ECOA component types that can be instantiated in the ECOA assembly.
There is one such XML file per component type.
Declaring a component type consists in declaring which services it provides and which services it requires, among services declared in previous slides.

Defined by the **system** architect

This ECOA tutorial represents the output of a research programme and is provided solely on an 'as is' basis and co-authors of this tutorial make no warranties expressed or implied, including no warranties as to completeness, accuracy or fitness for purpose, with respect to any of the information.

File « Pong.componentType »

0-Types
1-Services
2-ComponentDefinitions
3-InitialAssembly
4-ComponentImplementations
5-Integration

« service » = this means that the component provides that service

```
<?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">

  <service name="svc_PingPong">
    <ecoa-sca:interface syntax="svc_PingPong" qos="Provided-svc_PingPong"/>
  </service>

</componentType>
```

Defined by the **system** architect

This ECOA tutorial represents the output of a research programme and is provided solely on an 'as is' basis and co-authors of this tutorial make no warranties expressed or implied, including no warranties as to completeness, accuracy or fitness for purpose, with respect to any of the information.

File « demo.composite »
(i-e Assembly schema)

0-Types
1-Services
2-ComponentDefinitions
3-InitialAssembly
4-ComponentImplementations
5-Integration

This is how to instantiate an ECOA component

```
<?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="demo"
       targetNamespace="http://www.ecoa.technology/sca_extension-2.0">

  <csa:component name="demoPing">
    <ecoa-sca:instance componentType="Ping"/>
    <csa:reference name="svc_PingPong"/>
  </csa:component>

  <csa:component name="demoPong">
    <ecoa-sca:instance componentType="Pong"/>
    <csa:service name="svc_PingPong"/>
  </csa:component>

  <csa:wire source="demoPing/svc_PingPong" target="demoPong/svc_PingPong" />

</csa:composite>
```

Names of service instances

This XML file allows building a logical system architecture by declaring instances of component types and by connecting provided/required services (this is called « wiring »).
At this stage, ECOA components are only manipulated as « black boxes » with provided/required services.
This is useful at high level system design time.

« wire » = link between two ECOA components, which connects a provided instance of service to a required instance of service, conformly to compliant service contracts (considering interface prototypes and QoS).
There is one wire to be declared per service contract.

Defined by the **system** architect

This ECOA tutorial represents the output of a research programme and is provided solely on an ‘as is’ basis and co-authors of this tutorial make no warranties expressed or implied, including no warranties as to completeness, accuracy or fitness for purpose, with respect to any of the information.

File « myDemoPing.impl.xml » (1/3)

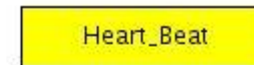
The name of this component implementation is « myDemoPing ». It defines a possible implementation for components typed « Ping ». Several implementations may be defined for the same component type.

0-Types
1-Services
2-ComponentDefinitions
3-InitialAssembly
4-ComponentImplementations
5-Integration

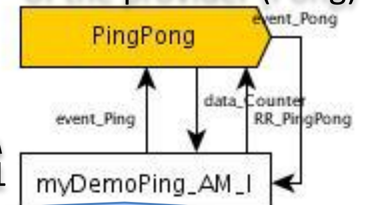
```
<componentImplementation
   xmlns="http://www.ecoa.technology/implementation-2.0" componentDefinition="Ping"

   <!-- list of used libraries -->
  <use library="pingpong"/>

  <!-- module AM to implement provided operations -->
  <moduleType name="myDemoPing_AM_t">
    <operations>
    <requestSent name="PingPong" isSynchronous="true“ timeout="30.0">
      <input name="Ping_Target" type="pingpong:T_Target_Position"/>
      <output name="Pong_Target" type="pingpong:T_Target_Position"/>
    </requestSent>
    <eventSent name="Ping"/>
    <eventReceived name="Pong"/>
    <dataRead type="uint32" name="Counter" maxVersions="8"/>
    <eventReceived name="TriggerPingRequest"/>
    <eventReceived name="TriggerPingEvent"/>
    <eventReceived name="TriggerPingCounter"/>
    </operations>
  </moduleType>

  <moduleImplementation name="myDemoPing_AM" language="C" moduleType="myDemoPing_AM_t"/>
  <moduleInstance name="myDemoPing_AM_I" implementationName="myDemoPing_AM" relativePriority="20"/>
  <triggerInstance name="Heart_Beat" relativePriority="10"/>
```

Declaration of a module type

Module 2



Service = abstraction of the provider (Pong)



Module 1

This XML file allows defining a possible software implementation of an ECOA component instance, i-e its internal breakdown into mono-threaded modules. Therefore a component implementation XML file always references a component type. Defining a component implementation XML is done by following these steps :

1. First of all, declaring module types in the same fashion as component types : module types are being defined by their interface (input/ouput operations at the boundary of each module.) At this stage, operations declared at module type level are not yet related to provided/required services declared at component level.
2. Secondly, this XML file allows defining several possible software implementations of module types (possibly using different languages: C, C++, Ada).
3. Thirdly, this XML file allows declaring module instances. A module instance is defined by the module type that it instantiates and the choice of a module implementation
   It is also possible to declare modules called « trigger » that can be used for implementing periodic activations of other modules (« Heart_Beat » in this example) or for waking up a module after timeout.

Instanciation of module types with their real time attributes relativePriority defines a priority scale within a component.

Defined by the **component** supplier

…

This ECOA tutorial represents the output of a research programme and is provided solely on an ‘as is’ basis and co-authors of this tutorial make no warranties expressed or implied, including no warranties as to completeness, accuracy or fitness for purpose, with respect to any of the information.

File « myDemoPing.impl.xml » (2/3)

4. Finally this XML file allows connecting module interface with:
   - Component interface, in such case it means the module either implements a service provided by the component, or requires a service provided by another component.
   - Another module interface, in such case it corresponds to an internal component interface (not visible outside of the component)

0-Types
1-Services
2-ComponentDefinitions
3-InitialAssembly
4-ComponentImplementations
5-Integration

```
<!-- Definition of module operation links -->
<requestLink>
  <clients><moduleInstance instanceName="myDemoPing_AM_I" operationName="PingPong"/></clients>
  <server><reference instanceName= "svc_PingPong" operationName="PingPong"/></server>
</requestLink>

<eventLink>
  <senders><trigger instanceName="Heart_Beat" period="2.000"/></senders>
  <receivers><moduleInstance instanceName="myDemoPing_AM_I" operationName="TriggerPingRequest"/></receivers>
</eventLink>

<eventLink>
  <senders><trigger instanceName="Heart_Beat" period="3.000"/></senders>
  <receivers><moduleInstance instanceName="myDemoPing_AM_I" operationName="TriggerPingEvent"/>
    <moduleInstance instanceName="myDemoPing_AM_I" operationName="TriggerPingCounter"/></receivers>
</eventLink>

<eventLink>
  <senders><moduleInstance instanceName="myDemoPing_AM_I" operationName="Ping"/></senders>
  <receivers><reference instanceName= "svc_PingPong" operationName="Ping"/></receivers>
</eventLink>
```

Connection to a component interface (i-e component external interface), known by the component service instance name.

Use of a periodic trigger to activate the TriggerPingEvent and TriggerPingCounter entry points of module instance « myDemoPing_AM_I »

Connection to another module interface (i-e component internal interface), defined by the module instance name and its associated module type operation name.

…

Defined by the **component** supplier

This ECOA tutorial represents the output of a research programme and is provided solely on an 'as is' basis and co-authors of this tutorial make no warranties expressed or implied, including no warranties as to completeness, accuracy or fitness for purpose, with respect to any of the information.

File « myDemoPing.impl.xml » (3/3)

0-Types
1-Services
2-ComponentDefinitions
3-InitialAssembly
4-ComponentImplementations
5-Integration

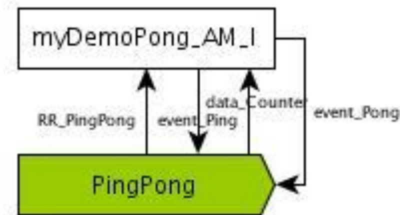
```
  <eventLink>
    <senders><reference instanceName= "svc_PingPong" operationName="Pong"/></senders>
    <receivers><moduleInstance instanceName="myDemoPing_AM_I" operationName="Pong"/></receivers>
  </eventLink>

  <dataLink>
    <writers><reference instanceName= "svc_PingPong" operationName="Counter"/></writers>
    <readers><moduleInstance instanceName="myDemoPing_AM_I" operationName="Counter"/></readers>
  </dataLink>
</componentImplementation>
```

Defined by the **component** supplier

This ECOA tutorial represents the output of a research programme and is provided solely on an 'as is' basis and co-authors of this tutorial make no warranties expressed or implied, including no warranties as to completeness, accuracy or fitness for purpose, with respect to any of the information.

File « myDemoPong.impl.xml »

- 0-Types
- 1-Services
- 2-ComponentDefinitions
- 3-InitialAssembly
- 4-ComponentImplementations
- 5-Integration



See XML file & source code

Defined by the **component** supplier

This ECOA tutorial represents the output of a research programme and is provided solely on an 'as is' basis and co-authors of this tutorial make no warranties expressed or implied, including no warranties as to completeness, accuracy or fitness for purpose, with respect to any of the information.

Ping Applicative Module (1/3)

Reminder : module source code is made of entry points that are activated either on lifecycle events, or according to operations.

0-Types
1-Services
2-ComponentDefinitions
3-InitialAssembly
4-ComponentImplementations
5-Integration

```
/* @file "myDemoPing_AM.c"
 * This is the user code for Module myDemoPing_AM
 */

#include <stdio.h>
#include <string.h>
#include "myDemoPing_AM.h"

/* The following functions must be implemented by this module: */

/* Entrypoints for lifecycle events */
void myDemoPing_AM__INITIALIZE__received(myDemoPing_AM__context* context) {
        /* One-shot initialisation activities: */
        /* To be implemented */
}

void myDemoPing_AM__START__received(myDemoPing_AM__context* context) {
        /* To be implemented */
}

void myDemoPing_AM__STOP__received(myDemoPing_AM__context* context) {
        /* To be implemented */
}

void myDemoPing_AM__SHUTDOWN__received(myDemoPing_AM__context* context) {
        /* To be implemented */
}
```

Defined by the **component** supplier

This ECOA tutorial represents the output of a research programme and is provided solely on an ‘as is’ basis and co-authors of this tutorial make no warranties expressed or implied, including no warranties as to completeness, accuracy or fitness for purpose, with respect to any of the information.

Ping Applicative Module (2/3)

```
/* Entrypoints for module operations */
void myDemoPing_AM__Pong__received
(myDemoPing_AM__context* context)
{
	ECOA__log log = { 13, "Pong received" };
	myDemoPing_AM_container__log_trace(context, log);
}
```

```
void myDemoPing_AM__TriggerPingRequest__received
(myDemoPing_AM__context* context)
 {
	ECOA__return_status return_status;
	ECOA__log log;
	pingpong__T_Target_Position pingTarget =
					{ 1, { 45, 45 }, ECOA__TRUE };
	pingpong__T_Target_Position pongTarget;

	 return_status = myDemoPing_AM_container__PingPong__request_sync (context,
				&pingTarget, &pongTarget);

	  if (return_status != ECOA__return_status_OK)
	  {
			snprintf(log.data, ECOA__LOG_MAXSIZE, "Request return_status : %2d",
					return_status);
			log.current_size = strlen(log.data);
	    } else
	   {
			snprintf(log.data, ECOA__LOG_MAXSIZE,
					"Pong response : %2d %2.1f %2.1f %d",
					pongTarget.Tactical_Item_ID, pongTarget.Location.Latitude,
					pongTarget.Location.Longitude, pongTarget.Is_Valid);
			log.current_size = strlen(log.data);
	    }
	    myDemoPing_AM_container__log_trace(context, log);

}
```


0-Types
1-Services
2-ComponentDefinitions
3-InitialAssembly
4-ComponentImplementations
5-Integration

Defined by the **component** supplier

This ECOA tutorial represents the output of a research programme and is provided solely on an ‘as is’ basis and co-authors of this tutorial make no warranties expressed or implied, including no warranties as to completeness, accuracy or fitness for purpose, with respect to any of the information.

Ping Applicative Module (3/3)

0-Types
1-Services
2-ComponentDefinitions
3-InitialAssembly
4-ComponentImplementations
5-Integration

```
void  myDemoPing_AM__TriggerPingEvent__received
(myDemoPing_AM__context* context)
{
    myDemoPing_AM_container__Ping__send(context);
}
```

```
void myDemoPing_AM__TriggerPingCounter__received
(myDemoPing_AM__context* context)
{
        ECOA__log log;
        ECOA__log return_status_log = { 14, "Release error" };
        myDemoPing_AM_container__Counter_handle handle;
        ECOA__return_status return_status;

        return_status = myDemoPing_AM_container__Counter__get_read_access(context,
                                  &handle);
        if (return_status == ECOA__return_status_OK)
        {
                snprintf(log.data, ECOA__LOG_MAXSIZE, "Counter : %2d", *handle.data);
                log.current_size = strlen(log.data);

                myDemoPing_AM_container__log_trace(context, log);

                return_status = myDemoPing_AM_container__Counter__release_read_access
                                     (context, &handle);
                if (return_status != ECOA__return_status_OK)
                {
                      myDemoPing_AM_container__log_debug(context, return_status_log);
                }
        }
}
```

Defined by the **component** supplier

This ECOA tutorial represents the output of a research programme and is provided solely on an ‘as is’ basis and co-authors of this tutorial make no warranties expressed or implied, including no warranties as to completeness, accuracy or fitness for purpose, with respect to any of the information.

Logical-system.xml

0-Types
1-Services
2-ComponentDefinitions
3-InitialAssembly
4-ComponentImplementations
5-Integration

```
<ecoa:logicalSystem id="cs1"
  xmlns:ecoa="http://www.ecoa.technology/logicalsystem-2.0">

  <!--
  Computing Node = « Alienware Aurora » desktop PC
  HyperThreading disabled (BIOS config)
  4 CPU cores
  Intel(R) Core(TM) i7-2600K CPU @ 3.40GHz
  Bogomips :  6785.34  (dmesg | grep BogoMIPS)
  stepDuration = 1/BogoMIPS = 1.47376e-4 s
  -->

 <logicalComputingPlatform id="myPlatform">
   <logicalComputingNode id="machine0">
     <endianess type="BIG" />
     <logicalProcessors number="4" type="x86_64">
       <stepDuration nanoSeconds="147376" />
     </logicalProcessors>
     <os name="linux" />
     <availableMemory gigaBytes="6" />
     <moduleSwitchTime microSeconds="10" />
   </logicalComputingNode>
 </logicalComputingPlatform>

</ecoa:logicalSystem>
```

This XML file allows declaring high-level physical characteristics of target ECOA platforms resources.

Several platforms can be defined to allow a multi-platforms deployment (supposing then platforms compliance with optional ECOA ELI implementation)

Provided by the **platform** supplier

This ECOA tutorial represents the output of a research programme and is provided solely on an ‘as is’ basis and co-authors of this tutorial make no warranties expressed or implied, including no warranties as to completeness, accuracy or fitness for purpose, with respect to any of the information.

Demo.impl.composite

0-Types
1-Services
2-ComponentDefinitions
3-InitialAssembly
4-ComponentImplementations
5-Integration

```
<?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="demo"
       targetNamespace="http://www.ecoa.technology/sca-extension-2.0">

  <csa:component name="demoPing">
    <ecoa-sca:instance componentType="Ping">
      <ecoa-sca:implementation name="myDemoPing"/>
    </ecoa-sca:instance>
    <csa:reference name="svc_PingPong"/>
  </csa:component>

  <csa:component name="demoPong">
    <ecoa-sca:instance componentType="Pong">
      <ecoa-sca:implementation name="myDemoPong"/>
    </ecoa-sca:instance>
    <csa:service name="svc_PingPong"/>
  </csa:component>

  <csa:wire source="demoPing/svc_PingPong" target="demoPong/svc_PingPong"/>

</csa:composite>
```

This XML file allows declaring the software level system architecture. It consists of declaring the assembly of component instances, taking into account chosen component implementation for each component instance.
Consequently this XML file is a software solution to the logical system architecture previously defined at component type level (in Demo.composite file).

Defined by the **system** integrator

This ECOA tutorial represents the output of a research programme and is provided solely on an ‘as is’ basis and co-authors of this tutorial make no warranties expressed or implied, including no warranties as to completeness, accuracy or fitness for purpose, with respect to any of the information.

Deployment.xml

0-Types
1-Services
2-ComponentDefinitions
3-InitialAssembly
4-ComponentImplementations
5-Integration

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

  <protectionDomain name="Ping_PD">
    <executeOn computingNode="machine0"  computingPlatform="myPlatform"/>
    <deployedModuleInstance componentName="demoPing" moduleInstanceName="myDemoPing_AM_I" modulePriority="30"/>
    <deployedTriggerInstance componentName="demoPing" triggerInstanceName="Heart_Beat" triggerPriority="10"/>
  </protectionDomain>

  <protectionDomain name="Pong_PD">
    <executeOn computingNode="machine0" computingPlatform="myPlatform"/>
    <deployedModuleInstance componentName="demoPong" moduleInstanceName="myDemoPong_AM_I" modulePriority="30"/>
  </protectionDomain>

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

</deployment>
```

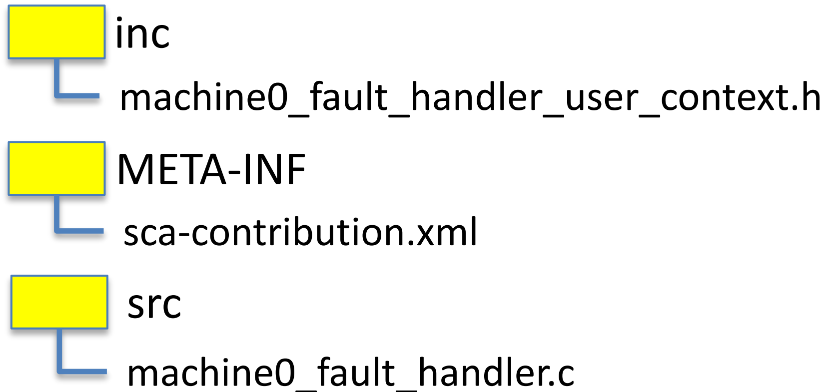
This XML file allows mapping the ECOA SW architecture onto computing nodes of the target ECOA computing platforms.
This file is used by each ECOA computing platform for configuring fault handler notifications, loading and deploying its associated components conformly to component implementations stored in « 4-ComponentImplementations ».

Module priorities are being defined so as to allow a DMA (Deadline Monotonic Approach) scheduling of ECOA modules by the platform, on each computing node (provided this is the chosen strategy by the system integrator for scheduling modules).
The system integrator has to choose module priorities that are compliant with module relative priorities specified for each component in component implementation files.

Defined by the **system** integrator

This ECOA tutorial represents the output of a research programme and is provided solely on an ‘as is’ basis and co-authors of this tutorial make no warranties expressed or implied, including no warranties as to completeness, accuracy or fitness for purpose, with respect to any of the information.

Other integration files



0-Types
1-Services
2-ComponentDefinitions
3-InitialAssembly
4-ComponentImplementations
5-Integration

inc/ and src/ directories allow defining files to configure the ECOA Fault Handler, for platforms on which the ECOA Fault Handler is implemented as a function within the infrastructure rather than an ASC.

This example illustrates a platform on which the ECOA Fault Handler is implemented as a function of the ECOA infrastructure. In that case, platform documentation gives ECOA Fault Handler level (platform or node), which allows defining files and functions names. Files content has then to be filled conformly to expected behaviour in case of error.

There may be complementary integration files which are not required by ECOA standard .
In this example, META-INF directory ensures compliance with SCA standard, as required by a target platform.
Platform documentation provides information on specific integration files requirements.

Defined by the **system** integrator

This ECOA tutorial represents the output of a research programme and is provided solely on an 'as is' basis and co-authors of this tutorial make no warranties expressed or implied, including no warranties as to completeness, accuracy or fitness for purpose, with respect to any of the information.