ECOA® Software Description with UML

Introduction

The European Component Oriented Architecture (ECOA®) presents a methodology for designing software systems as a series of Application Software Components (ASCs), each of which is implemented as one or more ECOA Modules. Only Modules have a physical implementation as software code.

The ECOA methodology divides the design task into a series of representations captured in XML, as described in ref.[1], and provides a software system (ECOA "Assembly") level diagram taken from SCA (ref.[2]). However, to provide a model based approach to designing at the ASC and ECOA Module level, UML (ref.[3]) is the preferred means. This paper describes a ***prototype***[1] "UML Profile" for designing ECOA ASCs and Modules – that is, how to represent the design and implementation artefacts required in ECOA software using UML.

Any views, opinions, or recommendations in this paper are those of the author and do not necessarily reflect those of BAE Systems.

Aims

In an ideal world, there would be a formal UML Profile for ECOA, i.e. a formal definition of how to represent ECOA artefacts using UML notation. In the absence of a formally defined profile, a prototype profile is (briefly) described here for use by software engineers designing and constructing ECOA software systems.

In time, this UML Profile may be developed to be supported by one or more tools to translate from UML diagrams drawn against the Profile into ECOA XML definition files, but these tools will of necessity be UML tool specific (e.g. Enterprise Architect, IBM Rational Rhapsody, or Eclipse/Papyrus). Use of such translation tools, coupled with ECOA XML-to-Code Generation tools, would complete the ECOA software development path from diagram to implementation code.

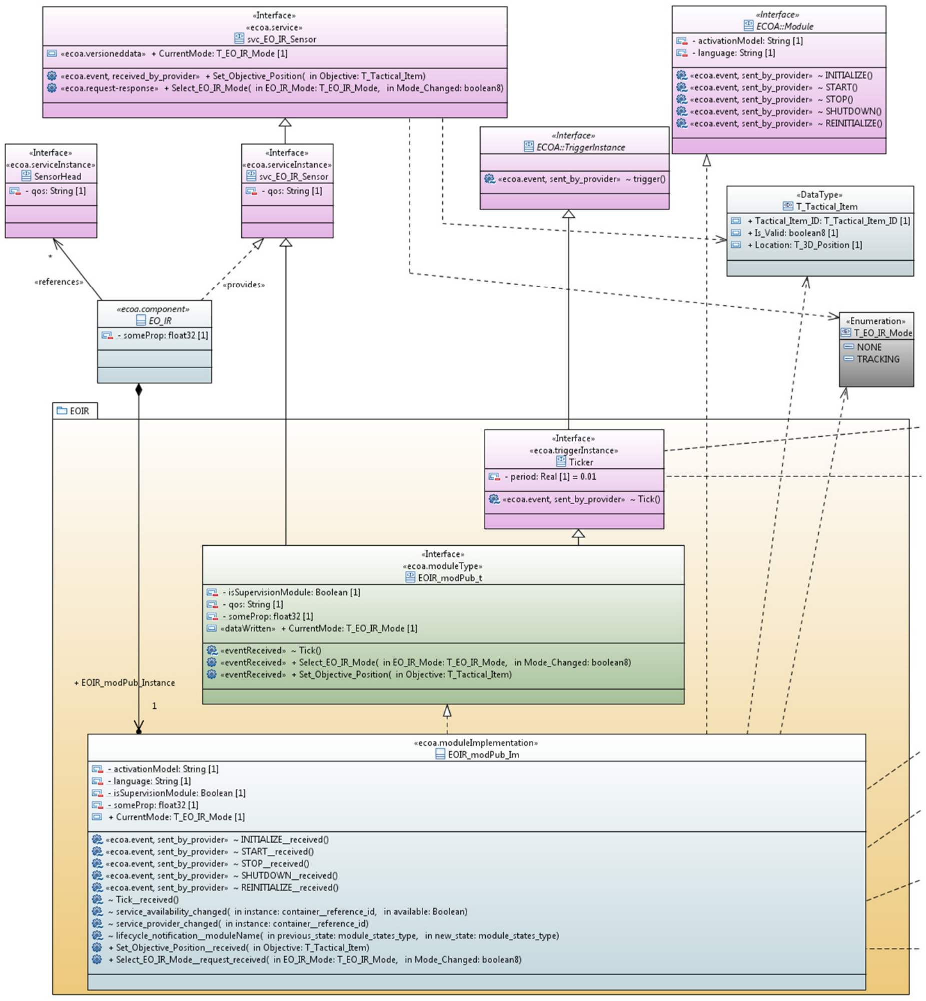
[1] It is therefore *expected* that at any given time, this profile will be incomplete and in some aspects wrong, but that it will progress, steadily, towards perfection (!) as time, circumstances, and attempts to use it, proceed.

Prototype UML Profile for ECOA

Figure 1 below depicts an example ECOA ASC (Component) decomposition using UML stereotypes to sub-classify UML classes, and the relationships between those classes, so as to represent ECOA entities.

The diagram depicts the composition and design artefact relationships for the ECOA ASC (Component) "*EO_IR*", which *provides* the ECOA Server "*svc_EO_IR_Sensor*", and *references* the ECOA Service "*SensorHead*". The software code implementation of the ASC is encompassed by the ECOA Module Implementation "*EOIR_modPub_Im*".

Figure 1 ECOA Software Description - Example UML Diagram[2][3]


«Interface»
«ecoa.service»
svc_EO_IR_Sensor
«ecoa.versioneddata» + CurrentMode: T_EO_IR_Mode [1]
«ecoa.event, received_by_provider» + Set_Objective_Position(in Objective: T_Tactical_Item)
«ecoa.request-response» + Select_EO_IR_Mode(in EO_IR_Mode: T_EO_IR_Mode, in Mode_Changed: boolean8)
«Interface»
ECOA::Module
- activationModel: String [1]
- language: String [1]
«ecoa.event, sent_by_provider» ~ INITIALIZE()
«ecoa.event, sent_by_provider» ~ START()
«ecoa.event, sent_by_provider» ~ STOP()
«ecoa.event, sent_by_provider» ~ SHUTDOWN()
«ecoa.event, sent_by_provider» ~ REINITIALIZE()
«Interface»
«ecoa.serviceInstance»
SensorHead
- qos: String [1]
«Interface»
«ecoa.serviceInstance»
svc_EO_IR_Sensor
- qos: String [1]
«Interface»
ECOA::TriggerInstance
«ecoa.event, sent_by_provider» ~ trigger()
«DataType»
T_Tactical_Item
+ Tactical_Item_ID: T_Tactical_Item_ID [1]
+ Is_Valid: boolean8 [1]
+ Location: T_3D_Position [1]
*
«references»
«provides»
«ecoa.component»
EO_IR
- someProp: float32 [1]
«Enumeration»
T_EO_IR_Mode
NONE
TRACKING
EOIR
«Interface»
«ecoa.triggerInstance»
Ticker
- period: Real [1] = 0.01
«ecoa.event, sent_by_provider» ~ Tick()
«Interface»
«ecoa.moduleType»
EOIR_modPub_t
- isSupervisionModule: Boolean [1]
- qos: String [1]
- someProp: float32 [1]
«dataWritten» + CurrentMode: T_EO_IR_Mode [1]
«eventReceived» ~ Tick()
«eventReceived» + Select_EO_IR_Mode(in EO_IR_Mode: T_EO_IR_Mode, in Mode_Changed: boolean8)
«eventReceived» + Set_Objective_Position(in Objective: T_Tactical_Item)
+ EOIR_modPub_Instance
1
«ecoa.moduleImplementation»
EOIR_modPub_Im
- activationModel: String [1]
- language: String [1]
- isSupervisionModule: Boolean [1]
- someProp: float32 [1]
+ CurrentMode: T_EO_IR_Mode [1]
«ecoa.event, sent_by_provider» ~ INITIALIZE__received()
«ecoa.event, sent_by_provider» ~ START__received()
«ecoa.event, sent_by_provider» ~ STOP__received()
«ecoa.event, sent_by_provider» ~ SHUTDOWN__received()
«ecoa.event, sent_by_provider» ~ REINITIALIZE__received()
~ Tick__received()
~ service_availability_changed(in instance: container__reference_id, in available: Boolean)
~ service_provider_changed(in instance: container__reference_id)
~ lifecycle_notification__moduleName(in previous_state: module_states_type, in new_state: module_states_type)
+ Set_Objective_Position__received(in Objective: T_Tactical_Item)
+ Select_EO_IR_Mode__request_received(in EO_IR_Mode: T_EO_IR_Mode, in Mode_Changed: boolean8)

[2] This diagram has been drawn with the Papyrus plug-in to Eclipse. Other UML tools are available.

[3] Different UML tools will colour UML entities differently. The colouring scheme suggested in this profile is given in the Summary Tables later in the document.

UML diagrams drawn against this profile, like Figure 1, are of course first and foremost, UML diagrams, not “ECOA” diagrams. UML “classes” are used to represent software types and entities (drawn as rectangular boxes) and directed arrows between them indicate the “association” or “dependency” relationship one has with another. Classes and associations are then “stereotyped” to distinguish variations from the basic entity, by enclosing a clarifying stereotype name in guillemets, such as data type classes stereotyped *«Enumeration»* to denote a data type comprising discrete named vales rather than numerical values. Association arrows are also drawn differently to distinguish the major types of association, such as a UML “generalisation” (a solid line with a closed, hollow, arrowhead) and a “composed of” (aggregation) association (a solid line with a diamond shape (which may be hollow or filled depending on the sub-type of the aggregation) on one end, and possibly with an open arrowhead on one or both ends). For a much better explanation of UML, see refs.[3] and [4].

In respect of designing and describing ECOA ASCs using UML as a visual medium, the representations we need to address are as follows – labelled according to the traditional ECOA development Steps:

- ECOA Data Types (Step 0)
- ECOA Services (Step 1)
- ECOA Assemblies (Step 3)
- ECOA Component Definitions (Step 2)
- ECOA Component Implementations (Step 4)
  - ECOA Service Instances
  - ECOA Module Types
  - ECOA Module Implementations
  - ECOA Module Instances
- ECOA Integration & Deployment (Step 5)

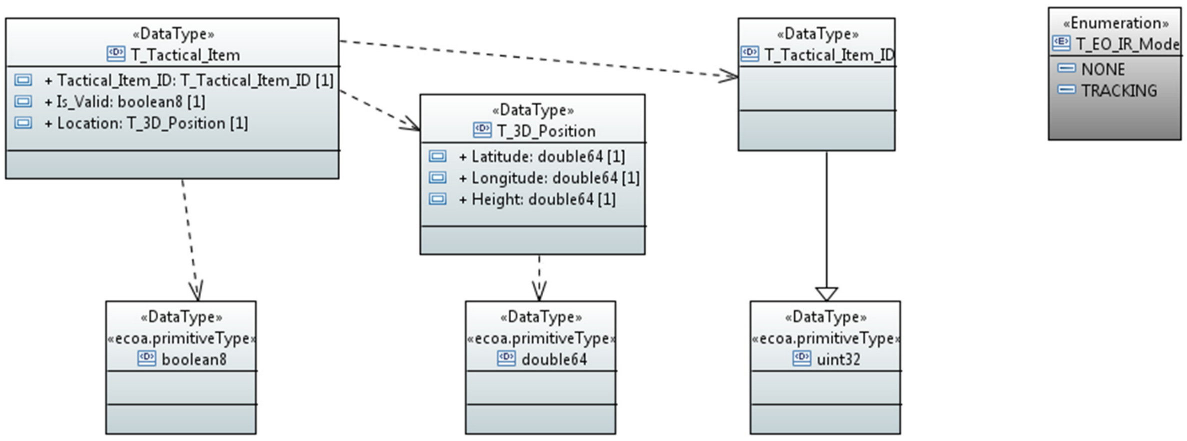
Note that “Step 3” appears out of order in this document because a development process would more likely involve sketching, and then hardening up on, a set of Services and an Assembly to meet a set of Requirements – identifying the Components required as they emerge from that process. More rarely might a predetermined set of Components be known prior to arranging them into an Assembly.

ECOA Data Types

The ECOA predefined types are simply represented as UML *«ecoa.primitiveType»* elements (that is UML elements stereotyped as "*«ecoa.primitiveType»*"), reflecting an unstructured data type. Enumeration types are represented using UML *«Enumeration»* elements that include the list of enumeration values (such as the *T_EO_IR_Mode* type represented in Figure 1).

Compound (structured) data types (such as records and unions) are represented as UML *«DataType»* elements, with the fields of the compound type represented as properties of the type (such as the *T_Tactical_Item* type in Figure 2, which has fields named *Tactical_Item_ID*, *Is_Valid*, and *Location*).

Figure 2 Example ECOA (Step 0) Data Types Definition Expressed in UML



ECOA data types are defined in XML "*name.types.xml*" files, using: *<enum>* XML tags for enumeration types; *<simple>* XML tags for unstructured types; *<record>* or *<variantRecord>* XML tags for records; and *<array>* or *<fixedArray>* XML tags for array types.

ECOA Services

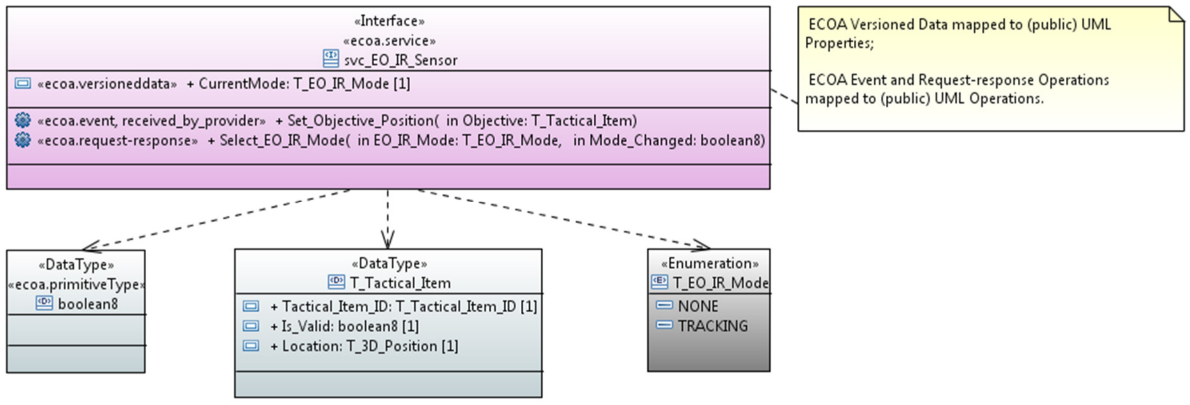
An ECOA Service is represented using a UML *«Interface»* class, an abstract definition of the operations of the Service, together with a description of the functionality of each.

Since *«Interface»*[4] classes will be used to represent several different ECOA artefacts in this Profile, ECOA Services are additionally stereotyped "*«ecoa.service»*" for clarity.

Event and Request-Response ECOA Operations of a Service appear as simply operations of the *«ecoa.service»* class. ECOA Versioned Data items are most appropriately represented as UML Properties of the *«ecoa.service»* class. The data types of the parameters of the Operations, and of Versioned Data items, can be linked graphically to the *«ecoa.service»* class using dependency associations (dashed arrows), as in Figure 3.

The Service Operations (including Versioned Data items) are themselves stereotyped as *«ecoa.event»*, *«ecoa.request-response»*, or *«ecoa.versioneddata»* as required. *«ecoa.event»* Service Operations are further stereotyped *«ecoa.sent_by_provider»* or *«ecoa.received_by_provider»* as the case may be.

Figure 3 Example ECOA (Step 1) Service Definition Expressed in UML


«Interface»
«ecoa.service»
svc_EO_IR_Sensor
«ecoa.versioneddata» + CurrentMode: T_EO_IR_Mode [1]
«ecoa.event, received_by_provider» + Set_Objective_Position(in Objective: T_Tactical_Item)
«ecoa.request-response» + Select_EO_IR_Mode(in EO_IR_Mode: T_EO_IR_Mode, in Mode_Changed: boolean8)
ECOA Versioned Data mapped to (public) UML Properties;
ECOA Event and Request-response Operations mapped to (public) UML Operations.
«DataType»
«ecoa.primitiveType»
boolean8
«DataType»
T_Tactical_Item
+ Tactical_Item_ID: T_Tactical_Item_ID [1]
+ Is_Valid: boolean8 [1]
+ Location: T_3D_Position [1]
«Enumeration»
T_EO_IR_Mode
NONE
TRACKING

ECOA Services are defined in XML "*name.interface.xml*" files, using *<serviceDefinition>* XML tags.

[4] The Eclipse/Papyrus tools use capitalized names for the built-in stereotype names, such as "*«Interface»*". Other tools may use lower case only, as has been used for the ECOA stereotypes introduced by this profile.

ECOA Assemblies

An ECOA Assembly is represented using a UML Composite Structure Diagram such as the fragment shown in Figure 4. This can be compared with the equivalent SCA derived Composite Diagram fragment of Figure 5. UML classes are used to depict the *«ecoa.component»*s of the Assembly, with UML Ports exported to represent ECOA *«ecoa.service»*s or *«ecoa.reference»*s, as appropriate.

ECOA Properties are represented with UML Properties of the *«ecoa.component»*s, whilst UML Comments are used to capture the Component's Insertion Policies. Note that there is no proposed method to capture Insertion Policies in the SCA diagram form.

Figure 4 Example ECOA (Step 3) Assembly Definition Expressed in UML

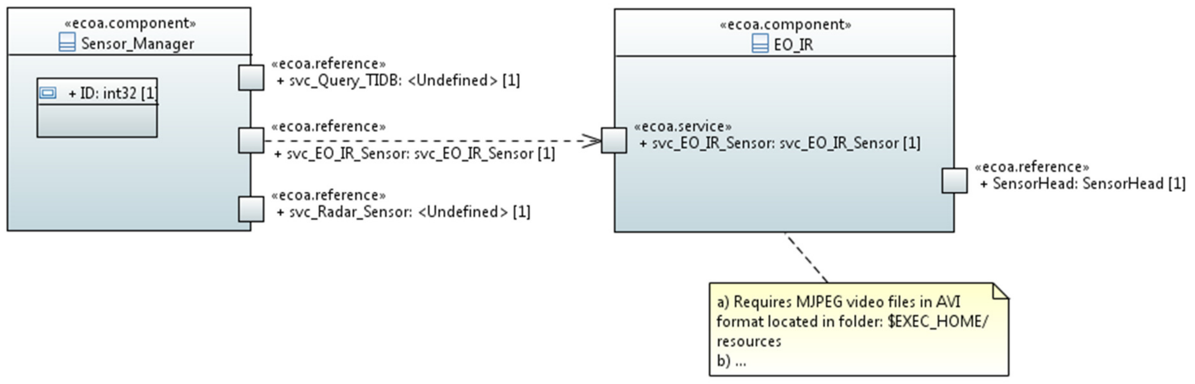
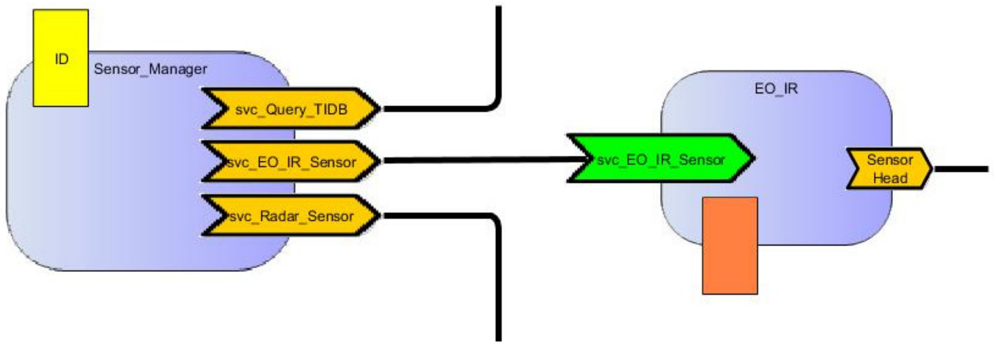

«ecoa.component»
Sensor_Manager
+ ID: int32 [1]
«ecoa.reference»
+ svc_Query_TIDB: <Undefined> [1]
«ecoa.reference»
+ svc_EO_IR_Sensor: svc_EO_IR_Sensor [1]
«ecoa.reference»
+ svc_Radar_Sensor: <Undefined> [1]
«ecoa.component»
EO_IR
«ecoa.service»
+ svc_EO_IR_Sensor: svc_EO_IR_Sensor [1]
«ecoa.reference»
+ SensorHead: SensorHead [1]
a) Requires MJPEG video files in AVI format located in folder: $EXEC_HOME/resources
b) ...

Figure 5 Example ECOA (Step 3) Assembly Diagram



ECOA Assemblies are initially defined (at ECOA development Step 3) in XML "*name.composite*" files, using *<composite>* XML tags. Within these, ECOA ASC (Component) of the Assembly is declared between *<component>* XML tags, and each Service Link (wire) between Services of the ASCs is declared using *<wire>* XML tags.

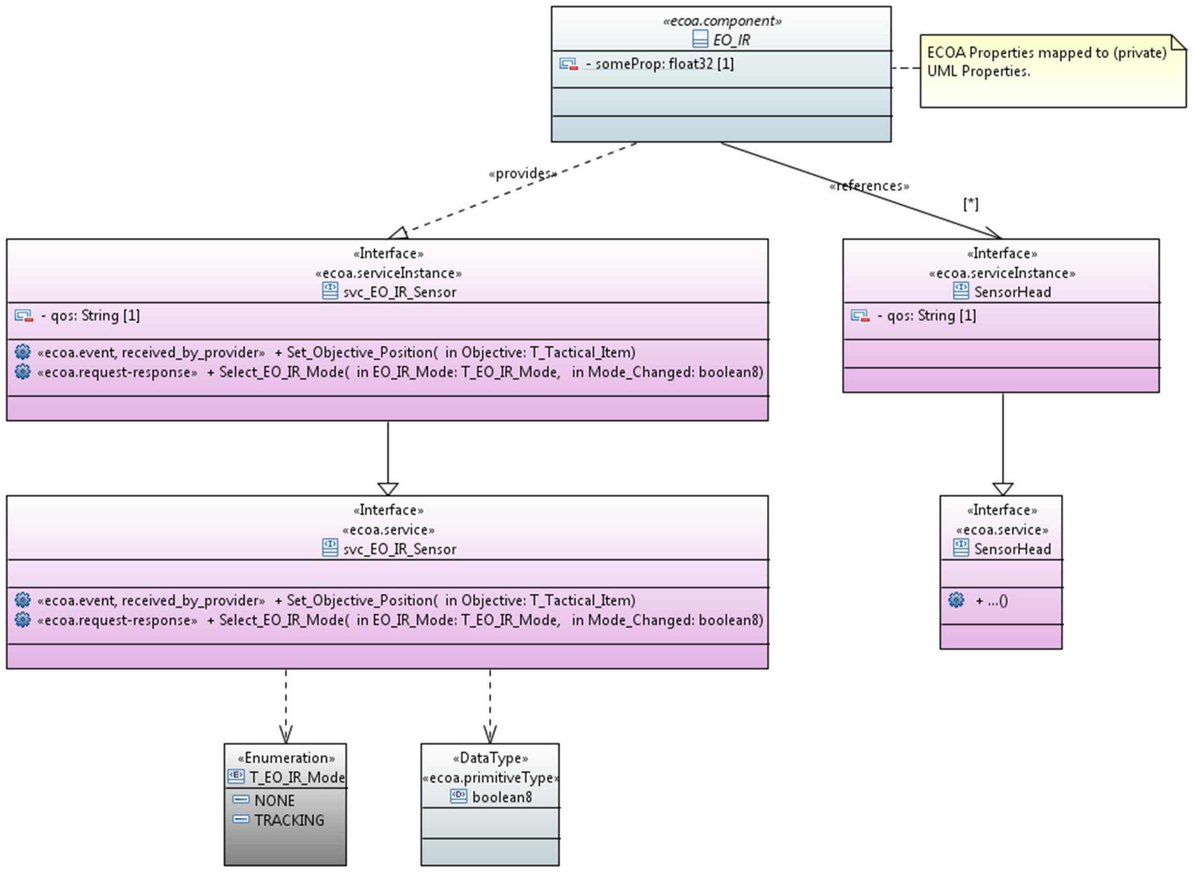
At ECOA development Step 5, the Assembly definition is expanded, in a "*name_impl.composite*" XML file, to include, for each Component, a naming of the Component Implementation that is produced during ECOA development Step 4.

ECOA Component Definitions

An ECOA Component Definition is represented using a (concrete) UML class, stereotyped *«ecoa.component»* (e.g. the EO_IR *«ecoa.component»* in Figure 6). As ECOA Components have no physical realization, *«ecoa.component»* UML classes are made abstract.

ECOA Components are defined (initially) simply in terms of the ECOA Services that the Component provides and/or references - or more precisely the particular instances of Services. The provision of (an instance of) a Service is depicted using a UML realization association, stereotyped *«provides»*. A referenced (instance of) Service is depicted with a UML unidirectional association, stereotyped *«references»*.

Figure 6 Example ECOA (Step 2) Component Definition Expressed in UML

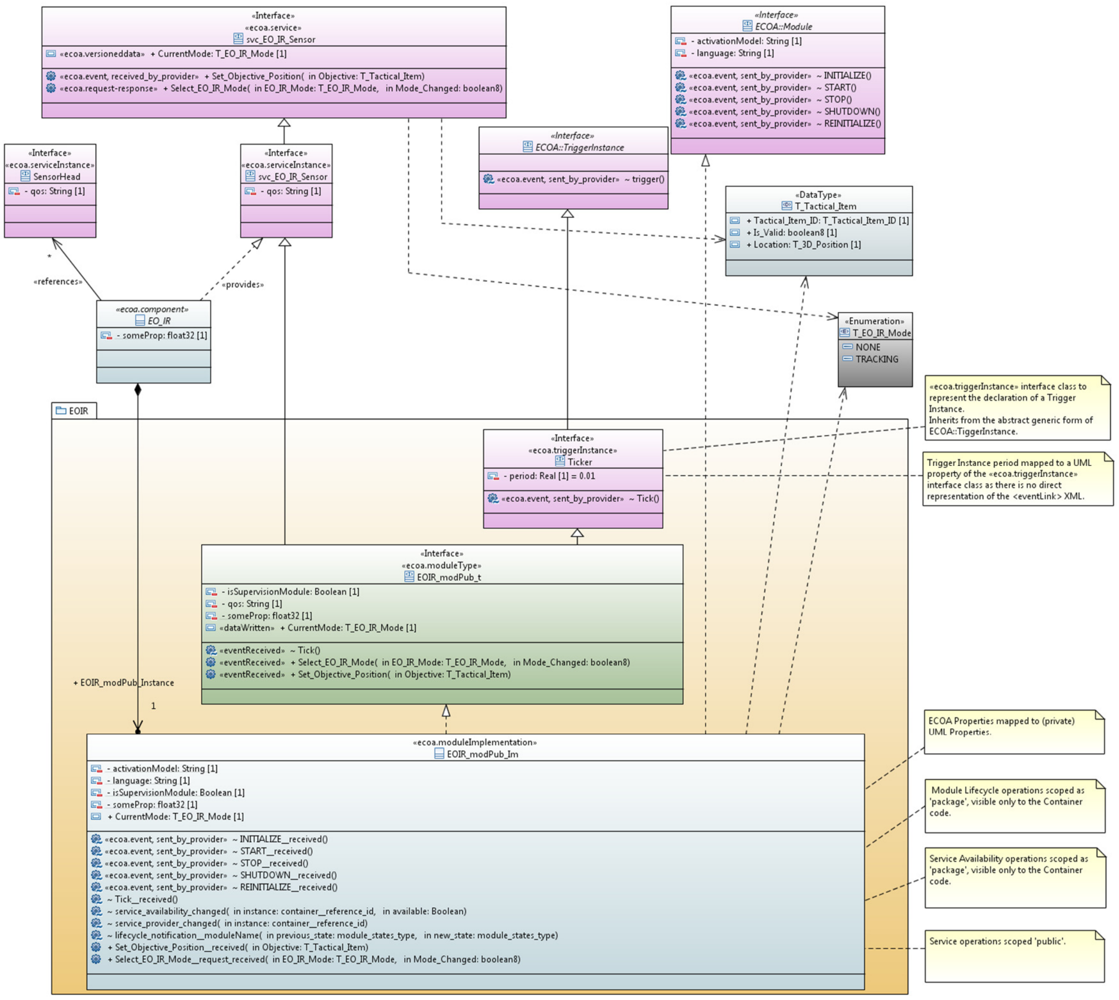

«ecoa.component»
EO_IR
- someProp: float32 [1]
ECOA Properties mapped to (private) UML Properties.
«provides»
«references»
[*]
«Interface»
«ecoa.serviceInstance»
svc_EO_IR_Sensor
- qos: String [1]
«ecoa.event, received_by_provider» + Set_Objective_Position(in Objective: T_Tactical_Item)
«ecoa.request-response» + Select_EO_IR_Mode(in EO_IR_Mode: T_EO_IR_Mode, in Mode_Changed: boolean8)
«Interface»
«ecoa.serviceInstance»
SensorHead
- qos: String [1]
«Interface»
«ecoa.service»
svc_EO_IR_Sensor
«ecoa.event, received_by_provider» + Set_Objective_Position(in Objective: T_Tactical_Item)
«ecoa.request-response» + Select_EO_IR_Mode(in EO_IR_Mode: T_EO_IR_Mode, in Mode_Changed: boolean8)
«Interface»
«ecoa.service»
SensorHead
+ ...()
«Enumeration»
T_EO_IR_Mode
NONE
TRACKING
«DataType»
«ecoa.primitiveType»
boolean8

ECOA Components are defined in XML "*name.componentType*" files, using *<componentType>* XML tags.

ECOA Component Implementations

An ECOA Component Implementation is represented by describing the decomposition of the Component into its constituent ECOA Modules and Trigger Instances. An example of such a decomposition we have already seen (as Figure 1), but here it is again in context (Figure 7).

Figure 7 Example ECOA (Step 4) Component Implementation Definition Expressed in UML


«Interface»
«ecoa.service»
svc_EO_IR_Sensor
«ecoa.versioneddata» + CurrentMode: T_EO_IR_Mode [1]
«ecoa.event, received_by_provider» + Set_Objective_Position(in Objective: T_Tactical_Item)
«ecoa.request-response» + Select_EO_IR_Mode(in EO_IR_Mode: T_EO_IR_Mode, in Mode_Changed: boolean8)
«Interface»
ECOA::Module
- activationModel: String [1]
- language: String [1]
«ecoa.event, sent_by_provider» ~ INITIALIZE()
«ecoa.event, sent_by_provider» ~ START()
«ecoa.event, sent_by_provider» ~ STOP()
«ecoa.event, sent_by_provider» ~ SHUTDOWN()
«ecoa.event, sent_by_provider» ~ REINITIALIZE()
«Interface»
ECOA::TriggerInstance
«ecoa.event, sent_by_provider» ~ trigger()
«Interface»
«ecoa.serviceInstance»
SensorHead
- qos: String [1]
«Interface»
«ecoa.serviceInstance»
svc_EO_IR_Sensor
- qos: String [1]
«DataType»
T_Tactical_Item
+ Tactical_Item_ID: T_Tactical_Item_ID [1]
+ Is_Valid: boolean8 [1]
+ Location: T_3D_Position [1]
*
«references»
«provides»
«ecoa.component»
EO_IR
- someProp: float32 [1]
«Enumeration»
T_EO_IR_Mode
NONE
TRACKING
«ecoa.triggerInstance» interface class to represent the declaration of a Trigger Instance.
Inherits from the abstract generic form of ECOA::TiggerInstance.
EOIR
«Interface»
«ecoa.triggerInstance»
Ticker
- period: Real [1] = 0.01
«ecoa.event, sent_by_provider» ~ Tick()
Trigger Instance period mapped to a UML property of the «ecoa.triggerInstance» interface class as there is no direct representation of the <eventLink> XML.
«Interface»
«ecoa.moduleType»
EOIR_modPub_t
- isSupervisionModule: Boolean [1]
- qos: String [1]
- someProp: float32 [1]
«dataWritten» + CurrentMode: T_EO_IR_Mode [1]
«eventReceived» ~ Tick()
«eventReceived» + Select_EO_IR_Mode(in EO_IR_Mode: T_EO_IR_Mode, in Mode_Changed: boolean8)
«eventReceived» + Set_Objective_Position(in Objective: T_Tactical_Item)
+ EOIR_modPub_Instance
1
ECOA Properties mapped to (private) UML Properties.
«ecoa.moduleImplementation»
EOIR_modPub_Im
- activationModel: String [1]
- language: String [1]
- isSupervisionModule: Boolean [1]
- someProp: float32 [1]
+ CurrentMode: T_EO_IR_Mode [1]
«ecoa.event, sent_by_provider» ~ INITIALIZE__received()
«ecoa.event, sent_by_provider» ~ START__received()
«ecoa.event, sent_by_provider» ~ STOP__received()
«ecoa.event, sent_by_provider» ~ SHUTDOWN__received()
«ecoa.event, sent_by_provider» ~ REINITIALIZE__received()
~ Tick__received()
~ service_availability_changed(in instance: container__reference_id, in available: Boolean)
~ service_provider_changed(in instance: container__reference_id)
~ lifecycle_notification__moduleName(in previous_state: module_states_type, in new_state: module_states_type)
+ Set_Objective_Position__received(in Objective: T_Tactical_Item)
+ Select_EO_IR_Mode__request_received(in EO_IR_Mode: T_EO_IR_Mode, in Mode_Changed: boolean8)
Module Lifecycle operations scoped as 'package', visible only to the Container code.
Service Availability operations scoped as 'package', visible only to the Container code.
Service operations scoped 'public'.

ECOA Service Instances

The particular instance of an ECOA Service, as provided by a particular ECOA Component, is depicted using a UML *«Interface»* class, stereotyped (unsurprisingly) as an *«ecoa.serviceInstance»*. An *«ecoa.serviceInstance»* is graphically related to its parent *«ecoa.service»* using a UML generalization association.

An ECOA Service Instance names, in its definition, an XML file containing a definition of the Quality of Service provided by that Service Instance. This is depicted in the UML model by a "*qos*" attribute of the *«ecoa.serviceInstance»* class (see Figure 6).

ECOA Service Instances are defined in XML in the "*name.componentType*" file as elements of the *<componentType>* structure, using *<service>* XML tags. The *<componentType>* structure also lists the Service Instances referenced by the Component, Service Instances defined in the *<componentType>* structure for the other (providing) Component.

For diagram clarity, it is permissible to omit *«ecoa.serviceInstance»* classes on a UML class diagram, and to depict an *«ecoa.component»* class as having *«provides»* or *«references»* relationships directly to the *«ecoa.service»* interface class.

ECOA Module Types

ECOA Components are implemented in software code by one or more ECOA Modules. Modules are initially defined by a Module Type, an abstract definition (in XML) of the Module. A Module Type is depicted once again by a UML *«Interface»* class, this time stereotyped as an *«ecoa.moduleType»*.

A Module Type can be thought of as an abstract realization (by expansion in XML) of one or more Service Instances, or parts of one or more Service Instances. Diagrammatically, this abstract realization is depicted using a UML generalization association, the *«ecoa.moduleType»* being a specialization (inverse of the generalization) of the *«ecoa.serviceInstance»*. To further emphasize that an *«ecoa.moduleType»* interface class is the first step from the service domain into the implementation domain, the UML representation will be coloured green (rather than the pink of abstract interfaces, or blue of concrete classes).

A Module Type declaration specifies whether the Module is a Supervision Module (as required by ref.[1]). This is depicted in the UML model by an "*isSupervisionModule*" attribute of the *«ecoa.moduleType»* class.

ECOA Module Types are defined as part of a Component Implementation, in XML "*name.impl.xml*" files, using *<moduleType>* XML tags.

ECOA Module Implementations

An ECOA Module Implementation defines how a Module Type is to be implemented in software code, and is therefore depicted as a UML (concrete) class, stereotyped as an *«ecoa.moduleImplementation»*, and linked to its parent *«ecoa.moduleType»* with a UML realization association.

The ECOA Module Implementation definition is part of a Component Implementation, in XML "*name.impl.xml*" files, using *<moduleImplementation>* XML tags.

The ECOA Module Implementation definition includes attributes specifying:

a) the language this specific Module Implementation is to be coded in;
b) the activation method to be applied to the executable instances of the Module Implementation;

as well as the inherited (from the parent Module Type) "*isSupervisionModule*" attribute.

All ECOA Module Implementations must implement the standard Module Lifecycle operations defined by the ECOA, depicted in UML by the *«ecoa.moduleImplementation»* class specializing the (abstract) ECOA Module class.

The UML *«ecoa.moduleImplementation»* class depiction therefore reflects both the properties defined in the XML *<moduleImplementation>* statement, and the (implementation form) of the ECOA Module Lifecycle operations.

The UML *«ecoa.moduleImplementation»* class depiction therefore represents in a design model the physical realization of an ECOA Module in code.

ECOA Module Instances

A Module Instance, is a runtime instantiation of a Module Implementation, and is represented in a UML model as a role of a UML *«aggregation»* (aka *«composed of»* or *«has»*) association. Hence in Figure 7, the EO_IR *«ecoa.component»* is *«composed of»* the *EOIR_modPub_Instance* Module Instance, an instantiation of the *EOIR_modPub_Im «ecoa.moduleImplementation».*

ECOA Module Instances are defined as part of a Component Implementation, in XML "*name.impl.xml*" files, using *<moduleInstance>* XML tags.

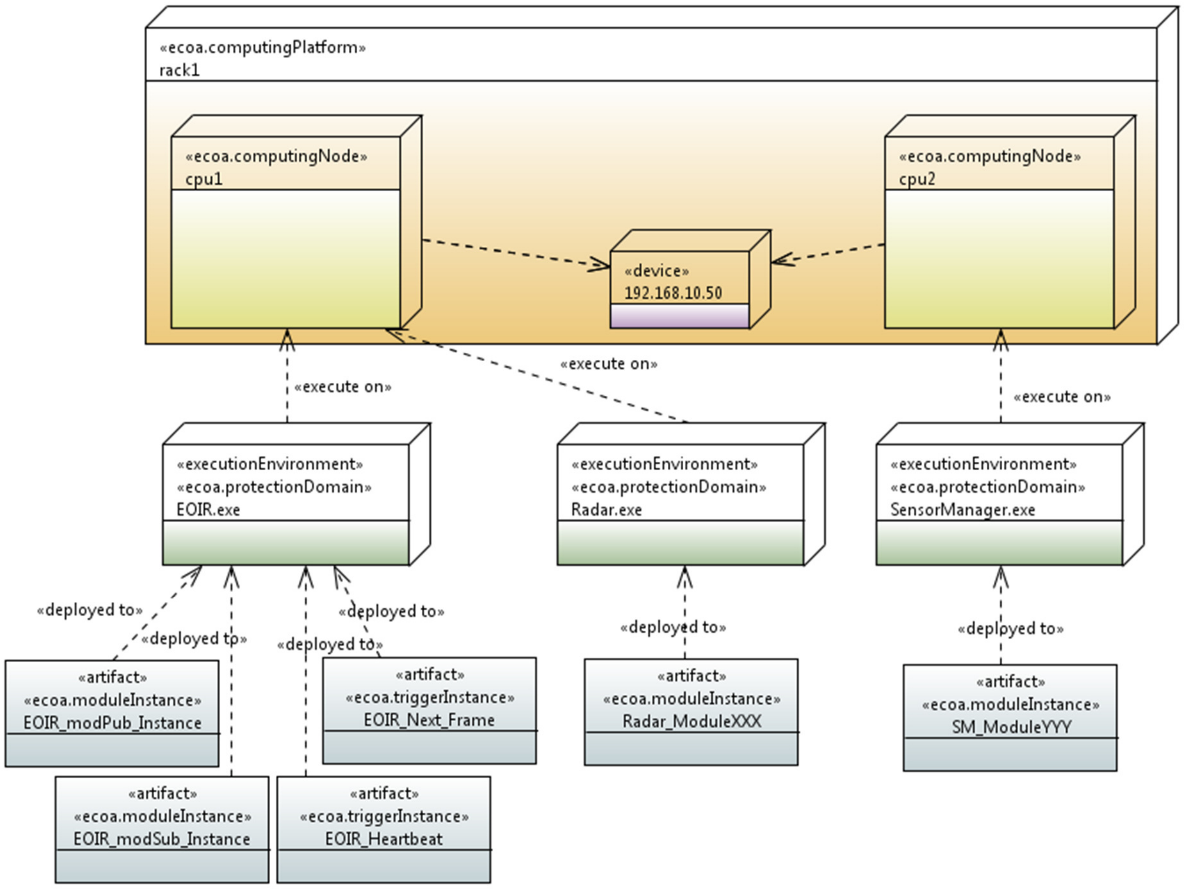
The XML Module Instance declaration states the predicted worst case execution time slot required by the Module Implementation code in order to complete its function, stated as a "*moduleDeadline*" attribute.

ECOA Integration & Deployment

Finally, to represent in UML the ECOA Deployment of a software system, we use UML Deployment Diagrams.

These comprise a number of UML Node, «*ExecutionEnvironment*» node, and «*Artifact*» node entities which are stereotyped for ECOA usage as «*ecoa.computingPlatform*», «*ecoa.computingNode*», «*ecoa.protectionDomain*», «*ecoa.moduleInstance*», and «*ecoa.triggerInstance*», as appropriate, and as illustrated in the example in Figure 8. Use of UML «*Device*» nodes on the Deployment Diagram allows the capture of Transport Binding information of the ECOA Integration (as illustrated).

Figure 8 Example ECOA (Step 5) Deployment Definition Expressed in UML



ECOA Deployments are defined in XML "*deployment.xml*" files, using *<deployment>* XML tags nesting *<protectionDomain>* declarations.

Transport Bindings are declared separately to the Deployment. For example, a UDP Transport Binding is declared in a "*udpbinding.xml*" file.

Summary

The following tables summarise the ECOA artefacts defined at each of the traditional ECOA Development Steps and their mapping to UML entities – where one or more ECOA specific stereotypes apply. The suggested colouring (where applicable) of each stereotyped UML entity is given (in braces).

Step 0 – Types

ECOA Artefact	***Stereotype(s)***	***UML Entity***
Predef Type	«ecoa.primitiveType»	Data Type (blue); Enumeration (grey)

Step 1 – Services

ECOA Artefact	***Stereotype(s)***	***UML Entity***
Service	«ecoa.service»	Interface (pink); Abstract Class (pink)
Event Operation	«ecoa.event»; «sent_by_provider»; «received_by_provider»	Operation
Request-Response Operation	«ecoa.request-response»	Operation
Versioned Data	«ecoa.versioneddata»	Property

Step 2 – Component Definitions

ECOA Artefact	***Stereotype(s)***	***UML Entity***
ASC (Component) (Type)	«ecoa.component»	Abstract Class (blue); Interface

Provided Service Relationship	«provides»	Interface Realization;
		Realization;
		Dependency

Referenced Service Relationship	«references»	Association (directional); Dependency
Service Instance	«ecoa.serviceInstance»	Interface (pink); Abstract Class (pink)
Provided Service (attribute)	«ecoa.service»	Port
Referenced Service (attribute)	«ecoa.reference»	Port

Step 3 – Assembly Definition

ECOA Artefact	*Stereotype(s)*	*UML Entity*
Assembly	«ecoa.assembly»	Class (blue); Abstract Class (blue)

Composite	«ecoa.composite»	Class (blue);
		Abstract Class (blue);
		Interface (blue)

Service	«ecoa.service»	Port
Reference	«ecoa.reference»	Port

Component	«ecoa.component»	Class (blue);
		Abstract Class (blue);

Service Link (Wire)	«serviceLink»	Dependency Association (directional)

Step 4 – Component Implementations

ECOA Artefact	*Stereotype(s)*	*UML Entity*
Module Type	«ecoa.moduleType»	Interface (green); Abstract Class (green)
Event Module Operation	«eventReceived»; «eventSent»	Operation (of an «ecoa.moduleType»)
Request-Response Module Operation	«requestReceived»; «requestSent»	Operation (of an «ecoa.moduleType»)
Versioned Data Module Operation	«dataRead»; «dataWritten»	Property (of an «ecoa.moduleType»)
Component	«ecoa.component»	Class (blue)
Module Implementation	«ecoa.moduleImplementation»	Class (blue)
Trigger Instance	«ecoa.triggerInstance»	Interface (pink)
Module Instance	-	Role of «aggregation»

Step 5 – Deployment

ECOA Artefact	*Stereotype(s)*	*UML Entity*
Computing Platform	«ecoa.computiungPlatform»	Node (orange)
Computing Node	«ecoa.computingNode»	Node (yellow)
Protection Domain	«ecoa.protectionDomain»	Execution Environment Node (green)
(Deployed) Module Instance	«ecoa.moduleInstance»	Artifact Node (blue)
(Deployed) Trigger Instance	«ecoa.triggerInstance»	Artifact Node (blue)
Transport Binding	«device»	Device Node (mauve)
Protection Domain Deployment	«execute on»	Dependency
Module/Trigger Instance Deployment	«deployed to»	Deployment Dependency

References

1	European Component Oriented Architecture (ECOA) Collaboration Programme: Architecture Specification (Parts 1 to 11) “ECOA” is a registered trade mark.
2	Service Component Architecture Assembly Model Specification OASIS, Version 1.1
3	OMG Unified Modeling Language (OMG UML) Object Management Group (OMG), Version 2.5 *http://www.omg.org/spec/UML/2.5*
4	Unified Modeling Language, The Addison-Wesley, 2005