European Component Oriented Architecture (ECOA) Collaboration Programme: Architecture Specification Part 7: Metamodel

BAE Ref No: IAWG-ECOA-TR-011
Dassault Ref No: DGT 144486-C

Issue: 3

Prepared by
BAE Systems (Operations) Limited and Dassault Aviation

Note: *This specification represents the output of a research programme and contains mature high-level concepts, though low-level mechanisms and interfaces remain under development and are subject to change. This standard of documentation is recommended as appropriate for limited lab-based evaluation only. Product development based on this standard of documentation is not recommended.*

Contents

Figures

0 Introduction

This Architecture Specification provides the definitive specification for creating ECOA-based systems. It describes the standardised programming interfaces and data-model that allow a developer to construct an ECOA-based system. The details of the other documents comprising the rest of this Architecture Specification can be found in Section 3.

This document is Part 7 of the Architecture Specification, and contains the Metamodel and XML schema definitions for an ECOA system.

The document is structured as follows:

- Section 6 describes the ECOA metamodel;
- Section 7 provides legality rules to follow when writing ECOA XML files;
- Section 8 details of v1.11.0 of the schemas
- Section 9 specifies the SCA subset selected for ECOA.

1 Scope

This purpose of this Architecture Specification is to establish a uniform method for design, development and integration of software systems using a component oriented approach.

2 Warning

This specification represents the output of a research programme and contains mature high-level concepts, though low-level mechanisms and interfaces remain under development and are subject to change. This standard of documentation is recommended as appropriate for limited lab-based evaluation only. Product development based on this standard of documentation is not recommended.

3 Normative References

Ref	Description
Architecture Specification Part 1	
	IAWG-ECOA-TR-001 / DGT 144474
	Issue 3
	Architecture Specification Part 1 – Concepts
Architecture Specification Part 2	
	IAWG-ECOA-TR-012 / DGT 144487
	Issue 3
	Architecture Specification Part 2 – Definitions
Architecture Specification Part 3	
	IAWG-ECOA-TR-007 / DGT 144482
	Issue 3
	Architecture Specification Part 3 – Mechanisms
Architecture Specification Part 4	
	IAWG-ECOA-TR-010 / DGT 144485
	Issue 3
	Architecture Specification Part 4 – Software Interface
Architecture Specification Part 5	
	IAWG-ECOA-TR-008 / DGT 144483
	Issue 3
	Architecture Specification Part 5 – Platform Requirements
Architecture Specification Part 6	
	IAWG-ECOA-TR-006 / DGT 144481
	Issue 3
	Architecture Specification Part 6 – ECOA Logical Interface
Architecture Specification Part 7	
	IAWG-ECOA-TR-011 / DGT 144486
	Issue 3

	Architecture Specification Part 7 – Metamodel
Architecture Specification Part 8	
	IAWG-ECOA-TR-004 / DGT 144477
	Issue 3
	Architecture Specification Part 8 – C Language Binding
Architecture Specification Part 9	
	IAWG-ECOA-TR-005 / DGT 144478
	Issue 3
	Architecture Specification Part 9 – C++ Language Binding
Architecture Specification Part 10	
	IAWG-ECOA-TR-003 / DGT 144476
	Issue 3
	Architecture Specification Part 10 – Ada language Binding
ISO/IEC 8652:1995(E) with COR.1:2000	
	Ada95 Reference Manual
	Issue 1
ISO/IEC 9899:1999(E)	Programming Languages – C
ISO/IEC 14882:2003(E)	Programming Languages C++

4 Definitions

For the purpose of this standard, the definitions given in Architecture Specification Part 2 and those shown below apply.

5 Abbreviations

API	Application Programming Interface
ARINC	Aeronautical Radio, Incorporated
ASAAC	Allied Standards Avionics Architecture Council
ASC	Application Software Component
COTS	Commercial Off-The-Shelf
CPU	Central Processing Unit
DDS	Data Distribution Service
ECOA	European Component Oriented Architecture
ELI	ECOA Logical Interface
EUID	ECOA Unique Identifier (ID)
FIFO	First In, First Out
HR	High Resolution

ID	Identifier
IMA	Integrated Modular Avionics
IoC	Inversion-of-Control
IP	Internet Protocol
LRU	Line Replaceable Unit
NaN	Not a Number
OS	Operating System
PC	Personal Computer
POSIX	Portable Operating System Interface
QoS	Quality of Service
RFC	Request For Comments
RT	Real Time
RTOS	Real-Time Operating System
SOA	Service-oriented Architecture
SW	Software
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
UML	Unified Modeling Language
UTC	Coordinated Universal Time
VME	Versa Module Europa (bus)
XML	eXtensible Markup Language
XSD	XML Schema Definition

6 ECOA Meta Models

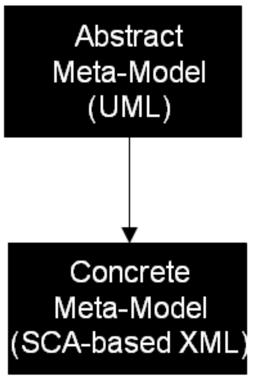

Figure 1 – ECOA Meta-Models

The structure of an ECOA system has been specified as an abstract meta-model which describes the ECOA system's data elements and their relationships. UML entity-relationship diagrams have been used to present the model information which can be found in Section 0.

Ultimately the requirements to exchange components, and automatically instantiate systems from them, require a precisely-specified and machine-readable version of the model. This is known as the concrete meta-model and the implementation is based on the open standard Service Component Architecture (SCA). Section 6.2 defines the concrete meta-model.

It is envisaged that the ECOA implementers will ultimately develop tool support that enables ECOA information to be captured in high-level design tools that support, for example, UML. However, the SCA-based concrete meta-model will remain the standard for exchange of information (e.g. between component suppliers and system integrators).

6.1 Abstract Meta-Model

6.1.1 Overview

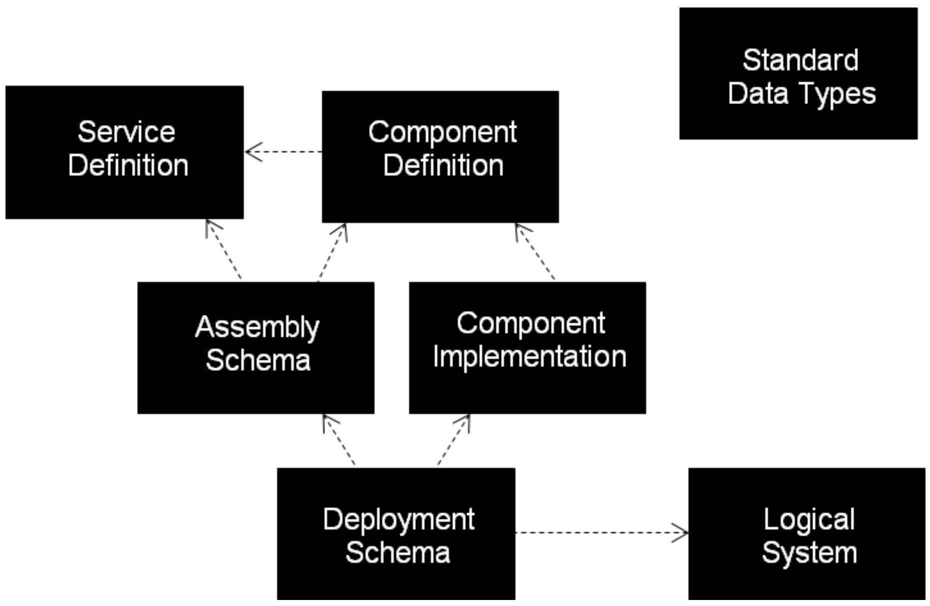

Figure 2 – Overview of Meta-Model

The following sections detail the ECOA abstract meta-model. They aim to provide a definition of all the concepts and objects that need to be formalized to describe an ECOA system. This abstract meta-model is designed using a set of self-sufficient views of a UML model; each view of the meta-model is describing one given concept. An overview of this model is shown in Figure 2 which describes overarching dependencies between main ECOA concepts.

An Assembly Schema (see section 6.1.4) describes the structure of the ECOA system, in a hardware independent fashion. It does this by describing service links between the Application Software Components. These are uni-directional connections between one service provided by an Application Software Component and another service required by an Application Software Component. Both services share the same full Service Definition and have a compatible Quality of Service (QoS). The service is offered or required as a whole, where a client has access to the whole set of service operations of the Service Definition.

An ECOA system is fully formalized within a Deployment Schema (see section 6.1.8) which details how the Application Software Components of the Assembly Schema are deployed on a Logical System: it specifies how modules of each Application Software Component are mapped onto logical processor nodes. A Logical System (see also section 6.1.8) describes a set of hardware computing resources and their physical connections.

The client defines an expected QoS for each required service (service-level and operation-level), and the provider defines an actual QoS for each provided service (service-level and operation-level). The expected and actual QoS need to be compatible for a service link to be established between both services. As a service is a collection of operations, a service link is, at technical level, implemented by a collection of module-level operation links between component modules. These operation links fulfil the rules of direction and multiplicity implied by the service link.

The Component Definition (Section 6.1.3) captures the interface of an Application Software Component and is formalized in terms of the services that are required and the services that are provided by an Application Software Component.

The interface of an Application Service, called Service Definition, is described as a set of operation signatures (see section 6.1.2).

Each Data Type (see section 6.1.7) used at the Service Definition or Component Definition level must be described.

A Component Instance is the software instantiation within the ECOA System of a given Component Implementation. A Component Implementation is the software realization of a Component Definition to which it conforms.

A Component Implementation (see section 6.1.5) is described within the ECOA System in terms of:

- Its Component Definition, which is, in SOA terms, the “Component Contract” to which it conforms,
- Its internal design, which is made of Modules, and Operation Links between Modules or Service Definitions of the component.

The concept of Module is defined (section 6.1.5) as a software entity implementing a given part of the ECOA Component Implementation. Operations in one Module may interact with Operations in another module via standard ECOA mechanisms.

Within a Component Implementation, Modules are linked together, at operation level, and are linked to the operations of the Services of the Component Implementation using Operation Links (see section 6.1.6). The Modules are the software entities that have to be deployed in the Deployment Schema.

The rules specified in Section 6.2.1.1 apply to the names for operations, component implementations and module implementations.

Note: The abstract meta-model is described with UML class diagrams which use the following conventions:

- The default multiplicity for any link between two entities is 1.
- A grey-filled class indicates a reference to another class diagram in which the mentioned entity is described.

6.1.2 Service Definition

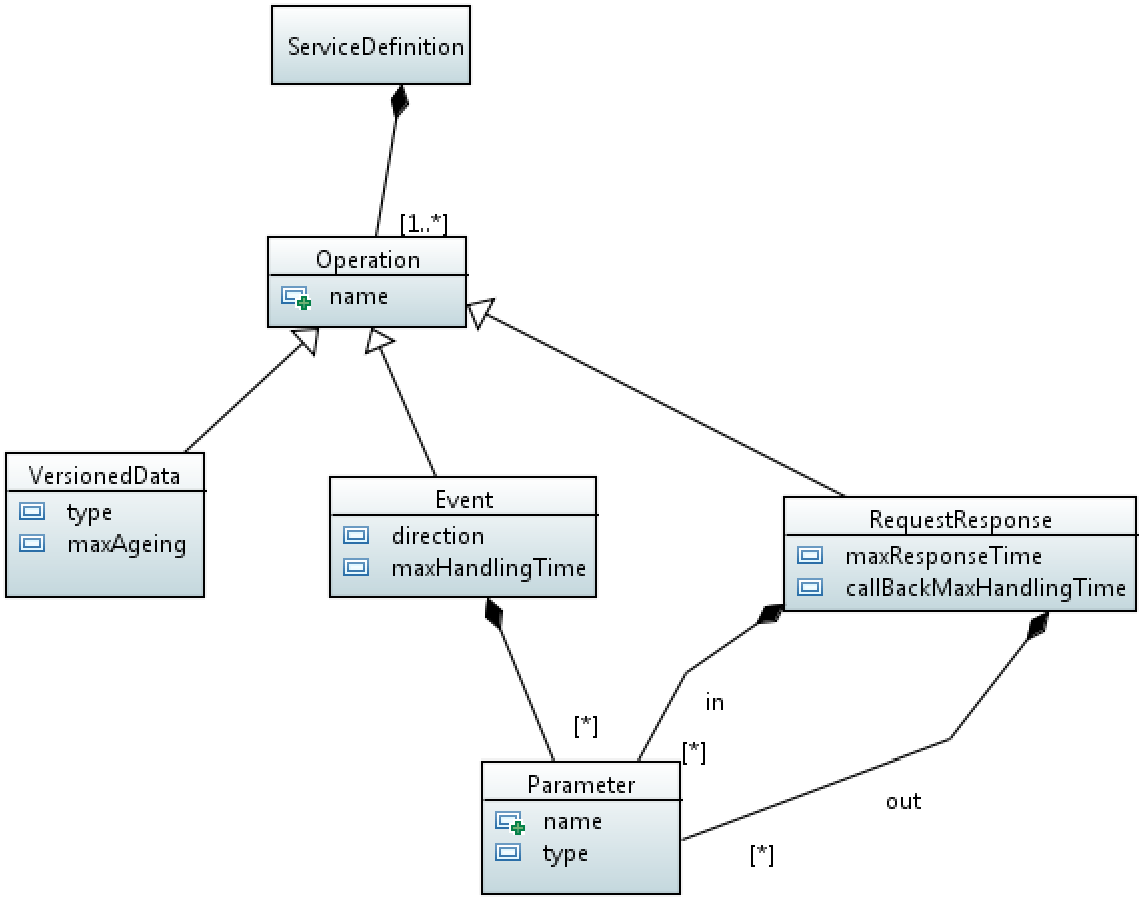

Figure 3 – ServiceDefinition meta-model

A **ServiceDefinition** is a set of **Operations** and **QoS Specifications**. An **Operation** is either, a **VersionedData Operation**, an **Event Operation** or a **RequestResponse Operation**. A ServiceDefinition contains at least one Operation.

For a VersionedData Operation in a ServiceDefinition, the data is published by the Application Software Component that provides the service.

An Event Operation in a ServiceDefinition has a direction: either received_by_provider or sent_by_provider.

A RequestResponse Operation in a ServiceDefinition is initiated by the Application Software Component that requires the service.

Each Service Operation may be qualified by specific QoS attributes is given hereafter. Unit is in seconds (s).

These attributes are requirements at component level and provide hypothesis for the internal design of the component. Internal design choices are under the responsibility of the Component Supplier while the Service Operation QoS attributes are initially defined by the System Designer.

Table 1 – Specific QoS attributes on operations

	Provided service	Required service
Event	**IN** **MaxHandlingTime** = maximum duration between event receipt and end of related processing	
	OUT **MaxHandlingTime** = specifies an intent on receivers for maximum duration between event receipt and end of related processing	
Request-Response	**IN (request_received)** **MaxResponseTime** = maximum duration between request receipt and response sent	**OUT (request_sent)** **MaxResponseTime** = maximum duration between request sent and response receipt In case of an asynchronous request-response, **callBackMaxHandlingTime** = maximum duration between response receipt and end of related processing
Versioned Data	**Published** **MaxAgeing** = maximum duration between data production (from the source) and the end of the writing process + (Worst MaxAgeing of all the consumed versioned data used for producing the data)	**Consumed** **MaxAgeing** = maximum duration between data production (from the source) and the end of the reading process + (Worst MaxAgeing of all the consumed versioned data used for producing the data)

Data ageing will take into account all the transformations processed through the components chain by analysing components behaviours (see section 6.1.9.1). In other words, data ageing will effectively be cumulative across a functional chain in order to make sense for endpoint consumers of the versioned data.

In addition to these attributes specific to each type of operation, two common QoS attributes are specified and applicable for each operation (independent of its type): the HighestRate and the LowestRate. They are based on the notion of slipping rate.

A slipping rate is defined with the help of two numbers:

- A time frame TF which is a time duration,
- A number N of operation calls which occur during the time frame.

The slipping rate expresses that, at any time, during the previous time frame (interval of time between the present time and the present time minus the time frame), exactly N operation calls have occurred.

As application of that notion, the HighestRate specifies the maximum number of occurrences of the operation within a specified time frame. If the number of occurrences is 1, the time frame corresponds to the minimum inter-arrival time between operations. The HighestRate helps to size input queues and to define processing activation laws within the component.

The LowestRate specifies the minimum number of occurrences of the operation within a specified time frame. If the number of occurrences is 1, the time frame corresponds to the maximum inter-arrival time between operations. The LowestRate helps to define the minimal processing activation laws within the component.

It is recommended to use the same time frame to express the HighestRate and the LowestRate to ease comparison between both.

For data, these rates express refreshment period requirements.

For an output event, an occurring rate can be defined to compare the receivers accepted input rates: this allows consistency checking between production and consumption rates.

For R/R replies, it is assumed that the R/R reply follows the same laws as the R/R request.

A comment can be added to describe each operation.

A single Module Operation may invoke many Container Operations during its execution, which introduces dependencies between the rates identified for those operations. Behaviour specifications at Service and Module level have been identified to allow these dependancies to be captured - see section 6.1.9.1.

All QoS attributes on operations are currently optional.

A ServiceDefinition also includes service-level specifications of Quality of Service (QoS) parameters (such as encryption level). These QoS specifications are used when matching up provided and required ServiceDefinition variants when creating ServiceLinks (section 6.1.6).

Note: the definition of service-level attributes requires additional work.

6.1.3 Component Definition

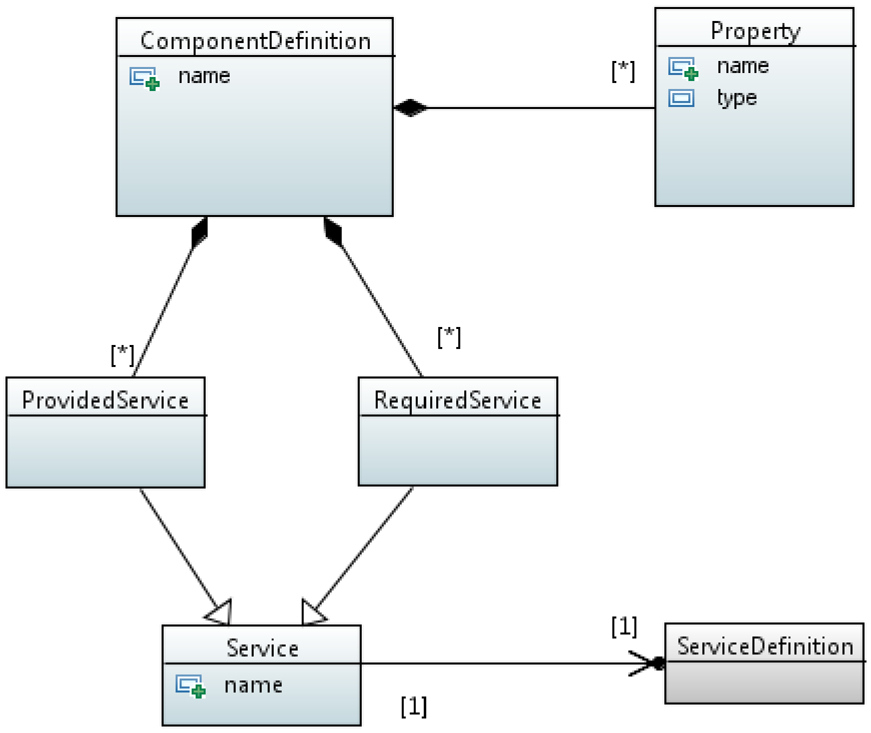

Figure 4 – ComponentDefinition meta-model

A **ComponentDefinition** is a set of **Properties**, **ProvidedServices** and **RequiredServices**. A ProvidedService or a RequiredService references a ServiceDefinition shared by ComponentDefinitions.

A ComponentDefinition must contain at least either a ProvidedService or a RequiredService.

Note: Elements of Behaviour modelling need to be added to the ComponentDefinition: rules for each operation, complexity/processing time estimations, required service usage rules, relationship between provided and required services, definition of states/modes, etc. See section 6.1.9.1.

6.1.4 Assembly Schema

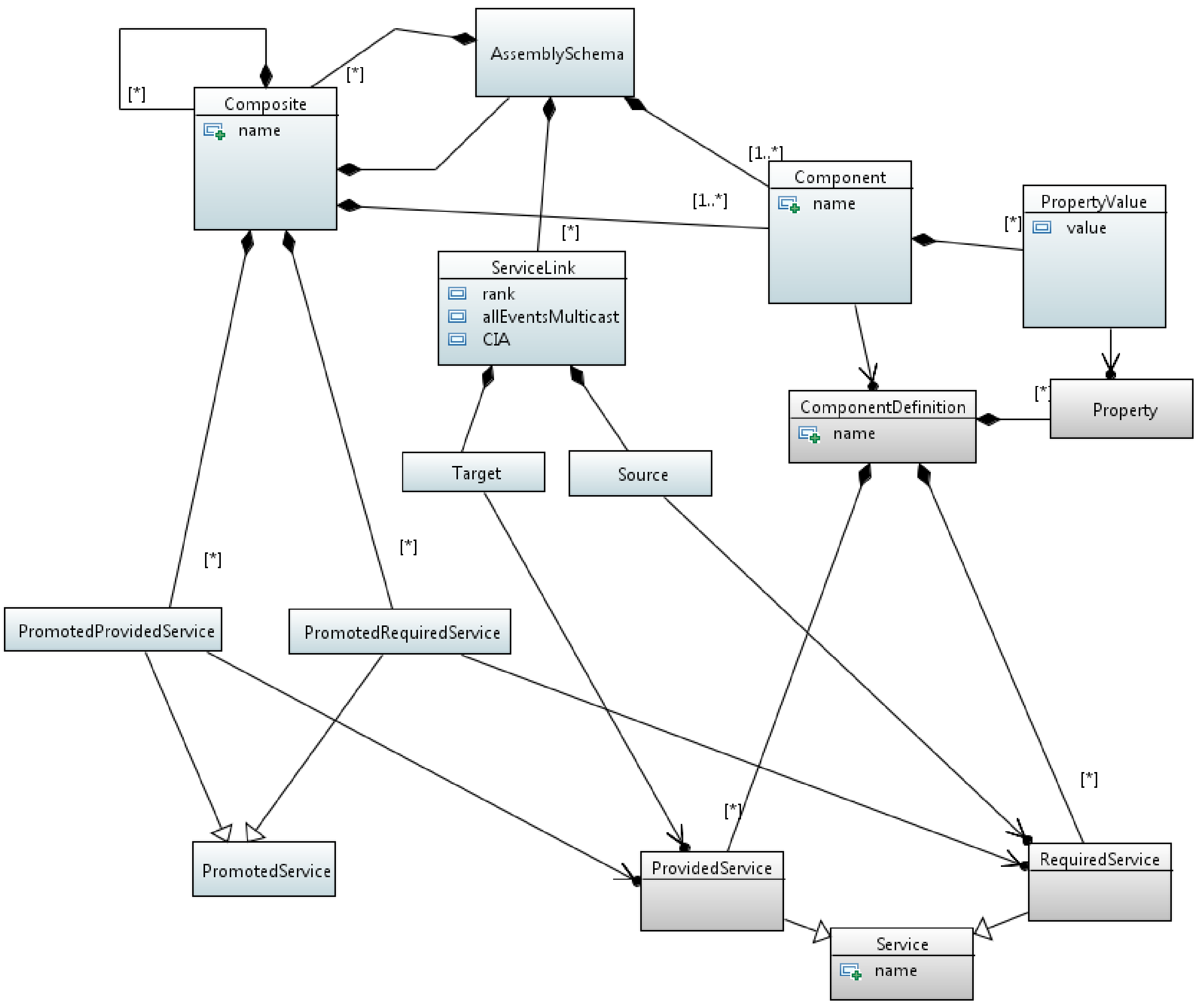

Figure 5 – AssemblySchema and ServiceLink meta-model

An **AssemblySchema** describes the structure of an ECOA system, independently of its physical deployment on hardware platforms. Note that grey in figure above means that the greyed element is more precisely defined in another subsection.

An Assembly is made of **Composites**, **Components** and **ServiceLinks**.

A Composite looks like a component; its definition is provided by a dedicated AssemblySchema and promotion links between its provided or required services and the provided or required services of its internal components. Promotion links are a logical concept to master complexity through hierarchical assembly schemas. They have no existence at IT or technical levels: the assembly schema actually deployed is the one containing only components.

A Component instantiates a ComponentDefinition in a given system. It has a set of instantiation parameters known as Properties defined at ComponentDefinition level.

A ServiceLink connects Components together via provided and required service references. Each ServiceLink connects one ProvidedServiceReference (target) to one RequiredServiceReference (source) each of which refers to a Component and to a RequiredService or ProvidedService of this component's ComponentDefinition (targets and sources are "technical" objects which are introduced to model ternary associations, without introducing the specific UML notation).

An attribute named **rank** can be associated to a ServiceLink. It is the way the System Integrator indicates a preference for an instance of a service provider over another when both are connected to the same service required. The lower the numerical value of rank, the higher the preference for the link. The rank is mandatory. An additional Boolean attribute **allEventsMulticast** indicates if all event operations of the service definitions are sent on this ServiceLink in a systematic way.

6.1.5 Component Implementation

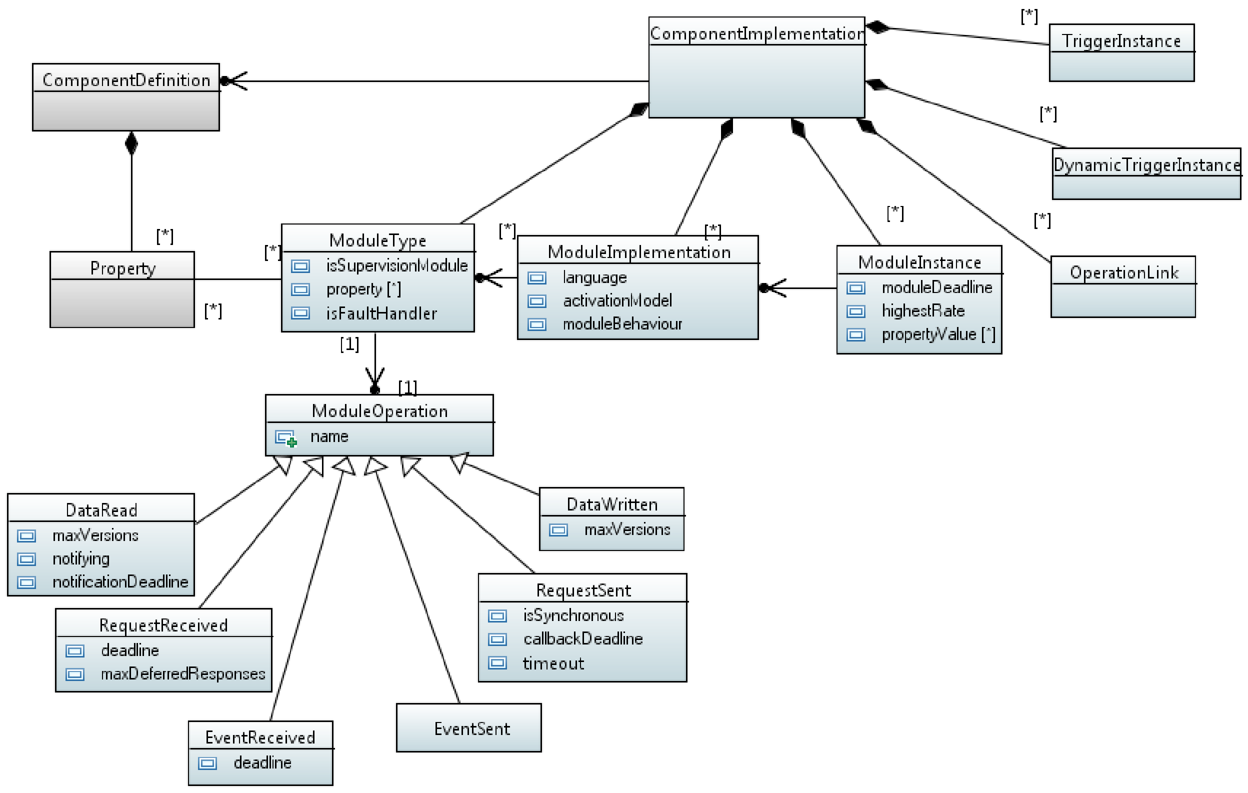

Figure 6 – ComponentImplementation meta-model

A software realization of a ComponentDefinition is described by a **ComponentImplementation**.

A ComponentImplementation gives information to secure the integration of Application Software Components in a system, possibly sharing execution platforms, and to make early verification of the system possible.

A ComponentImplementation is made of **ModuleTypes**, **ModuleInstances**, **TriggerInstances**, **DynamicTriggerInstances** and **OperationLinks**.

A ModuleImplementation corresponds to a piece of software implementing in a certain programming language a given part of the Component Implementation that must be executable in a single thread (no parallelism, no internal synchronisation). A behaviour describing worst case execution paths may be associated to a ModuleImplementation (see 6.1.9.3). A ModuleInstance corresponds to the instantiation of a given **ModuleImplementation**.

A ModuleType defines the interface of a ModuleImplementation in terms of ModuleOperations at module-level or in terms of properties. These operations correspond to the same exchange mechanisms used by Application Software Components (i.e. data, event; request-response), but have a direction (reflecting the module's point of view), and have additional attributes over and above the service defined ones.

Each DataRead operation that indicates the module is a reader of a versioned data relies on the following attributes:

- maxVersions: the maximum number of versions that the module may access in read mode,
- notifying: if 'true', a callback is generated by the platform tooling and is called by the container each time it is aware of a data update.

Each RequestReceived operation which is the entry-point called on the receiving of a request relies on the following attribute:

- maxConcurrentRequests: the maximum number of R/R IDs that the module may retain for that entry-point before sending the associated replies, regardless of incoming requestLinks related to that entry-point. If this number is reached, additional R/R cannot be retain and are discarded by the Container.

Each RequestSent operation which indicates the module may send a request towards another module relies on the following attributes

- timeout: the maximum time during while the module is blocked waiting for a reply. If the timeout is set to -1, the R/R is an indefinite blocking call.
- isSynchronous: if true, the R/R is synchronous: the call blocks the calling module until the receiving of the response or the expiration of the timeout.
- callbackDeadline: the deadline associated to the callback called when the response arrives on the client side.

Each DataWritten operation that indicates the module is the writer of a given versioned data relies on the following attribute:

- maxVersions: the maximum of versions that the module may access in read-write mode.

The operations' names will appear in the module's container API. Each operation name shall be unique for a given moduleType definition. By annotating a module as a supervision one, a module may support system-level capabilities such as error handling or module lifecycle management. By annotating a module wih the attribute IsFaultHandler, the module is considered as an Fault Handler and it may support fault management capabilities such as infrastructure error notifications or recovery actions.

An additional QoS attribute defines the deadline for each operation (event, request, callback, notification). This deadline allows analyzing the schedulability of operations at design time. It also allows the container to check that the temporal behaviour of the operation at execution time is the expected one.

A ModuleInstance corresponds to an instance of a ModuleImplementation which itself is of a defined ModuleType. A ModuleInstance has its own internal state and has an assigned module deadline (used to determine its execution priority). The module deadline attribute is defined in ModuleInstance which is calculated through considering the response times of operations. The default activation model of the Module Instance is the reactive model; meaning the container activates it as long as there are incoming operations for it. An alternative activation model is the rhythmic model; meaning the container activates it every unit of its deadline, and only if there is an incoming operation. This will be further described in the final release of this document and is included here for completeness.

The notion of ModuleInstance provides the ability to instantiate, a number of times, the same software code in multiple execution contexts (e.g. different module deadline or execution node) inside an ECOA Application Software Component.

A TriggerInstance is similar to a ModuleInstance, except that it is dedicated to producing periodic events: it has no module type; it does not need to be implemented, as the periodic events will be generated automatically by the infrastructure.

A DynamicTriggerInstance is a trigger that generates non periodic events. The delay between the generation of two events can be dynamically be set at runtime. As for the TriggerInstance, the DynamicTriggerInstance is generated by the infrastructure.

Each Property at ComponentDefinition level is implemented within one or more ModuleTypes of the ComponentImplementation.

The internal structure of a given ComponentImplementation must be specified in terms of OperationLinks.

6.1.6 Operation Links

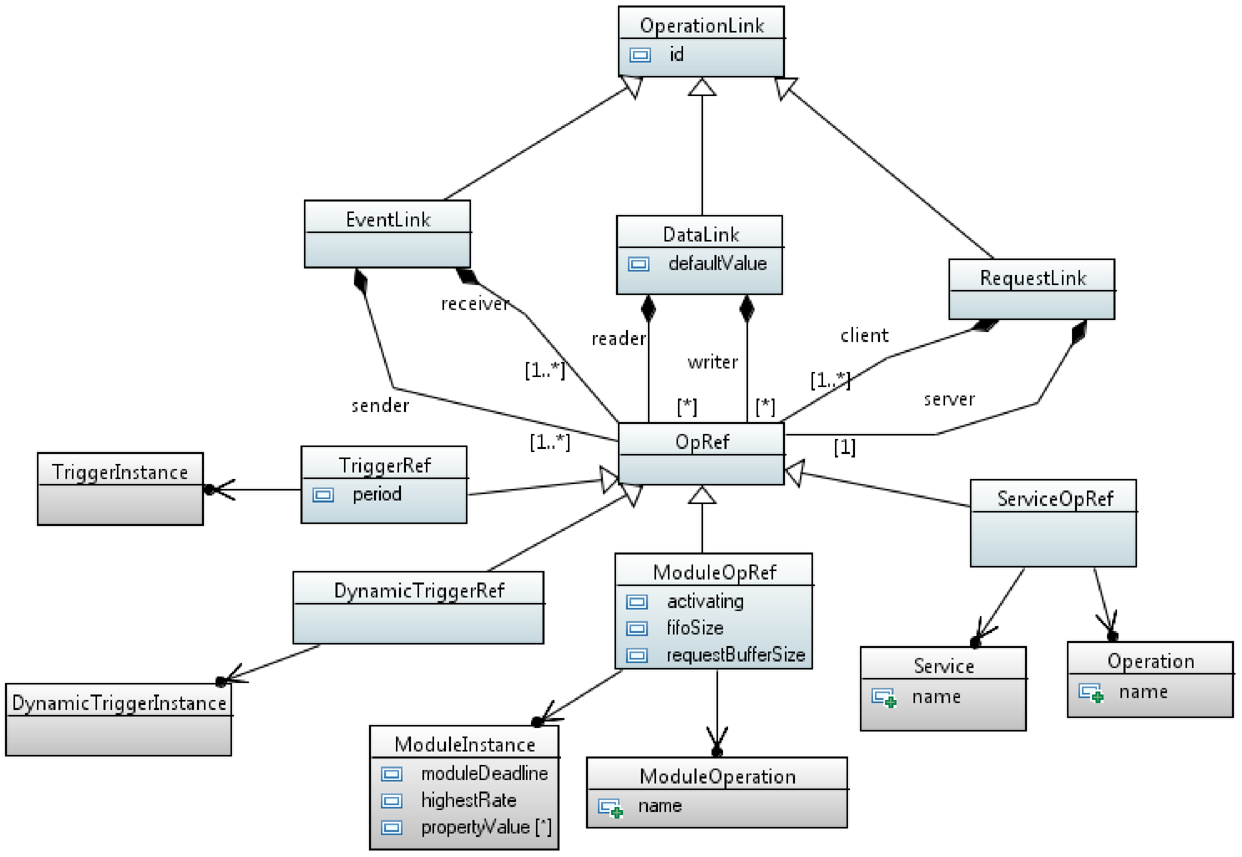

Figure 7 – OperationLink meta-model

The **OperationLinks** describe the interactions/synchronisations between Modules within the same Application Software Component. An OperationLink is a “star-like” connection linking “internal” (module-level) operations. Inter-Module interactions are specified using **DataLink**, **EventLink** or **RequestLink**, depending on the kind of module operations that are linked. These three kinds of links are oriented and have different possible multiplicities:

- A DataLink may have n writers and p readers (a unique data-writer is recommended but not mandatory),
- A RequestLink has one server and p possible clients,
- An EventLink can have n possible senders and p possible receivers of the event.

Note that the cardinality of the requestLink only concerns module internals. To implement redundant servers at component level, multiple service links can be defined between multiple client components and multiple server components.

An internal module operation is referred to by a **ModuleOpRef**, which refers to a ModuleInstance and one of its ModuleOperations. Attributes are associated to a ModuleOpRef to define:

- if the operation is activating or not (activating),
- the maximum number of waiting operation calls for this operation (fifoSize) – waiting means the operation calls have not been taken out of the infrastructure,

- and the maximum number of pending requests (requestBufferSize) that can be simultaneously processed by the server ModuleInstance for that particular ModuleOpRef – pending means the requests have been taken ouf of the infrastructure but the replies have not been sent yet to the client.

As a consequence, the maxConcurrentRequests value of the requestReceived entry point refered to by the ModuleOpRef shall be greater or equal to the sum of requestBufferSize values for all incoming requestLinks linked to that entry point (i-e requestLinks being defined within the component).

Internally to the component, an operation of the service is referred to by a **ServiceOpRef**, which refers to a ProvidedService or RequiredService, and one of the Operations of its ServiceDefinition.

The purpose of the TriggerInstance is to define a periodic event generator internally to the component scope: the TriggerInstance will act as the sender of the event, at the specified period. The generator is handled by the container (e.g. an OS watchdog or an auto-generated invisible module which sends an event). This avoids the creation of event generation components which will break the inversion of control principle, as they will need to access to the OS to generate periodic events. This system allows the creation of several flows of periodic events in a synchronised way (if all events come from the same TriggerInstance), or in a non-synchronised way (if they come from different TriggerInstances). It also allows combining a periodic source of events with other, non-periodic sources.

The purpose of the DynamicTriggerInstance is to define a one-shot event generator internally to the component scope: the DynamicTriggerInstance will act as a sender of a valued event, within a given delay specified at runtime. The principle is to receive an event, named “in” event hereafter, and to send after a given delay an associated event, named “out” event. The first parameter of the “in” event is the delay. Other parameters can be any of the ECOA types. Multiple occurrences of the same event can be queued waiting for the delays to expire. A « reset » operation can purge all waiting event occurrences.

Note that:

- The same RequestReceived, EventReceived or EventSent operation of a module can be part of different ModuleOperationLinks at the same time. All other module operations (DataRead, DataWritten and RequestSent operations) exist in only one OperationLink.
- Each DataLink is associated to a Data that it represents and that is shared within the Application Software Component. It may connect several writers that can be component-internal writes or references. In any case, a reader gets access to the most recent value accessible on the platform.

6.1.7 Data Types

Data types are “portable types” and are only used to describe information transmitted on wires between components and operation links between modules. By using these types, information can be then serialized for transmission with the help of the ELI (Architecture Specification Part 6). The way they are physically bound to a given processor is left to the platform provider based on language bindings (Architecture Specification Part 4, Architecture Specification Part 8, Architecture Specification Part 9 and Architecture Specification Part 10).

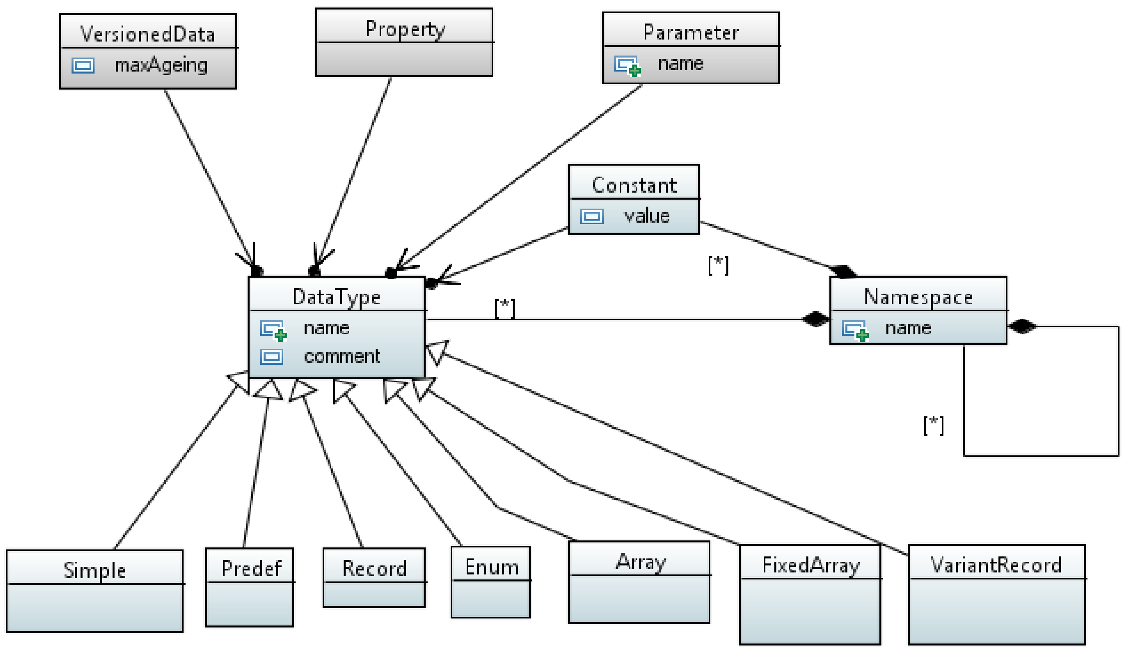

Figure 8 – DataType definition

A **DataType** is a language-neutral type definition. It is used as a shared definition, to help define ServiceDefinitions: it is referenced by VersionedData, and by Parameters of Events and Operations. It is also used to type Properties. A comment can describe the DataType.

A DataType definition describes the **NameSpace** in which it is located. A NameSpace is composed of DataTypes and NameSpaces, the different types are described in Figure 9 and Figure 10.

A **Constant** is a remarkable integer or floating-point value, identified with a name and located in a given NameSpace. A Constant may be of any of the Simple or Predef DataTypes.

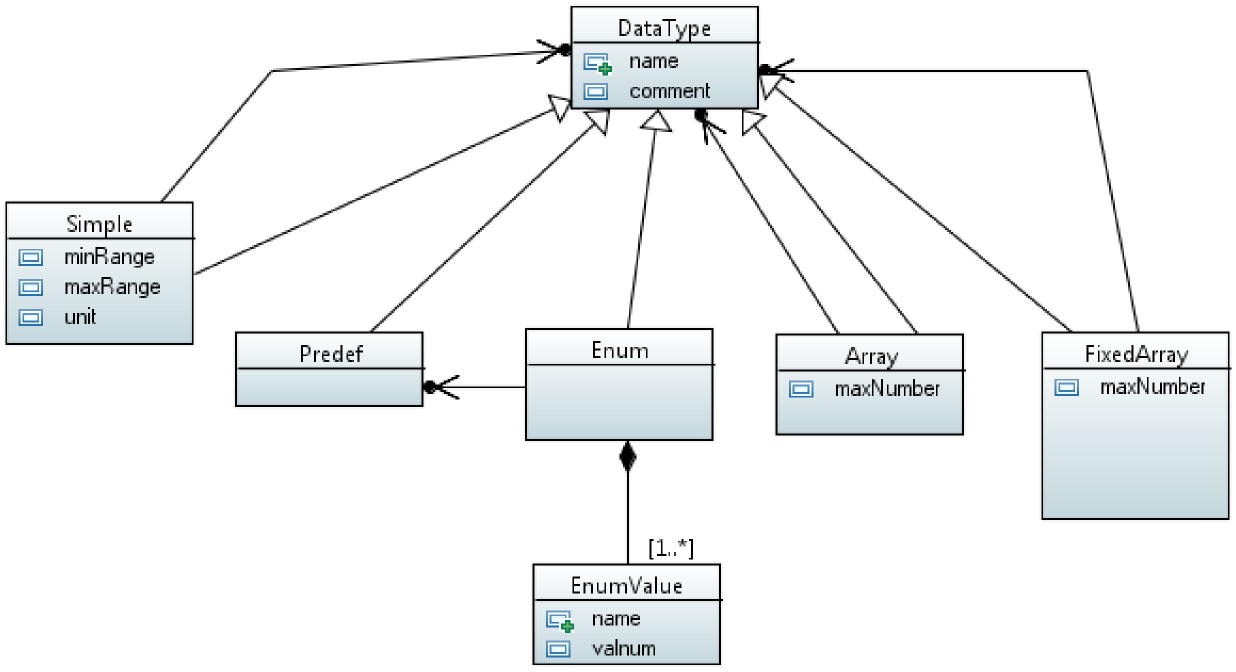

Figure 9 – Supported Data Types

A **Predef** type belongs to a fixed list of predefined types[1]: boolean8, char8, byte, int8, int16, int32, int64[2], uint8, uint16, uint32, uint64, float32, double64. 8-bit characters are encoded in ASCII. For boolean8, the value 1 means TRUE while the value 0 FALSE. float32 and double64 are IEEE754 compatible.

A **Simple** type is defined to give a meaningful name to a predef or another simple type. It can define range limits and a unit. Each limit can be a literal numeric, or a reference to a symbolic constant. The unit is functional and expressed as a string (e.g. 'second').

An **Enum** type is based on a predef type and defines the list of authorized values, **EnumValue**, each of which has a symbolic name. Each value can be a literal numeric, or a reference to a symbolic constant.

An **Array** defines a variable-capacity array, whose maximum capacity is fixed. All elements are of the same type. The maximum capacity can be a literal numeric, or a reference to a symbolic constant.

A **FixedArray** defines a fixed-capacity array. All elements are of the same type. The capacity can be a literal numeric, or a reference to a symbolic constant.

[1] The list of available types may be extended in the future as requirements evolve. For example, fixed-point types may be required.

[2] 64bit types may not be supported on every ECOA platform.

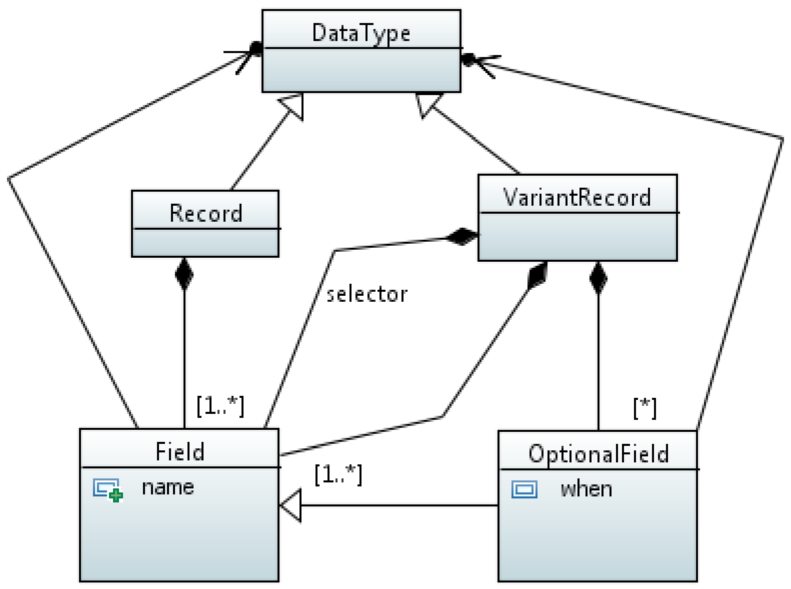

Figure 10 – Records and VariantRecords

A **Record** is a structure with named **Fields**, of any type.

A **VariantRecord** is like a Record, with a special field called the selector (of boolean, integer or enum type). Some of the fields, **OptionalField**, of a VariantRecord are optional: they are valued only when the selector has a certain value (given by the attribute "when").

Note that nested types are not allowed; i.e. it cannot be possible to define local types specific to a given field. All types used at field level must be defined prior to the record definition.

6.1.8 Deployment Schema and Logical System

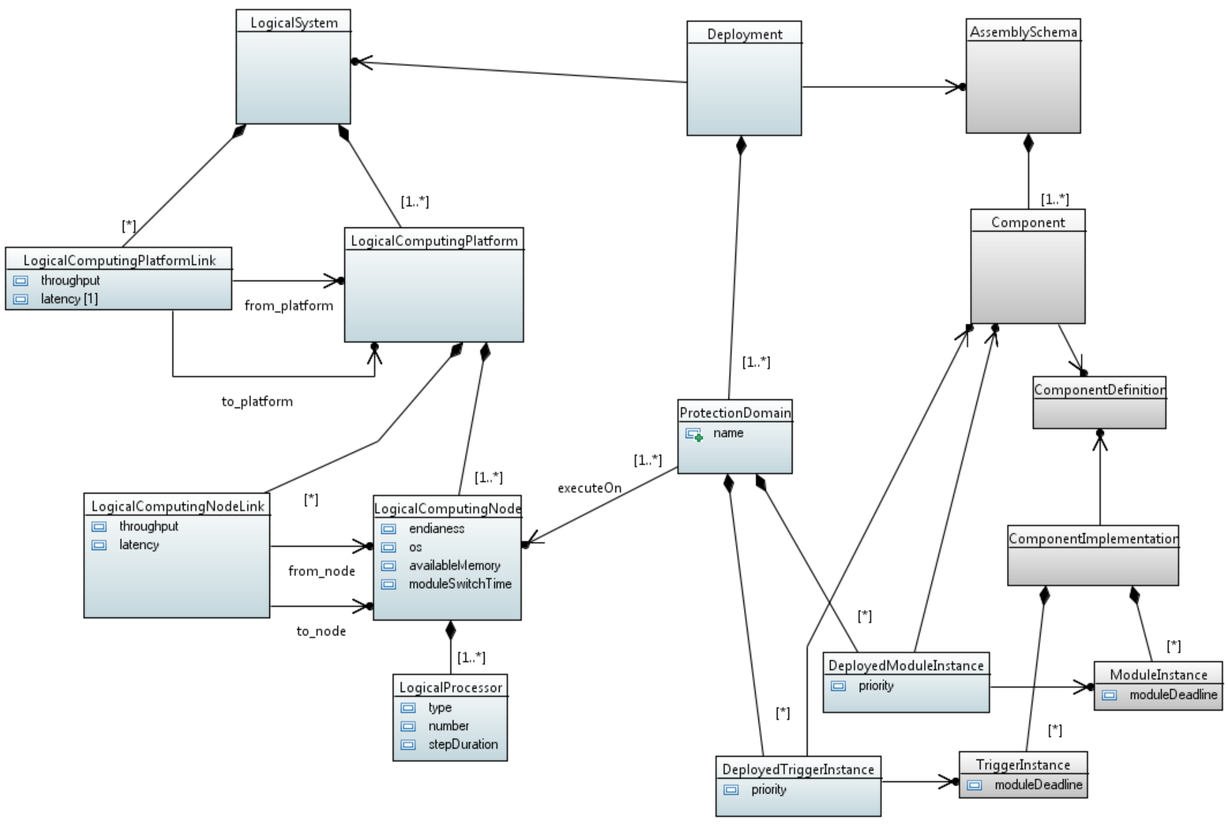

Figure 11 – Deployment Schema

A **Deployment Schema** refers to an Assembly Schema.

It contains the mapping of **ProtectionDomains** on **LogicalComputingNodes**.

A ProtectionDomain offers spatial isolation (memory protection), and possibly also temporal isolation (e.g. ARINC 653 partition scheduling), on a given LogicalComputingNode. It corresponds to the concept of process or partition, depending on the OS used.

Each ProtectionDomain hosts a number of ModuleInstances (which are referenced by the DeployedModuleInstance objects). All names of ModuleInstances hosted by a ProtectionDomain shall be unique within the ProtectionDomain scope.

The **priority** attribute is defined in DeployedModuleInstances which is calculated based on ModuleInstance ModuleDeadline and rate through considering the response times of operations. This will be further described in the Developer's Guide and is included here for completeness.

A LogicalComputingNode allows early verification of the performance of a system by providing an idealised model of a set of processors. This ideal processing resource is parameterised by a number of key attributes such as computing step, memory capacity, module switch time and number/standard of processors.

The initial model of a logical computing node chosen at this stage of the architecture definition is as a symmetric multiprocessor hosting one single OS image. It contains sets of **LogicalProcessors**; these may be heterogeneous if the OS provides an abstract interface. LogicalComputingNodes are linked together through **LogicalComputingNodeLinks** and they constitute a **LogicalComputingPlatform**. LogicalComputingPlatforms may then be linked together through **LogicalComputingPlatformLinks** and they constitute a **LogicalSystem**.

The mapping from logical computing nodes to actual physical processors (or cores) is not defined in this issue of the document. It may be that it is not the same as the mapping from LogicalComputingNodes to LogicalProcessors. For example one physical processor or core might be used instead of multiple LogicalProcessors or vice-versa. The mapping of logical links onto actual physical buses is not also addressed. These mappings are provided through specific artefacts supplied by the platform provider.

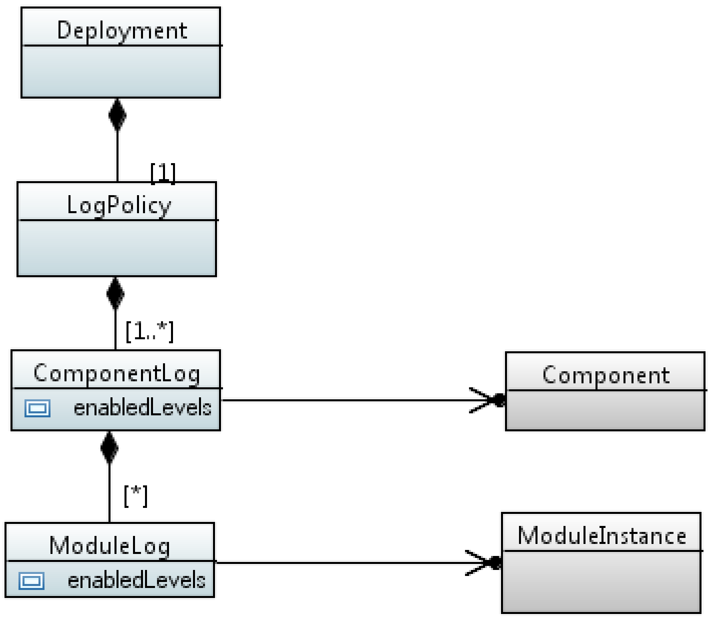

Figure 12 – Log Policy Definition

6.1.9 Behaviour Specification

This section is an initial proposal to specify the behaviours at component level, service level and at module level. Its maturity is low. It will be exercised and refined in future stages.

6.1.9.1 At component level

Component behaviour is used to define how a component behaves within a system. It will describe how it makes use of its required services based upon how its own provided services are used. For example the use of a service operation on a provided service may cause multiple service operations to be invoked on one or more of its required service.

Component behaviour will be captured during the design phase using UML notation.

6.1.9.2 At service level

Service behaviour deals with how service operations within services are meant to be used. It will define how to use one or more service operations and in which order. For example a service may provide service operation A, B and C, but these must be used by firstly invoking operation A, followed by either operation B or C, but not both.

Service behaviour will be captured during the design phase using UML notation.

6.1.9.3 At module level

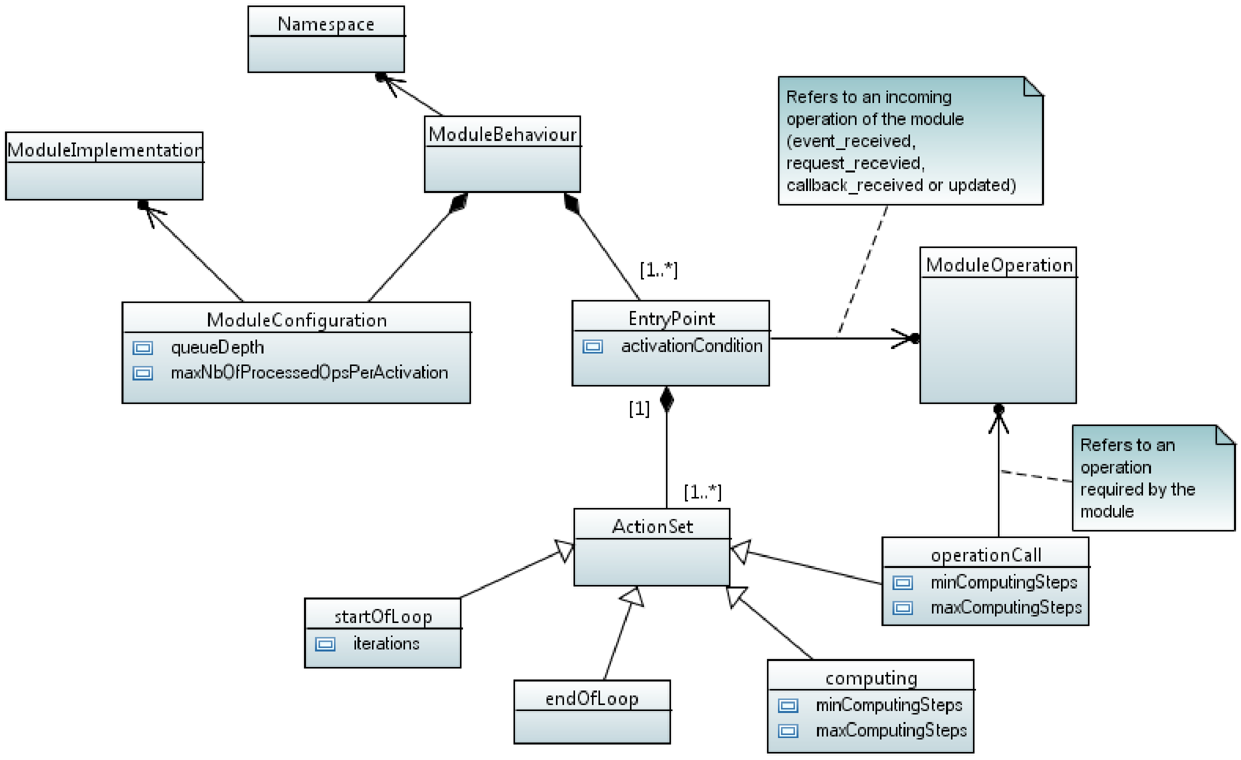

Figure 13 – Module Behaviour

A **moduleBehaviour** allows describing characteristics of a **ModuleImplementation** item, which will be used for deployment and early verification analyses (for example: consistency with the component level behaviour). It mainly gives a decomposition of module treatments, allowing an assessment of CPU resources requirements.

A **moduleBehaviour** is composed of a **moduleConfiguration** and a set of **entryPoints**.

The **moduleConfiguration** refers to the **ModuleImplementation** whose behaviour is described. It defines the maximum number of incoming operations (queueDepth) that can be queued for an instance of this moduleImplementation: note that only “activating” operations (i.e. event_received, request_received or callback of an asynchronous request_sent) are queued. Operations queue management behaviour is FIFO.

NB: a configuration attribute is also created as a provision to specify the maximum number of incoming operations that can be processed within a single module activation (in case an ECOA future model would allow to parameterize the “activating/not activating” property of incoming operations – in the current version, the value is always 1).

An **entryPoint** corresponds to a sequence of **Actions** to be processed on the reception of an activating incoming module operation (according to the rank of the operation in the module operations queue, and according to the scheduling policy of the execution platform). So, each **entryPoint** refers to its associated activating **ModuleOperation**.

The different types of **Actions** that can be specified in this “high level” entry-point behaviour are the following ones:

- startOfLoop (with the number of iterations to be processed),
- endOfLoop,
- computing : it is an internal treatment,
- operationCall : refers to a required module operation (to run a treatment on another module)

For the last two **Actions**, it is possible to define a minimum number of computing steps and a maximum number of computing steps, representing processing complexities for the best case and for the worst case execution scenarios of the related entryPoint. Complexity assessments, through “step” units, have to be consistent with the definition of “step” used to value the **LogicalComputingNode** “stepDuration” parameter (in the **LogicalSystem**).

This way, the “high level” entry-point behaviour allows assessing the Best Case Execution Time (BCET) and the Worst Case Execution Time (WCET) of each module entry-point.

It is important to notice that, in the case of an operation call, values of computing steps correspond to the internal cost of the operation call, and not to the cost of actions to be processed by the called module.

6.2 Concrete Meta-Model

6.2.1 Mapping onto Service-Component Architecture (SCA)

The Service-Component Architecture (SCA) is a standardised model for building applications and (software) systems using a Service-Oriented Architecture (SOA), developed by a set of industry partners. Initially developed as an industrial collaboration, this open standard is now reaching maturity and is maintained by the OASIS (http://www.oasis-open.org/) organisation. Using some SCA concepts and implementations avoids unnecessary re-implementation and potentially leverages existing tool support.

This section describes the translation of the abstract meta-model described in the previous section onto an XML meta-model based on the SCA assembly model. In fact, the XML meta-model re-describes all ECOA artefacts already described by the abstract meta-model but in a way usable by software tooling.

6.2.1.1 Rules on XML writing

Certain rules need to be followed to ensure that the XML is consistent and correct. The following rules are in addition to the normal validation requirements of the XML relative to its schema.

- Information names used within XML files are case sensitive. If the name of one item is used many times, character strings used for that name shall use the same case sensitivity.
- The parsing of XML files is done in one pass; i.e. items need to be defined before they are used. For example, the type for a field in a structure shall be defined before the definition of the structure.
- Each component implementation name must be unique within the protection domain hosting it
- Each component instance name must be unique within the assembly schema
- Each module implementation name must be unique across the assembly
- Each module instance name must be unique within the component implementation
- Each module implementation name must be unique within the protection domain hosting it
- Each operation name must be unique within each module definition
- Operation and module names must follow the naming conventions for identifiers used in the most common programming languages: a name being a sequence of letters, figures and underscores, beginning with a letter.
- The order of Operation Parameters in the Component Definition and Implementaiton must match the order declared in the Service Definition.

6.2.1.2 XPath Syntax

The syntax used to identify an element is the XPath one. XPath uses path expressions to select nodes in an XML document. The node is selected by following a path or steps. The most useful path expressions are listed below:

Table 2 – XPath Expressions

Expression	Description
Nodename	Selects all child nodes of the named node
@	Selects attributes of the current node

More information can be found at http://www.w3schools.com/xpath/xpath_syntax.asp.

Nodename in XPath should be a NCName (a name which does not contain colon character) or a QName (prefix:localName where prefix is defined as a reference to a namespace elsewhere).

To avoid a prefix definition we add a new syntax: {namespace}localName where 'namespace' is equal to 'ecoa-sca' in the table below.

6.2.1.3 ECOA to SCA Mapping

Table 3 – Relations between the ECOA abstract meta-model and the SCA Assembly model

ECOA Abstract item	SCA item
ServiceDefinition and service-level operations	Interface extension See Section 8.3 (ecoa-interface-1.0.xsd)
ComponentDefinition	componentType See Section 8.9(ecoa-interface-1.0.xsd)
Property	Property
Property / @name	property/@name
Property / @type	property/@{ecoa-sca}type property/@type="xsd:string" – See note 1.
ProvidedService	Service
ProvidedService/@name	service/@name
ProvidedService / @ServiceDefinition	service/{ecoa-sca}interface/@syntax
RequiredService	Reference
RequiredService / @name	reference/@name
RequiredService / @ServiceDefinition	reference / {ecoa-sca}interface / @syntax
ComponentImplementation, module artefacts, module-level operations and promotion links	Implementation extension See Section 8.8 (ecoa-implementation-1.0.xsd)
Component	Component
Component / @name	component/@name
Component / @ComponentDefinitionRef	component/{ecoa-sca}instance/ @componentType
Component / @ComponentImplementationRef	component/{ecoa-sca}instance/ {ecoa-sca}implementation/@path
ServiceLink	Wire
ServiceLink / @ProvidedServiceRef	wire / @target
ServiceLink / @RequiredServiceRef	wire / @source
AssemblySchema	Composite See Section 8.4 (ecoa-sca-instance-1.0.xsd)
DeploymentSchema	Refinement of a composite. Computing nodes are described in separate XML files.
Data types	Specific description See Section 8.13 (ecoa-types-1.0.xsd)

Notes:

1. SCA uses xsd types to type component properties but ECOA has defined its own data typing model. As the assembly schema associates ECOA concepts to SCA concepts, the way to associate the ECOA type of the property to the SCA type is to use systematically the generic "xsd:string".

6.2.2 Schemas

The ECOA concrete meta-model references the following files produced by the OASIS organisation. Currently the ECOA metamodel is defined against version 1.1 of the SCA.

SCA (sca- 1.1-cd06.xsd) which is available from:

http://docs.oasis-open.org/opencsa/sca-assembly/sca-1.1-cd06.xsd

SCA core (sca-core-1.1-cd06.xsd) which is available from:

http://docs.oasis-open.org/opencsa/sca-assembly/sca-core-1.1-cd06.xsd

SCA contributions (sca-contribution-1.1-cd06.xsd) which is available from:

http://docs.oasis-open.org/opencsa/sca-assembly/sca-contribution-1.1-cd06.xsd

The ECOA metamodel only refers to a subset of the SCA concepts (see 6.2.1.3). the section describes in extension the selected subset. It is so possible to comment out unused XSD entries in SCA schemas to validate the ECOA XML files. Filtered versions of SCA schemas are available; their names are suffixed with '-subset'.

The following table describes the ECOA schemas, which are presented in full in Section 8.

Table 4 – ECOA Defined Schemas

Filename	Description	Section
ecoa-sca-1.0.xsd	Required for compatibility with SCA. Defines SCA extension schemas.	8.1
ecoa-sca-attributes-1.0.xsd	Required for compatibility with SCA. Defines SCA attribute extensions.	8.2
ecoa-sca-interface-1.0.xsd	Describes reference to service definition at component level within the assembly	8.3
ecoa-sca-instance-1.0.xsd	Describes reference to component implementation description at component level within the assembly	8.4
ecoa-bin-desc-1.0.xsd	Defines the links between module implementations and binary objects.	8.5
ecoa-common-1.0.xsd	Declares the use of a library of data types.	8.6
ecoa-deployment-1.0.xsd	Defines how Modules are mapped onto a logical architecture (ie. protection domains and processing nodes)	8.7

Filename	Description	Section
ecoa-implementation-1.0.xsd	Describes all the information needed to integrate the software implementation of an ECOA component in an ECOA system.	8.8
ecoa-interface-1.0.xsd	Describes an ECOA service, including a set of operations.	8.9
ecoa-interface-qos.xsd	Describes the provided and required quality of service associated with a component definition.	8.10
ecoa-logicalsystem-1.0.xsd	Describes a logical computing architecture consisting of computing nodes and protection domains connected by a network. This architecture description is intended to support early verification.	8.11
ecoa-module-behaviour-1.0.xsd	Describes the behaviour of module operations.	8.12
ecoa-types-1.0.xsd	Describes the syntax for defining ECOA types constructed from the basic ECOA predefined types.	8.13
ecoa-project-1.0.xsd	Describes directories used for one given ECOA application	8.14
ecoa-udpbinding-1.0.xsd	Describes the binding for UDP communication using ELI.	8.15
ecoa-uid-1.0.xsd	Describes directories used for one given ECOA application	8.16
sca-1.1-cd06-subset.xsd	Selected subset of Service Component Architecture Schema Version 1.1	8.17
sca-contribution-1.1-cd06-subset.xsd	Selected subset of Service Component Architecture Contribution Schema Version 1.1	8.18
sca-core-1.1-cd06-subset.xsd	Selected subset of Service Component Architecture Core Schema Version 1.1	8.19

ECOA Schemas and XML files are fully compliant with the W3C XML Standards. They are validated with the following files:

- XMLSchema.xsd that escribes the Schema for XML Schemas. Origin of the file used for the purpose of ECOA: http://www.w3c.org/2001/XMLSchema.xsd
- xml.xsd that describes the XML namespace, in a form suitable for import by other schema documents. Origin of the file used for the purpose of ECOA: http://www.w3c.org/2001/xml.xsd

6.2.3 Filename Conventions

Table 5 specifies standard filenames for the different instances of the ECOA concrete meta-model as generated by an ECOA toolset. It also defines the main XSD file associated to the kind of file and it can be used as an entry point within the concrete meta-model.

Table 5 – ECOA Standard Filenames

	ECOA Standard Filename	Comments	XSD
Project definition	`#filename#.project.xml`	General information about one ECOA application - Optional in current stages	ecoa-project-1.0.xsd
Type definitions	`#filename#.types.xml`	Data types used by operations within service or module definitions	ecoa-types-1.0.xsd
Service definition	`#filename#.interface.xml`	List of service operations Name required for conformance to SCA	ecoa-interface-1.0.xsd
Component definition	`#filename#.componentType`	List of services provided and required by the component and its properties. Name required for conformance to SCA	sca-1.1-cd06-subset.xsd
Service QoS definition	`#filename#.interface.qos.xml`	Service operation-level QoS	ecoa-interface-qos-1.0.xsd
Component implementation	`#filename#.impl.xml`	Description of component architecture: modules, triggers, module operation links, etc	ecoa-implementation-1.0.xsd
Module behaviour	`#filename#.behaviour.xml`	File defining a macro temporal behaviour of the module	ecoa-module-behaviour-1.0.xsd
Initial assembly schema	`#filename#.composite`	Application architecture connecting component instances through wires. Decorrelated from any implementation Name required for conformance to SCA	sca-1.1-cd06-subset.xsd
Final assembly schema	`#filename#.impl.composite`	This file adds to `#filename#.composite` pointers to component implementations. Name required for conformance to SCA	sca-1.1-cd06-subset.xsd
Deployment schema	`#filename#.deployment.xml`	Mapping of modules onto computing nodes	ecoa-deployment-1.0.xsd

	ECOA Standard Filename	Comments	XSD
Mapping onto binary files	`#filename#.bin-desc.xml` `bin-desc.xml`	Mapping of logical module implementation names onto actual physical binary files Useful for packaging	ecoa-bin-desc-1.0.xsd
Logical System	`#filename#.logical-system.xml`	Description of the computing platforms: computing nodes, links between them and performance characteristics.	ecoa-logicalsystem-1.0.xsd

6.2.4 Interim data organisation

Within the development toolset all data describing the example are organized into files and directories.

Figure 14 shows an intermediate organisation used during early stages of the programme. This organisation might then evolved based on the optional ecoa-project file in future stages.

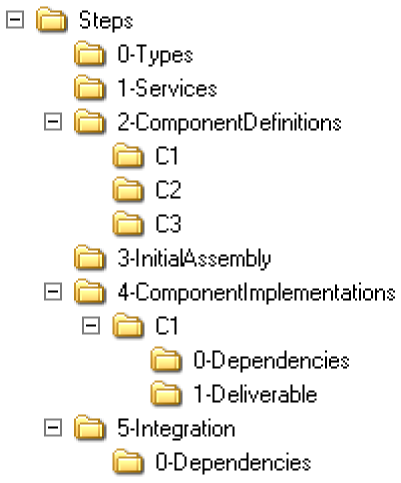

Figure 14 – Directories

Data types used for every definition are defined by "***.types.xml" located in the directory named "0-Types".

Service definitions are defined by "***.interface.xml" located in the directory named "1-Services".

Each Component Definition is described by a "***.componentType" file located in a sub-directory of the directory named "2-ComponentDefinitions". The name of the sub-directory is the name of the component definition itself. For each Component Definition, "***.interface-qos.xml" files describe the initial QoS expected for each service provided or required by an instance of this component definition.

The initial Assembly Schema is defined by a "***.composite" file located in the directory named "3-InitialAssembly".

Each Component Implementation is described by a "***.impl.xml" file located in a sub-directory of the directory named "4-ComponentImplementations". The name of the sub-directory is the name of the component implementation itself. The component supplier may also overload the expected with a new QoS; however, the new QoS shall be compatible with the expected one (e.g. an overloaded data maxageing can be lesser than the expected one). For each Component Implementation, "***.behaviour.xml" files describe the behaviour of each module operation. The file bin-desc.xml describes the list of binary objects associated to modules. computingPlaform and computingNode attributes of the element executeOn in the deployment XML file shall match id attributes of logicialComputingPlatform element and one of its logicalComputingNode child elements in the logical-system XML file. The values for these attributes are free character strings. It is not required to use fixed prefixes.

The directory "5-Integration" describes associations and mappings of software onto a logical system. The logical system is described by the file "logical-system.xml": it defines logical computing nodes and logical links between them. The association between the component instances and the component implementations is described by the "***.impl.composite". The grouping of modules into partition domains and the mapping of partition domains onto logical computing nodes is described by the file "deployment.xml". The actual deployment (fine grain deployment) is described by platform-specific files and shall be documented by the platform provider. These files are not described in this example. The file "sca-contribution.xml" is only there for compatibility with the SCA standard.

The following table summarizes text above.

Table 6 – Model Data Organisation

Directory	Sub-directory 1	Sub-directory 2	Files
0-Types	N/A	N/A	***.types.xml
1-Services	N/A	N/A	***.interface.xml
2-ComponentDefinitions	<name_of_component _definition>		<name_of_component>.componentType ***.interface.qos.xml By example: required_<service_name>.qos.xml and provided_<service_name>.qos.xml
3-InitialAssembly	N/A		***.composite
4-ComponentImplementations	<name_of_implement ation>		<name_of_implementation>.impl.xml ***.interface.qos.xml (e.g. new_required_<service_name>.qos.xml) bin-desc.xml ***.behaviour.xml Binary files (e.g. *.o or *.dll)
		0-Dependencies	Data type, service and component definitions if "0-Types", "1-Services" and "2-ComponentDefinitions" directories are not available.
		1-Deliverable	Zipped file of the upper directory
5-Integration	N/A		***.impl.composite logical-system.xml deployment.xml sca-contribution.xml
	0-Dependencies	N/A	Set of directories containing component implementations if 4-ComponentImplementations is not available

Tools developed for the ECOA Phase 1 demonstrations may rely on this data organisation.

7 Legality Rules

This section defines textual to insure coherency and consistency of ECOA XML files as well as their compatibility with the underlying SCA technology.

The section is splitted in several subsections following the interim data organization.

7.1 Types

Empty section

7.2 Services

Empty section

7.3 Component Definitions

Empty section

7.4 Initial Assembly schema

[XML-AS-1] Each component property defined in an assembly schema shall use the type xsd:string for its attribute “type”.

The actual ECOA type of the property is defined by the mandatory attribute “@{ecoa-sca}type”.

7.5 Component Implementations

Empty section

7.6 Integration

Empty section

8 ECOA Schemas

8.1 ecoa-sca-1.0.xsd

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
  xmlns:ecoa-sca="http://www.ecoa.technology/sca"
  xmlns:sca="http://docs.oasis-open.org/ns/opencsa/sca/200912"
  elementFormDefault="qualified"
  targetNamespace="http://www.ecoa.technology/sca">
  <import namespace="http://docs.oasis-open.org/ns/opencsa/sca/200912"
    schemaLocation="sca-core-1.1-cd06-subset.xsd"/>
  <include schemaLocation="extensions/ecoa-sca-instance-1.0.xsd"/>
  <include schemaLocation="extensions/ecoa-sca-interface-1.0.xsd"/>
  <include schemaLocation="ecoa-sca-attributes-1.0.xsd"/>

</schema>
```

8.2 ecoa-sca-attributes-1.0.xsd

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
xmlns:xs="http://www.w3.org/2001/XMLSchema"
  elementFormDefault="qualified" targetNamespace="http://www.ecoa.technology/sca">
  <attribute name="rank" type="xs:int"/>
  <attribute default="false" name="allEventsMulticasted" type="xs:boolean">
    <annotation>
      <documentation>Boolean indicating if all events provided by the sender are
        multicast or not
      </documentation>
    </annotation>
  </attribute>
  <attribute name="type" type="xs:string"/>
  <attribute name="deployment" type="xs:string"/>
  <complexType name="cia">
    <xs:annotation>
      <xs:documentation>PROVISIONAL Defines level of CIA required for a wire
      </xs:documentation>
    </xs:annotation>
    <xs:attribute name="confidentiality" type="xs:string" use="required"/>
    <xs:attribute name="integrity" type="xs:string" use="required"/>
    <xs:attribute name="availability" type="xs:string" use="required"/>
  </complexType>

</schema>
```

8.3 ecoa-sca-interface-1.0.xsd

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
xmlns:ecoa="http://www.ecoa.technology/sca"
```

```
  xmlns:jxb="http://java.sun.com/xml/ns/jaxb" xmlns:sca="http://docs.oasis-
open.org/ns/opencsa/sca/200912"
  elementFormDefault="qualified" jxb:version="1.0"
targetNamespace="http://www.ecoa.technology/sca">

  <import namespace="http://docs.oasis-open.org/ns/opencsa/sca/200912"
    schemaLocation="../sca-core-1.1-cd06-subset.xsd"/>

  <element name="interface" substitutionGroup="sca:interface" type="ecoa:Interface">
    <annotation>
      <appinfo>
        <jxb:class name="ecoaInterfaceElement"/>
      </appinfo>
    </annotation>
  </element>

  <complexType name="Interface">
    <annotation>
      <appinfo>
        <jxb:class name="EcoaInterface"/>
      </appinfo>
    </annotation>
    <complexContent>
      <extension base="sca:Interface">
        <attribute name="syntax" type="anyURI" use="required"/>
        <attribute name="qos" type="anyURI" use="optional"/>
      </extension>
    </complexContent>
  </complexType>

</schema>
```

8.4 ecoa-sca-instance-1.0.xsd

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
xmlns:ecoa="http://www.ecoa.technology/sca"
  xmlns:sca="http://docs.oasis-open.org/ns/opencsa/sca/200912"
  elementFormDefault="qualified" targetNamespace="http://www.ecoa.technology/sca">


  <import namespace="http://docs.oasis-open.org/ns/opencsa/sca/200912"
    schemaLocation="../sca-core-1.1-cd06-subset.xsd"/>

  <element name="instance" substitutionGroup="sca:implementation" type="ecoa:Instance"/>

  <complexType name="Instance">
    <complexContent>
      <extension base="sca:Implementation">
        <sequence>
          <element maxOccurs="1" minOccurs="0" name="implementation">
            <complexType>
              <attribute name="name" type="string" use="required"/>
            </complexType>
          </element>
        </sequence>

        <attribute name="componentType" type="anyURI" use="required"/>
        <attribute name="version" type="string" use="optional"/>
      </extension>
    </complexContent>
  </complexType>

</schema>
```

8.5 ecoa-bin-desc-1.0.xsd

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<xsd:schema xmlns="http://www.ecoa.technology/bin-desc-1.0"
xmlns:tns="http://www.ecoa.technology/bin-desc-1.0"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified"
  targetNamespace="http://www.ecoa.technology/bin-desc-1.0">

  <xsd:include schemaLocation="ecoa-common-1.0.xsd"/>
  <xsd:element name="binDesc" type="BinDesc"/>

  <xsd:complexType name="BinDesc">
    <xsd:annotation>
      <xsd:documentation>Links between module implementations and binary objects
      </xsd:documentation>
    </xsd:annotation>
    <xsd:sequence>
      <xsd:element name="processorTarget" type="ProcessorTarget"/>
      <xsd:element maxOccurs="unbounded" name="binaryModule" type="BinaryModule"/>
    </xsd:sequence>
    <!-- the following attribute points to a logical name -->
    <xsd:attribute name="componentImplementation" type="NameId" use="required"/>
  </xsd:complexType>

  <xsd:complexType name="ProcessorTarget">
    <xsd:annotation>
      <xsd:documentation>"Identification of the processor for which modules have
        been compiled"
      </xsd:documentation>
    </xsd:annotation>
    <xsd:attribute name="type" type="xsd:string" use="required"/>
  </xsd:complexType>

  <xsd:complexType name="BinaryModule">
    <xsd:annotation>
      <xsd:documentation>A set of named operations.</xsd:documentation>
    </xsd:annotation>
    <xsd:attribute name="reference" type="xsd:NCName" use="required">
      <xsd:annotation>
        <xsd:documentation>Name of the module implementation</xsd:documentation>
      </xsd:annotation>
    </xsd:attribute>
    <xsd:attribute name="object" type="xsd:anyURI" use="required">
      <xsd:annotation>
        <xsd:documentation>Filename of the binary implementing the referenced
```

```
          module
        </xsd:documentation>
      </xsd:annotation>
    </xsd:attribute>
    <xsd:attribute name="userContextSize" type="xsd:int" use="required">
      <xsd:annotation>
        <xsd:documentation>Size in bytes of the module user context
        </xsd:documentation>
      </xsd:annotation>
    </xsd:attribute>
    <xsd:attribute name="stackSize" type="xsd:int" use="required">
      <xsd:annotation>
        <xsd:documentation>maximum size in bytes of the stack used by any module
          entry point (including all sub-function calls)
        </xsd:documentation>
      </xsd:annotation>
    </xsd:attribute>
    <xsd:attribute name="heapSize" type="xsd:int" use="required">
      <xsd:annotation>
        <xsd:documentation>maximum size in bytes of the heap (memory dynamically
          allocated by the module binary itself: malloc or object instances)
        </xsd:documentation>
      </xsd:annotation>
    </xsd:attribute>
    <xsd:attribute name="checksum" type="xsd:string" use="required">
      <xsd:annotation>
        <xsd:documentation>MD5sum of the binary</xsd:documentation>
      </xsd:annotation>
    </xsd:attribute>

  </xsd:complexType>

</xsd:schema>
```

8.6 ecoa-common-1.0.xsd

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">

  <!-- The following regexps define what is allowed/forbidden for each kind of names
    used in ECOA. They must take into account the contraints of different programming
    languages and development environments supported by ECOA (characters allowed in file
    names, identifiers, etc.) NOTE: XML character classes (\i, \c, etc.) are
intentionally
    avoided, because of the complexity of their definition. -->

  <!-- Name of a library containing data types -->
  <!-- Note: The '.' character is used to structure libraries into hierarchical
namespaces
    (like Java packages). -->
  <xsd:simpleType name="LibraryName">
    <xsd:restriction base="xsd:string">
      <xsd:pattern value="[A-Za-z][A-Za-z0-9_\.]*"/>
    </xsd:restriction>
  </xsd:simpleType>
  <!-- Name that can be used as an identifier in ECOA models and in the source code
    of ECOA components -->
  <!-- Note: Names starting with '_' are excluded from ECOA models. -->
  <xsd:simpleType name="NameId">
    <xsd:restriction base="xsd:string">
      <xsd:pattern value="[A-Za-z][A-Za-z0-9_\-]*"/>
    </xsd:restriction>
  </xsd:simpleType>

  <!-- Name of a data type inside a library -->
  <xsd:simpleType name="TypeName">
    <xsd:restriction base="NameId">
    </xsd:restriction>
  </xsd:simpleType>

  <!-- Name of a type, possibly prefixed by the name of the library that defines
    it. -->
  <!-- The prefix may be omitted only for predefined types. -->
  <!-- A type T defined in a library L will be denoted "L:T". -->
  <xsd:simpleType name="TypeQName">
    <xsd:restriction base="xsd:string">
      <xsd:pattern value="([A-Za-z][A-Za-z0-9_\.]*:)?[A-Za-z][A-Za-z0-9_]*"/>
    </xsd:restriction>
  </xsd:simpleType>

  <xsd:element name="use">

    <xsd:annotation>
      <xsd:documentation>Declares the use of a library of data types. A type T
        defined in a library L will be denoted "L:T".
      </xsd:documentation>
    </xsd:annotation>

    <xsd:complexType>
      <xsd:attribute name="library" type="LibraryName" use="required"/>
    </xsd:complexType>
  </xsd:element>

</xsd:schema>
```

8.7 ecoa-deployment-1.0.xsd

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<xsd:schema xmlns="http://www.ecoa.technology/deployment-1.0"
xmlns:tns="http://www.ecoa.technology/deployment-1.0"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified"
  targetNamespace="http://www.ecoa.technology/deployment-1.0">
  <xsd:include schemaLocation="ecoa-common-1.0.xsd"/>

  <xsd:element name="deployment" type="Deployment">
    <xsd:key name="execnamekey">
      <xsd:selector xpath="tns:protectionDomain"/>
      <xsd:field xpath="@name"/>
    </xsd:key>
  </xsd:element>
  <xsd:complexType name="Deployment">
    <xsd:sequence>
      <xsd:choice maxOccurs="unbounded" minOccurs="1">
        <xsd:element maxOccurs="unbounded" minOccurs="0" name="LogPolicy"
          type="LogPolicy"/>
        <xsd:element maxOccurs="unbounded" name="protectionDomain"
type="ProtectionDomain"/>
      </xsd:choice>
    </xsd:sequence>
    <xsd:attribute name="finalAssembly" type="NameId" use="required">
      <xsd:annotation>
        <xsd:documentation>Name of the composite referenced by this deployment
        </xsd:documentation>
      </xsd:annotation>
    </xsd:attribute>
    <xsd:attribute name="logicalSystem" type="NameId" use="required">
      <xsd:annotation>
        <xsd:documentation>Name of the logical system this deployment is made on
        </xsd:documentation>
      </xsd:annotation>
    </xsd:attribute>
  </xsd:complexType>
  <xsd:complexType name="ProtectionDomain">
    <xsd:annotation>
      <xsd:documentation>Defines an OS executable, offering memory (and possibly
        also temporal) protection
      </xsd:documentation>
    </xsd:annotation>
    <xsd:sequence>
      <xsd:element name="executeOn">
        <xsd:complexType>
          <xsd:attribute name="computingNode" type="NameId" use="required"/>
          <xsd:attribute name="computingPlatform" type="NameId" use="optional">
            <xsd:annotation>
              <xsd:documentation>Id of a logical system.</xsd:documentation>
            </xsd:annotation>
          </xsd:attribute>
        </xsd:complexType>
      </xsd:element>
      <xsd:choice maxOccurs="unbounded" minOccurs="0">
        <xsd:element maxOccurs="unbounded" minOccurs="0"
          name="deployedModuleInstance">
          <xsd:complexType>
            <xsd:attribute name="componentName" type="NameId" use="required"/>
            <xsd:attribute name="moduleInstanceName" type="NameId" use="required"/>
            <xsd:attribute name="modulePriority" type="ModulePriority"
              use="required">
              <xsd:annotation>
                <xsd:documentation>abstract module priority that can be used by
                  the platform to map the module on an actual OS priority
                </xsd:documentation>
              </xsd:annotation>
            </xsd:attribute>
          </xsd:complexType>
        </xsd:element>
        <xsd:element maxOccurs="unbounded" minOccurs="0"
          name="deployedTriggerInstance">
          <xsd:complexType>
            <xsd:attribute name="componentName" type="NameId" use="required"/>
            <xsd:attribute name="triggerInstanceName" type="NameId"
              use="required"/>
            <xsd:attribute name="triggerPriority" type="ModulePriority"
              use="required">
              <xsd:annotation>
                <xsd:documentation>abstract trigger priority that can be used by
                  the platform to map the trigger on an actual OS priority
                </xsd:documentation>
              </xsd:annotation>
            </xsd:attribute>
          </xsd:complexType>
        </xsd:element>
      </xsd:choice>
    </xsd:sequence>
    <xsd:attribute name="name" type="NameId" use="required"/>
  </xsd:complexType>

  <xsd:simpleType name="ModulePriority">
    <xsd:restriction base="xsd:decimal">
      <xsd:minInclusive value="0"/>
      <xsd:maxInclusive value="255"/>
    </xsd:restriction>
  </xsd:simpleType>

  <xsd:complexType name="LogPolicy">
    <xsd:annotation>
      <xsd:documentation>Defines the log policy for deployed components and
        modules
      </xsd:documentation>
    </xsd:annotation>
    <xsd:sequence>
      <xsd:element maxOccurs="unbounded" minOccurs="0" name="componentLog"
        type="ComponentLog"/>
    </xsd:sequence>
  </xsd:complexType>

  <xsd:complexType name="ComponentLog">
    <xsd:annotation>
      <xsd:documentation>Defines default level of logging for a given component
      </xsd:documentation>
    </xsd:annotation>
    <xsd:sequence>
      <xsd:element maxOccurs="unbounded" minOccurs="0" name="moduleLog"
        type="ModuleLog"/>
    </xsd:sequence>
    <xsd:attribute name="instanceName" type="xsd:string" use="required"/>
    <xsd:attribute name="enabledLevels" type="xsd:string" use="required"/>
  </xsd:complexType>

  <xsd:complexType name="ModuleLog">
    <xsd:annotation>
      <xsd:documentation>Defines level of logging for a deployed module instance
```

```
      </xsd:documentation>
    </xsd:annotation>
    <xsd:attribute name="instanceName" type="xsd:string" use="required"/>
    <xsd:attribute name="enabledLevels" type="xsd:string" use="required"/>
  </xsd:complexType>

</xsd:schema>
```

8.8 ecoa-implementation-1.0.xsd

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<xsd:schema xmlns="http://www.ecoa.technology/implementation-1.0"
  xmlns:tns="http://www.ecoa.technology/implementation-1.0"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  elementFormDefault="qualified"
targetNamespace="http://www.ecoa.technology/implementation-1.0">
  <xsd:include schemaLocation="ecoa-common-1.0.xsd"/>
  <xsd:element name="componentImplementation" type="ComponentImplementation">
    <!-- keys: name unicity constraints -->
    <xsd:key name="moduleTypekey">
      <xsd:selector xpath="tns:moduleType"/>
      <xsd:field xpath="@name"/>
    </xsd:key>
    <xsd:key name="moduleImplementationkey">
      <xsd:selector xpath="tns:moduleImplementation"/>
      <xsd:field xpath="@name"/>
    </xsd:key>
    <xsd:key name="moduleInstancekey">
      <xsd:selector xpath="tns:moduleInstance"/>
      <xsd:field xpath="@name"/>
    </xsd:key>
    <xsd:key name="triggerInstancekey">
      <xsd:selector xpath="tns:triggerInstance"/>
      <xsd:field xpath="@name"/>
    </xsd:key>
    <xsd:key name="dynamicTriggerInstancekey">
      <xsd:selector xpath="tns:dynamicTriggerInstance"/>
      <xsd:field xpath="@name"/>
    </xsd:key>
    <!-- triggers, dynamicTriggers and ordinary modules must have distinct names -->
    <xsd:key name="moduleOrTriggerInstancekey">
      <xsd:selector
        xpath="tns:moduleInstance|tns:triggerInstance|tns:dynamicTriggerInstance"/>
      <xsd:field xpath="@name"/>
    </xsd:key>
    <!-- The same operation shall appear only one time if present in the element
      clients -->
    <xsd:key name="moduleInstanceClientRequestLinkkey">
      <xsd:selector xpath="tns:requestLink/tns:clients/tns:moduleInstance"/>
      <xsd:field xpath="@instanceName"/>
      <xsd:field xpath="@operationName"/>
    </xsd:key>
    <xsd:key name="serviceClientRequestLinkkey">
      <xsd:selector xpath="tns:requestLink/tns:clients/tns:service"/>
      <xsd:field xpath="@instanceName"/>
```

```
        <xsd:field xpath="@operationName"/>
      </xsd:key>
      <!-- keyrefs: constraints that a reference refers to a name defined in a key -->
      <xsd:keyref name="moduleInstancekeyRef" refer="moduleInstancekey">
        <xsd:selector xpath="*/*/tns:moduleInstance"/>
        <xsd:field xpath="@instanceName"/>
      </xsd:keyref>
      <xsd:keyref name="triggerInstancekeyRef" refer="triggerInstancekey">
        <xsd:selector xpath="tns:eventLink/tns:trigger"/>
        <xsd:field xpath="@triggerInstance"/>
      </xsd:keyref>
      <xsd:keyref name="dynamicTriggerInstancekeyRef" refer="dynamicTriggerInstancekey">
        <xsd:selector xpath="tns:eventLink/*/tns:dynamicTrigger"/>
        <xsd:field xpath="@instanceName"/>
      </xsd:keyref>
      <xsd:keyref name="moduleImplementation_to_moduleType" refer="moduleTypekey">
        <xsd:selector xpath="tns:moduleImplementation"/>
        <xsd:field xpath="@moduleType"/>
      </xsd:keyref>
      <xsd:keyref name="moduleInstance_to_moduleImplementation"
refer="moduleImplementationkey">
        <xsd:selector xpath="tns:moduleInstance"/>
        <xsd:field xpath="@implementationName"/>
      </xsd:keyref>
    </xsd:element>
    <xsd:complexType name="ComponentImplementation">
      <xsd:annotation>
        <xsd:documentation>
          Describes all the information needed to integrate
          the software implementation of an ECOA component in an ECOA
          system.
        </xsd:documentation>
      </xsd:annotation>
      <xsd:sequence>
        <xsd:element maxOccurs="unbounded" minOccurs="0" ref="use"/>
        <xsd:element maxOccurs="unbounded" minOccurs="0" name="service"
          type="ServiceQoS"/>
        <xsd:element maxOccurs="unbounded" minOccurs="0" name="reference"
          type="ServiceQoS"/>
        <xsd:element maxOccurs="unbounded" minOccurs="0" name="moduleType"
          type="ModuleType"/>
        <xsd:element maxOccurs="unbounded" minOccurs="0" name="moduleImplementation"
          type="ModuleImplementation"/>
        <xsd:element maxOccurs="unbounded" minOccurs="0" name="moduleInstance"
          type="ModuleInstance"/>
        <xsd:element maxOccurs="unbounded" minOccurs="0" name="triggerInstance"
          type="TriggerInstance"/>
        <xsd:element maxOccurs="unbounded" minOccurs="0"
          name="dynamicTriggerInstance" type="DynamicTriggerInstance"/>
        <xsd:choice maxOccurs="unbounded">
          <xsd:element maxOccurs="unbounded" minOccurs="0" name="dataLink"
            type="DataLink"/>
          <xsd:element maxOccurs="unbounded" minOccurs="0" name="eventLink"
            type="EventLink"/>
          <xsd:element maxOccurs="unbounded" minOccurs="0" name="requestLink"
            type="RequestLink"/>
        </xsd:choice>
      </xsd:sequence>
      <xsd:attribute name="componentDefinition" type="NameId" use="required"/>
    </xsd:complexType>
    <xsd:complexType name="ServiceQoS">
      <xsd:annotation>
        <xsd:documentation>To define a new QoS for a provided or required
          service
        </xsd:documentation>
      </xsd:annotation>
      <xsd:attribute name="name" type="NameId" use="required"/>
      <xsd:attribute name="newQoS" type="xsd:anyURI" use="required"/>
    </xsd:complexType>
    <xsd:complexType name="ModuleType">
      <xsd:annotation>
        <xsd:documentation>Describes a single-threaded ECOA module,
          implemented as software, contributing to the implementation of
          an ECOA component.
        </xsd:documentation>
      </xsd:annotation>
      <xsd:sequence>
        <xsd:element minOccurs="0" name="properties">
          <xsd:annotation>
            <xsd:documentation>Set of module properties. The value of each module
              property is set at design time.
            </xsd:documentation>
          </xsd:annotation>
          <xsd:complexType>
            <xsd:sequence>
              <xsd:element maxOccurs="unbounded" name="property" type="Parameter">
                <xsd:annotation>
                  <xsd:documentation>The value of each module property is set at
                    design time at instance definition level.
                  </xsd:documentation>
                </xsd:annotation>
              </xsd:element>
            </xsd:sequence>
          </xsd:complexType>
          <xsd:unique name="propertyname">
            <xsd:selector xpath="tns:property"/>
            <xsd:field xpath="@name"/>
          </xsd:unique>
        </xsd:element>
        <xsd:element name="operations">
          <xsd:complexType>
            <xsd:choice maxOccurs="unbounded">
              <xsd:element minOccurs="0" name="dataWritten" type="VersionedData">
                <xsd:annotation>
                  <xsd:documentation>Read+Write access to a versioned data.
                  </xsd:documentation>
                </xsd:annotation>
              </xsd:element>
              <xsd:element minOccurs="0" name="dataRead">
                <xsd:annotation>
                  <xsd:documentation>Read-only access to a versioned data.
                  </xsd:documentation>
                </xsd:annotation>
                <xsd:complexType>
                  <xsd:complexContent>
                    <xsd:extension base="VersionedData">
                      <!-- This deadline is associated to the notifying case -->
                      <xsd:attribute name="notificationDeadline" type="xsd:double"
                        use="optional"/>
                      <xsd:attribute default="false" name="notifying"
                        type="xsd:boolean" use="optional"/>
                    </xsd:extension>
                  </xsd:complexContent>
                </xsd:complexType>
              </xsd:element>
              <xsd:element minOccurs="0" name="eventSent" type="Event">
                <xsd:unique name="eventparameter_sent">
                  <xsd:selector xpath="tns:input"/>
                  <xsd:field xpath="@name"/>
                </xsd:unique>
              </xsd:element>
              <xsd:element minOccurs="0" name="eventReceived">
                <xsd:complexType>
                  <xsd:complexContent>
                    <xsd:extension base="Event">
                      <xsd:attribute name="deadline" type="xsd:double"
                        use="optional"/>
                    </xsd:extension>
                  </xsd:complexContent>
                </xsd:complexType>
                <xsd:unique name="eventparameter_received">
                  <xsd:selector xpath="tns:input"/>
                  <xsd:field xpath="@name"/>
                </xsd:unique>
              </xsd:element>
              <xsd:element minOccurs="0" name="requestSent">
                <xsd:complexType>
                  <xsd:complexContent>
                    <xsd:extension base="RequestResponse">
                      <xsd:attribute name="timeout" type="xsd:double"
                        use="required">
                        <xsd:annotation>
                          <xsd:documentation>Timeout value to unblock/inform
                            respectively a synchronous/asynchronous RR
                            If the value is negative, the timeout is infinite.
                          </xsd:documentation>
                        </xsd:annotation>
                      </xsd:attribute>
                      <xsd:attribute name="callbackDeadline" type="xsd:double"
                        use="optional"/>
                      <xsd:attribute name="isSynchronous" type="xsd:boolean"
                        use="required"/>
                    </xsd:extension>
                  </xsd:complexContent>
                </xsd:complexType>
                <xsd:unique name="requestparameter_req">
                  <xsd:selector xpath="tns:input|tns:output"/>
                  <xsd:field xpath="@name"/>
```

```
            </xsd:unique>
          </xsd:element>
          <xsd:element minOccurs="0" name="requestReceived">
            <xsd:complexType>
              <xsd:complexContent>
                <xsd:extension base="RequestResponse">
                  <xsd:attribute name="deadline" type="xsd:double"
                    use="optional"/>
                  <xsd:attribute default="10" name="maxConcurrentRequests"
                    type="xsd:positiveInteger" use="optional">
                    <xsd:annotation>
                      <xsd:documentation>Max number of concurrent requests that
                        the module may handle for the related entry-point,
                        regardless of incoming requestLinks related to that
                        entry-point. This number should be greater or equal to
                        the sum of requestBufferSizes defined on incoming
                        requestLinks.
                      </xsd:documentation>
                    </xsd:annotation>
                  </xsd:attribute>
                </xsd:extension>
              </xsd:complexContent>
            </xsd:complexType>
            <xsd:unique name="requestparameter_pro">
              <xsd:selector xpath="tns:input|tns:output"/>
              <xsd:field xpath="@name"/>
            </xsd:unique>
          </xsd:element>
        </xsd:choice>
      </xsd:complexType>
      <xsd:key name="operationkey">
        <xsd:selector xpath="tns:*"/>
        <xsd:field xpath="@name"/>
      </xsd:key>
    </xsd:element>
  </xsd:sequence>
  <xsd:attribute name="name" type="NameId" use="required"/>
  <xsd:attribute default="false" name="isSupervisionModule" type="xsd:boolean"
    use="optional"/>
  <xsd:attribute default="false" name="isFaultHandler" type="xsd:boolean"
    use="optional">
    <xsd:annotation>
      <xsd:documentation>To indicate if the module is a Fault
        Handler or not and to generate fault handling-related API
      </xsd:documentation>
    </xsd:annotation>
  </xsd:attribute>
</xsd:complexType>
<xsd:complexType name="Event">
  <xsd:sequence>
    <xsd:element maxOccurs="unbounded" minOccurs="0" name="input" type="Parameter"/>
  </xsd:sequence>
  <xsd:attribute name="name" type="NameId" use="required"/>
</xsd:complexType>
<xsd:complexType name="RequestResponse">
  <xsd:sequence>
```

```
      <xsd:element maxOccurs="unbounded" minOccurs="0" name="input" type="Parameter"/>
      <xsd:element maxOccurs="unbounded" minOccurs="0" name="output"
        type="Parameter"/>
    </xsd:sequence>
    <xsd:attribute name="name" type="NameId" use="required"/>
  </xsd:complexType>
  <xsd:complexType name="VersionedData">
    <xsd:attribute name="name" type="NameId" use="required"/>
    <xsd:attribute name="type" type="TypeQName" use="required">
      <xsd:annotation>
        <xsd:documentation>Type stored by the versioned data.
        </xsd:documentation>
      </xsd:annotation>
    </xsd:attribute>
    <xsd:attribute default="1" name="maxVersions" type="xsd:positiveInteger"
      use="optional">
      <xsd:annotation>
        <xsd:documentation>Max number of versions accessed at the same
          time.
        </xsd:documentation>
      </xsd:annotation>
    </xsd:attribute>
  </xsd:complexType>
  <xsd:complexType name="Parameter">
    <xsd:annotation>
      <xsd:documentation>A parameter a an operation (Event,
        RequestResponse or VersionedData)
      </xsd:documentation>
    </xsd:annotation>
    <xsd:attribute name="name" type="NameId" use="required"/>
    <xsd:attribute name="type" type="TypeQName" use="required"/>
  </xsd:complexType>
  <xsd:complexType name="ModuleImplementation">
    <xsd:attribute name="name" type="NameId" use="required"/>
    <xsd:attribute name="language" use="required">
      <!-- Programming language -->
      <xsd:simpleType>
        <xsd:restriction base="xsd:string">
          <xsd:enumeration value="C"/>
          <xsd:enumeration value="C++"/>
          <xsd:enumeration value="Ada"/>
          <xsd:enumeration value="Java"/>
        </xsd:restriction>
      </xsd:simpleType>
    </xsd:attribute>
    <xsd:attribute name="moduleType" type="NameId" use="required"/>
    <xsd:attribute default="reactive" name="activationModel">
      <!-- Activation model for activating operations management -->
      <xsd:simpleType>
        <xsd:restriction base="xsd:string">
          <xsd:enumeration value="reactive"/>
          <xsd:enumeration value="rhythmic"/>
        </xsd:restriction>
      </xsd:simpleType>
    </xsd:attribute>
    <xsd:attribute name="moduleBehaviour" type="xsd:anyURI" use="optional"/>
```

```
</xsd:complexType>
<xsd:complexType name="Instance">
  <xsd:annotation>
    <xsd:documentation/>
  </xsd:annotation>
  <xsd:attribute name="name" type="NameId" use="required"/>
  <xsd:attribute name="moduleDeadline" type="xsd:double" use="required"/>
</xsd:complexType>
<xsd:complexType name="ModuleInstance">
  <xsd:annotation>
    <xsd:documentation>Describes an instance of a Module (having its
      own internal state).
    </xsd:documentation>
  </xsd:annotation>
  <xsd:complexContent>
    <xsd:extension base="Instance">
      <xsd:sequence>
        <xsd:element minOccurs="0" name="highestRate" type="ModuleRate"/>
        <xsd:element maxOccurs="1" minOccurs="0" name="propertyValues"
          type="PropertyValues"/>
      </xsd:sequence>
      <xsd:attribute name="implementationName" type="NameId" use="required"/>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
<xsd:complexType name="ModuleRate">
  <xsd:attribute name="numberOfOccurrences" type="xsd:decimal" use="optional">
    <xsd:annotation>
      <xsd:documentation>Max number of received activating operations
        during a specified duration
      </xsd:documentation>
    </xsd:annotation>
  </xsd:attribute>
  <xsd:attribute name="timeFrame" type="xsd:double" use="optional">
    <xsd:annotation>
      <xsd:documentation>Equal to min inter-arrival time between 2
        activating operations when numberOfOccurrences value is 1.
        In other cases, specifies a sizing duration for activating operations
        bursts. Unit is second.
      </xsd:documentation>
    </xsd:annotation>
  </xsd:attribute>
</xsd:complexType>

<xsd:complexType name="PropertyValues">
  <xsd:annotation>
    <xsd:documentation>set of module property values</xsd:documentation>
  </xsd:annotation>
  <xsd:sequence>
    <xsd:element maxOccurs="unbounded" name="propertyValue" type="PropertyValue">
    </xsd:element>
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="PropertyValue">
  <xsd:simpleContent>
```

```
      <xsd:extension base="xsd:string">
        <xsd:attribute name="name" type="xsd:string" use="required"/>
      </xsd:extension>
    </xsd:simpleContent>
  </xsd:complexType>

  <xsd:complexType name="DataLink">
    <xsd:annotation>
      <xsd:documentation>Link between DATA operations.</xsd:documentation>
    </xsd:annotation>
    <xsd:sequence>
      <xsd:element name="writers">
        <xsd:complexType>
          <xsd:sequence maxOccurs="unbounded">
            <xsd:choice>
              <xsd:element name="reference" type="OpRef"/>
              <xsd:element name="moduleInstance" type="OpRef"/>
            </xsd:choice>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>
      <xsd:element name="readers">
        <xsd:complexType>
          <xsd:sequence maxOccurs="unbounded">
            <xsd:choice>
              <xsd:element name="service" type="OpRef"/>
              <xsd:element name="moduleInstance" type="OpRefActivatingFifo"/>
            </xsd:choice>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>
      <xsd:element minOccurs="0" name="defaultvalue" type="xsd:string">
        <xsd:annotation>
          <xsd:documentation>The value that consumers will read, if they
            read before any production has occured
          </xsd:documentation>
        </xsd:annotation>
      </xsd:element>
    </xsd:sequence>
    <xsd:attribute name="id" type="xsd:int" use="optional"/>
  </xsd:complexType>
  <xsd:complexType name="EventLink">
    <xsd:annotation>
      <xsd:documentation>Link between EVENT operations.
      </xsd:documentation>
    </xsd:annotation>
    <xsd:sequence>
      <xsd:element minOccurs="0" name="senders">
        <xsd:complexType>
          <xsd:sequence maxOccurs="unbounded">
            <xsd:choice>
              <xsd:element name="service" type="OpRef"/>
              <xsd:element name="reference" type="OpRef"/>
              <xsd:element name="moduleInstance" type="OpRef"/>
              <xsd:element name="trigger" type="OpRef_Trigger"/>
              <xsd:element name="dynamicTrigger" type="OpRef"/>
            </xsd:choice>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>
      <xsd:element name="receivers">
        <xsd:complexType>
          <xsd:sequence maxOccurs="unbounded">
            <xsd:choice>
              <xsd:element name="service" type="OpRef"/>
              <xsd:element name="reference" type="OpRef"/>
              <xsd:element name="moduleInstance" type="OpRefActivatingFifo"/>
              <xsd:element name="dynamicTrigger" type="OpRef"/>
            </xsd:choice>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>
    </xsd:sequence>
    <xsd:attribute name="id" type="xsd:int" use="optional"/>
  </xsd:complexType>
  <xsd:complexType name="RequestLink">
    <xsd:annotation>
      <xsd:documentation>Link between RR operations. Must have exactly one
        server. Can have many clients.
      </xsd:documentation>
    </xsd:annotation>
    <xsd:sequence>
      <xsd:element name="clients">
        <xsd:complexType>
          <xsd:sequence maxOccurs="unbounded">
            <xsd:choice>
              <xsd:element name="service" type="OpRef"/>
              <xsd:element name="moduleInstance" type="OpRefActivatingFifo">
                <xsd:annotation>
                  <xsd:documentation>Note: attributes 'activating' and
                    'fifoSize' concern the response, and are applicable to
                    asynchronous RR operations only.
                  </xsd:documentation>
                </xsd:annotation>
              </xsd:element>
            </xsd:choice>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>
      <xsd:element name="server">
        <xsd:complexType>
          <xsd:choice>
            <xsd:element name="reference" type="OpRef"/>
            <xsd:element name="moduleInstance" type="OpRefServer">
              <xsd:annotation>
                <xsd:documentation>Note: optional attributes concern the request
                </xsd:documentation>
              </xsd:annotation>
            </xsd:element>
          </xsd:choice>
        </xsd:complexType>
      </xsd:element>
    </xsd:sequence>
    <xsd:attribute name="id" type="xsd:int" use="optional"/>
  </xsd:complexType>
  <xsd:complexType name="OpRef">
    <xsd:attribute name="instanceName" type="NameId" use="required">
      <xsd:annotation>
        <xsd:documentation>Reference to a module instance, a service, or a
          reference
        </xsd:documentation>
      </xsd:annotation>
    </xsd:attribute>
    <xsd:attribute name="operationName" type="NameId" use="required"/>
  </xsd:complexType>
  <xsd:complexType name="OpRefActivating">
    <xsd:complexContent>
      <xsd:extension base="OpRef">
        <xsd:attribute default="true" name="activating" type="xsd:boolean"
          use="optional">
          <xsd:annotation>
            <xsd:documentation>Does the reception of the event/data/rr cause
              the activation of the receiver module ?
            </xsd:documentation>
          </xsd:annotation>
        </xsd:attribute>
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>
  <xsd:complexType name="OpRefServer">
    <xsd:complexContent>
      <xsd:extension base="OpRefActivatingFifo">
        <xsd:attribute default="1" name="requestBufferSize" type="xsd:positiveInteger"
          use="optional">
          <xsd:annotation>
            <xsd:documentation>Maximum number of pending requests that can be
              simultaneously processed by the server module. This
              number is associated to the incoming requestLink. Pending
              means the requests have been taken out of the
              FIFO but the replies have not been sent to clients.
            </xsd:documentation>
          </xsd:annotation>
        </xsd:attribute>
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>
  <xsd:complexType name="OpRefActivatingFifo">
    <xsd:complexContent>
      <xsd:extension base="OpRefActivating">
        <xsd:attribute default="8" name="fifoSize" type="xsd:int" use="optional">
          <xsd:annotation>
            <xsd:documentation>Max number of incoming operations that can be
              stored in the receiver module's FIFO queue for that particular
              operation link, before the activation of the corresponding
              entrypoint. There is one fifoSize per operation link on the
              receiver side.
              If this max number is exceeded, new incoming operations are
              trashed.
            </xsd:documentation>
          </xsd:annotation>
        </xsd:attribute>
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>
  <xsd:complexType name="OpRef_Trigger">
    <xsd:attribute name="instanceName" type="NameId" use="required"/>
    <xsd:attribute name="period" type="xsd:double" use="required">
      <xsd:annotation>
        <xsd:documentation>period in seconds</xsd:documentation>
      </xsd:annotation>
    </xsd:attribute>
  </xsd:complexType>
  <xsd:complexType name="TriggerInstance">
    <xsd:complexContent>
      <xsd:extension base="Instance">
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>
  <xsd:complexType name="DynamicTriggerInstance">
    <xsd:complexContent>
      <xsd:extension base="Instance">
        <xsd:sequence>
          <xsd:element maxOccurs="unbounded" minOccurs="0" name="parameter"
            type="Parameter"/>
        </xsd:sequence>
        <xsd:attribute default="1" name="size" type="xsd:int" use="optional">
          <xsd:annotation>
            <xsd:documentation>Max number of events waiting for delay expiration
              in the trigger
            </xsd:documentation>
          </xsd:annotation>
        </xsd:attribute>
        <xsd:attribute default="0.0" name="delayMin" type="xsd:double"
          use="optional">
          <xsd:annotation>
            <xsd:documentation>The trigger will not accept delays lower that
              this value (in seconds)
            </xsd:documentation>
          </xsd:annotation>
        </xsd:attribute>
        <xsd:attribute name="delayMax" type="xsd:double" use="optional">
          <xsd:annotation>
            <xsd:documentation>
              The trigger will not accept delays higher that
              this value (in seconds)
            </xsd:documentation>
          </xsd:annotation>
        </xsd:attribute>
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>
</xsd:schema>
```

8.9 ecoa-interface-1.0.xsd

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<xsd:schema xmlns="http://www.ecoa.technology/interface-1.0"
xmlns:tns="http://www.ecoa.technology/interface-1.0"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified"
  targetNamespace="http://www.ecoa.technology/interface-1.0">
  <xsd:include schemaLocation="ecoa-common-1.0.xsd"/>
  <xsd:element name="serviceDefinition" type="ServiceDefinition"/>
  <xsd:complexType name="ServiceDefinition">
    <xsd:annotation>
      <xsd:documentation>The definition of an ECOA service, including a set of
        operations.
      </xsd:documentation>
    </xsd:annotation>
    <xsd:sequence>
      <xsd:element maxOccurs="unbounded" minOccurs="0" ref="use"/>
      <xsd:element name="operations" type="Operations"/>
    </xsd:sequence>
  </xsd:complexType>
  <xsd:complexType name="Operations">
    <xsd:annotation>
      <xsd:documentation>A set of named operations.</xsd:documentation>
    </xsd:annotation>
    <xsd:choice maxOccurs="unbounded" minOccurs="0">
      <xsd:element name="data" type="Data"/>
      <xsd:element name="event" type="Event"/>
      <xsd:element name="requestresponse" type="RequestResponse"/>
    </xsd:choice>
  </xsd:complexType>
  <xsd:complexType abstract="true" name="Operation">
    <xsd:attribute name="name" type="NameId" use="required"/>
    <xsd:attribute name="comment" type="xsd:string" use="optional"/>
  </xsd:complexType>
  <xsd:complexType name="Data">
    <xsd:annotation>
      <xsd:documentation>Use of the "versioned data" exchange mechanism.
      </xsd:documentation>
    </xsd:annotation>
    <xsd:complexContent>
      <xsd:extension base="Operation">
        <xsd:attribute name="type" type="TypeQName" use="required"/>
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>
  <xsd:complexType name="Event">
    <xsd:annotation>
      <xsd:documentation>Use of the "event" exchange mechanism.
      </xsd:documentation>
    </xsd:annotation>
    <xsd:complexContent>
      <xsd:extension base="Operation">
        <xsd:sequence>
          <xsd:element maxOccurs="unbounded" minOccurs="0" name="input"
            type="Parameter"/>
        </xsd:sequence>
        <xsd:attribute name="direction" type="E_EventDirection" use="required"/>
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>
  <xsd:complexType name="RequestResponse">
    <xsd:annotation>
      <xsd:documentation>Use of the "request-response" exchange mechanism.
      </xsd:documentation>
    </xsd:annotation>
    <xsd:complexContent>
      <xsd:extension base="Operation">
        <xsd:sequence>
          <xsd:element maxOccurs="unbounded" minOccurs="0" name="input"
            type="Parameter"/>
          <xsd:element maxOccurs="unbounded" minOccurs="0" name="output"
            type="Parameter"/>
        </xsd:sequence>
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>
  <xsd:simpleType name="E_EventDirection">
    <xsd:restriction base="xsd:string">
      <xsd:enumeration value="SENT_BY_PROVIDER"/>
      <xsd:enumeration value="RECEIVED_BY_PROVIDER"/>
    </xsd:restriction>
  </xsd:simpleType>
  <xsd:complexType name="Parameter">
    <xsd:attribute name="name" type="NameId" use="required"/>
    <xsd:attribute name="type" type="TypeQName" use="required"/>
  </xsd:complexType>
</xsd:schema>
```

8.10 ecoa-interface-qos-1.0.xsd

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<xsd:schema xmlns="http://www.ecoa.technology/interface-qos-1.0"
  xmlns:tns="http://www.ecoa.technology/interface-qos-1.0"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  elementFormDefault="qualified" targetNamespace="http://www.ecoa.technology/interface-
qos-1.0">
  <xsd:include schemaLocation="ecoa-common-1.0.xsd"/>
  <xsd:element name="serviceInstanceQoS" type="ServiceInstanceQoS"/>
  <xsd:complexType name="ServiceInstanceQoS">
    <xsd:annotation>
      <xsd:documentation>The definition of an ECOA service, including a set of
        operations.
      </xsd:documentation>
    </xsd:annotation>
    <xsd:sequence maxOccurs="1" minOccurs="1">
      <xsd:element name="operations" type="Operations"/>
    </xsd:sequence>
  </xsd:complexType>
  <xsd:complexType name="Operations">
    <xsd:annotation>
```

```
      <xsd:documentation>A set of named operations.</xsd:documentation>
    </xsd:annotation>
    <xsd:choice maxOccurs="unbounded" minOccurs="1">
      <xsd:element name="data" type="Data"/>
      <xsd:element name="event" type="Event"/>
      <xsd:element name="requestresponse" type="RequestResponse"/>
    </xsd:choice>
  </xsd:complexType>
  <xsd:complexType name="Data">
    <xsd:annotation>
      <xsd:documentation>Use of the "versionned data" exchange mechanism.
      </xsd:documentation>
    </xsd:annotation>
    <xsd:sequence>
      <xsd:element maxOccurs="1" minOccurs="0" name="highestRate" type="OperationRate">
        <xsd:annotation>
          <xsd:documentation>Max number of occurrences within a reference time
            frame
          </xsd:documentation>
        </xsd:annotation>
      </xsd:element>
      <xsd:element maxOccurs="1" minOccurs="0" name="lowestRate" type="OperationRate">
        <xsd:annotation>
          <xsd:documentation>Min number of occurrences within a reference time
            frame
          </xsd:documentation>
        </xsd:annotation>
      </xsd:element>
    </xsd:sequence>
    <xsd:attribute name="name" type="NameId" use="required"/>
    <xsd:attribute name="maxAgeing" type="xsd:double" use="optional">
      <xsd:annotation>
        <xsd:documentation>Operation Provided : max duration between Data
          production (from the source) and the end of writing process.
          Operation Required : max duration between Data production
          (from the source) and the end of reading process.
          Unit is second.
        </xsd:documentation>
      </xsd:annotation>
    </xsd:attribute>
    <xsd:attribute name="notificationMaxHandlingTime" type="xsd:double"
      use="optional">
      <xsd:annotation>
        <xsd:documentation>Notifying data case: maxHandlingTime for notification
          event. Unit is second.
        </xsd:documentation>
      </xsd:annotation>
    </xsd:attribute>
  </xsd:complexType>
  <xsd:complexType name="Event">
    <xsd:annotation>
      <xsd:documentation>Use of the "event" exchange mechanism.
      </xsd:documentation>
    </xsd:annotation>
    <xsd:sequence>
      <xsd:element maxOccurs="1" minOccurs="0" name="highestRate" type="OperationRate">
        <xsd:annotation>
          <xsd:documentation>Max number of occurrences within a reference time
            frame
          </xsd:documentation>
        </xsd:annotation>
      </xsd:element>
      <xsd:element maxOccurs="1" minOccurs="0" name="lowestRate" type="OperationRate">
        <xsd:annotation>
          <xsd:documentation>Min number of occurrences within a reference time
            frame
          </xsd:documentation>
        </xsd:annotation>
      </xsd:element>
    </xsd:sequence>
    <xsd:attribute name="name" type="NameId" use="required"/>
    <xsd:attribute name="maxHandlingTime" type="xsd:double" use="optional">
      <xsd:annotation>
        <xsd:documentation>Event Sent : specifies an intent on receivers for
          maximal duration between Event Reception and end of related processing
          Event Received : maximal duration between Event Received and end of
          related processing.
          Unit is second.
        </xsd:documentation>
      </xsd:annotation>
    </xsd:attribute>
  </xsd:complexType>
  <xsd:complexType name="RequestResponse">
    <xsd:annotation>
      <xsd:documentation>Use of the "request-reply" exchange mechanism.
      </xsd:documentation>
    </xsd:annotation>
    <xsd:sequence>
      <xsd:element maxOccurs="1" minOccurs="0" name="highestRate" type="OperationRate">
        <xsd:annotation>
          <xsd:documentation>Max number of occurrences within a reference time
            frame
          </xsd:documentation>
        </xsd:annotation>
      </xsd:element>
      <xsd:element maxOccurs="1" minOccurs="0" name="lowestRate" type="OperationRate">
        <xsd:annotation>
          <xsd:documentation>Min number of occurrences within a reference time
            frame
          </xsd:documentation>
        </xsd:annotation>
      </xsd:element>
    </xsd:sequence>
    <xsd:attribute name="name" type="NameId" use="required"/>
    <xsd:attribute name="maxResponseTime" type="xsd:double" use="optional">
      <xsd:annotation>
        <xsd:documentation>Operation Provided : maximal duration between Request
          Reception and Callback Sent
          Operation Required : maximal duration between Request Sent and
          Callback reception.
          Unit is second.
        </xsd:documentation>
      </xsd:annotation>
    </xsd:attribute>
    <xsd:attribute name="callbackMaxHandlingTime" type="xsd:double"
      use="optional">
      <xsd:annotation>
        <xsd:documentation>maxHandlingTime to execute the callback entry-point.
          Unit is second.
        </xsd:documentation>
      </xsd:annotation>
    </xsd:attribute>
  </xsd:complexType>
  <xsd:complexType name="OperationRate">
    <xsd:attribute name="numberOfOccurrences" type="xsd:decimal" use="optional">
      <xsd:annotation>
        <xsd:documentation>Min or max number of operations occurring during a
          specified duration
        </xsd:documentation>
      </xsd:annotation>
    </xsd:attribute>
    <xsd:attribute name="timeFrame" type="xsd:double" use="optional">
      <xsd:annotation>
        <xsd:documentation>Equal to min or max inter-arrival time when
          NumberOfOccurrences value is 1.
          In other cases, specifies a sizing duration for operations bursts.
          Unit is second.
        </xsd:documentation>
      </xsd:annotation>
    </xsd:attribute>
  </xsd:complexType>
</xsd:schema>
```

8.11 ecoa-logicalsystem-1.0.xsd

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<xsd:schema xmlns="http://www.ecoa.technology/logicalsystem-1.0"
  xmlns:tns="http://www.ecoa.technology/logicalsystem-1.0"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://www.ecoa.technology/logicalsystem-1.0">
  <xsd:element name="logicalSystem">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element maxOccurs="unbounded" name="logicalComputingPlatform">
          <xsd:complexType>
            <xsd:sequence>
              <xsd:element maxOccurs="unbounded" name="logicalComputingNode">
                <xsd:complexType>
                  <xsd:sequence>
                    <xsd:element name="endianess">
                      <xsd:complexType>
                        <xsd:attribute name="type" use="required">
                          <xsd:simpleType>
                            <xsd:restriction base="xsd:string">
                              <xsd:enumeration value="BIG"/>
                              <xsd:enumeration value="LITTLE"/>
                            </xsd:restriction>
                          </xsd:simpleType>
                        </xsd:attribute>
                      </xsd:complexType>
                    </xsd:element>
                    <xsd:element maxOccurs="unbounded" name="logicalProcessors">
                      <xsd:complexType>
                        <xsd:sequence>
                          <xsd:element name="stepDuration">
                            <xsd:complexType>
                              <xsd:attribute name="nanoSeconds" type="xsd:integer"
                                use="required"/>
                            </xsd:complexType>
                          </xsd:element>
                        </xsd:sequence>
                        <xsd:attribute name="type" type="xsd:string"
                          use="required"/>
                        <xsd:attribute name="number" type="xsd:integer"
                          use="required"/>
                      </xsd:complexType>
                    </xsd:element>
                    <xsd:element name="os">
                      <xsd:complexType>
                        <xsd:attribute name="name" use="required">
                          <xsd:simpleType>
                            <xsd:restriction base="xsd:string">
                              <xsd:enumeration value="windows"/>
                              <xsd:enumeration value="linux"/>
                              <xsd:enumeration value="fastos"/>
                              <xsd:enumeration value="ims-vxworks"/>
                              <xsd:enumeration value="ima-integrity"/>
                            </xsd:restriction>
                          </xsd:simpleType>
                        </xsd:attribute>
                        <xsd:attribute name="version" type="xsd:string"
                          use="optional"/>
                      </xsd:complexType>
                    </xsd:element>
                    <xsd:element name="availableMemory">
                      <xsd:complexType>
                        <xsd:attribute name="gigaBytes" type="xsd:integer"
                          use="required"/>
                      </xsd:complexType>
                    </xsd:element>
                    <xsd:element name="moduleSwitchTime">
                      <xsd:complexType>
                        <xsd:attribute name="microSeconds" type="xsd:integer"
                          use="required"/>
                      </xsd:complexType>
                    </xsd:element>
                  </xsd:sequence>
                  <xsd:attribute name="id" type="xsd:string" use="required"/>
                </xsd:complexType>
              </xsd:element>
              <xsd:element maxOccurs="unbounded" minOccurs="0"
                name="logicalComputingNodeLinks">
                <xsd:complexType>
```

```
          <xsd:sequence>
            <xsd:element maxOccurs="unbounded" name="link">
              <xsd:complexType>
                <xsd:sequence>
                  <xsd:element name="throughput">
                    <xsd:complexType>
                      <xsd:attribute name="megaBytesPerSecond"
                        type="xsd:integer" use="required"/>
                    </xsd:complexType>
                  </xsd:element>
                  <xsd:element name="latency">
                    <xsd:complexType>
                      <xsd:attribute name="microSeconds" type="xsd:integer"
                        use="required"/>
                    </xsd:complexType>
                  </xsd:element>
                </xsd:sequence>
                <xsd:attribute name="id" type="xsd:ID"/>
                <xsd:attribute name="to" type="xsd:string" use="required"/>
                <xsd:attribute name="from" type="xsd:string"
                  use="required"/>
              </xsd:complexType>
            </xsd:element>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>
    </xsd:sequence>
    <xsd:attribute name="id" type="xsd:ID" use="required"/>
  </xsd:complexType>
</xsd:element>
<xsd:element maxOccurs="unbounded" minOccurs="0"
  name="logicalComputingPlatformLinks">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element maxOccurs="unbounded" name="link">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="throughput">
              <xsd:complexType>
                <xsd:attribute name="megaBytesPerSecond" type="xsd:integer"
                  use="required"/>
              </xsd:complexType>
            </xsd:element>
            <xsd:element name="latency">
              <xsd:complexType>
                <xsd:attribute name="microSeconds" type="xsd:integer"
                  use="required"/>
              </xsd:complexType>
            </xsd:element>
          </xsd:sequence>
          <xsd:attribute name="id" type="xsd:ID"/>
          <xsd:attribute name="to" type="xsd:string" use="required"/>
          <xsd:attribute name="from" type="xsd:string" use="required"/>
        </xsd:complexType>
      </xsd:element>
    </xsd:sequence>
```

```
          </xsd:complexType>
        </xsd:element>
      </xsd:sequence>
      <xsd:attribute name="id" type="xsd:ID" use="required"/>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

8.12 ecoa-module-behaviour-1.0.xsd

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<xsd:schema xmlns="http://www.ecoa.technology/module-behaviour-1.0"
  xmlns:tns="http://www.ecoa.technology/module-behaviour-1.0"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  elementFormDefault="qualified" targetNamespace="http://www.ecoa.technology/module-
behaviour-1.0">

  <!-- Behaviour of module operations -->

  <xsd:include schemaLocation="ecoa-common-1.0.xsd"/>
  <xsd:element name="moduleBehaviour" type="ModuleBehaviour"/>
  <xsd:complexType name="ModuleBehaviour">
    <xsd:sequence>
      <xsd:element maxOccurs="unbounded" minOccurs="0" ref="use"/>
      <xsd:element maxOccurs="1" minOccurs="1" name="moduleConfiguration"
        type="ModuleConfiguration"/>
      <xsd:element maxOccurs="unbounded" minOccurs="1" name="entryPoint"
        type="EntryPoint"/>
    </xsd:sequence>
  </xsd:complexType>
  <xsd:complexType name="ModuleConfiguration">
    <xsd:annotation>
      <xsd:documentation>
        Module implementation characteristics
      </xsd:documentation>
    </xsd:annotation>
    <xsd:attribute name="moduleImplementationName" type="NameId" use="required">
      <xsd:annotation>
        <xsd:documentation>
          Reference to a moduleImplementation name
        </xsd:documentation>
      </xsd:annotation>
    </xsd:attribute>
    <xsd:attribute default="1" name="queueDepth" type="xsd:positiveInteger"
      use="optional">
      <xsd:annotation>
        <xsd:documentation>
          depth of the incoming operations queue (Warning : to be
          placed in another XML file for next stages)
        </xsd:documentation>
      </xsd:annotation>
    </xsd:attribute>
    <xsd:attribute default="1" name="maxNbOfProcessedOpsPerActivation"
      type="xsd:positiveInteger">
      <xsd:annotation>
```

```
        <xsd:documentation>
          Max number of processed operations per activation (Not for
          stage 1 : to be used later, once activating property on incoming
          operations will be taken into account)
        </xsd:documentation>
      </xsd:annotation>
    </xsd:attribute>
  </xsd:complexType>
  <xsd:complexType name="ActionSet">
    <xsd:annotation>
      <xsd:documentation>A set of actions to be sequentially executed by the
        module
      </xsd:documentation>
    </xsd:annotation>
    <xsd:choice maxOccurs="unbounded">
      <xsd:element name="loop" type="Loop"/>
      <xsd:element name="computing" type="Computing"/>
      <xsd:element name="operationCall" type="OperationCall"/>
    </xsd:choice>
  </xsd:complexType>
  <xsd:complexType name="EntryPoint">
    <xsd:complexContent>
      <xsd:extension base="ActionSet">
        <xsd:attribute name="name" type="NameId" use="required"/>
        <xsd:attribute name="activatingCondition" type="NameId" use="required">
          <xsd:annotation>
            <xsd:documentation>Reference to an incoming operation
            </xsd:documentation>
          </xsd:annotation>
        </xsd:attribute>
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>
  <xsd:complexType name="Loop">
    <xsd:complexContent>
      <xsd:extension base="ActionSet">
        <xsd:attribute name="Iterations" type="xsd:integer" use="required">
          <xsd:annotation>
            <xsd:documentation>Number of iterations</xsd:documentation>
          </xsd:annotation>
        </xsd:attribute>
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>
  <xsd:complexType name="Computing">
    <xsd:attribute name="minComputingSteps" type="xsd:long" use="required">
      <xsd:annotation>
        <xsd:documentation>Minimum number of computing steps</xsd:documentation>
      </xsd:annotation>
    </xsd:attribute>
    <xsd:attribute name="maxComputingSteps" type="xsd:long" use="required">
      <xsd:annotation>
        <xsd:documentation>Maximum number of computing steps</xsd:documentation>
      </xsd:annotation>
    </xsd:attribute>
  </xsd:complexType>
  <xsd:complexType name="OperationCall">
    <xsd:attribute name="moduleOperationRef" type="NameId" use="required">
      <xsd:annotation>
        <xsd:documentation>Reference to a required operation</xsd:documentation>
      </xsd:annotation>
    </xsd:attribute>
    <xsd:attribute default="0" name="minComputingSteps" type="xsd:long"
      use="optional">
      <xsd:annotation>
        <xsd:documentation>Minimum number of computing steps due to the
          operation call itself (not including computing steps to execute the
          operation in the called module)
        </xsd:documentation>
      </xsd:annotation>
    </xsd:attribute>
    <xsd:attribute default="0" name="maxComputingSteps" type="xsd:integer"
      use="optional">
      <xsd:annotation>
        <xsd:documentation>Maximum number of computing steps due to the
          operation call itself (not including computing steps to execute the
          operation in the called module)
        </xsd:documentation>
      </xsd:annotation>
    </xsd:attribute>
  </xsd:complexType>
</xsd:schema>
```

8.13 ecoa-types-1.0.xsd

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<xsd:schema xmlns="http://www.ecoa.technology/types-1.0"
xmlns:tns="http://www.ecoa.technology/types-1.0"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified"
  targetNamespace="http://www.ecoa.technology/types-1.0">
  <xsd:include schemaLocation="ecoa-common-1.0.xsd"/>
  <xsd:simpleType name="E_predef">
    <xsd:annotation>
      <xsd:documentation>Predefined ECOA types</xsd:documentation>
    </xsd:annotation>
    <xsd:restriction base="xsd:string">
      <xsd:enumeration value="boolean8"/>
      <xsd:enumeration value="int8"/>
      <xsd:enumeration value="int16"/>
      <xsd:enumeration value="int32"/>
      <xsd:enumeration value="int64"/>
      <xsd:enumeration value="uint8"/>
      <xsd:enumeration value="uint16"/>
      <xsd:enumeration value="uint32"/>
      <xsd:enumeration value="uint64"/>
      <xsd:enumeration value="char8"/>
      <xsd:enumeration value="byte"/>
      <xsd:enumeration value="float32"/>
      <xsd:enumeration value="double64"/>
    </xsd:restriction>
  </xsd:simpleType>
  <xsd:complexType name="Simple">
    <xsd:annotation>
      <xsd:documentation>A type based on a predefined type (simple or E_predef)
        with a name, min/max constraints, and a unit.
      </xsd:documentation>
    </xsd:annotation>
    <xsd:attribute name="type" type="TypeQName" use="required"/>
    <xsd:attribute name="name" type="TypeName" use="required"/>
    <xsd:attribute name="minRange" type="ConstantReferenceOrFloatValue"
      use="optional"/>
    <xsd:attribute name="maxRange" type="ConstantReferenceOrFloatValue"
      use="optional"/>
    <xsd:attribute name="unit" type="xsd:string" use="optional">
      <xsd:annotation>
        <xsd:documentation>Use of International System units is recommended.
        </xsd:documentation>
      </xsd:annotation>
    </xsd:attribute>
    <xsd:attribute name="comment" type="xsd:string" use="optional"/>
  </xsd:complexType>
  <xsd:complexType name="Constant">
    <xsd:annotation>
      <xsd:documentation>Constant definition</xsd:documentation>
    </xsd:annotation>
    <xsd:attribute name="name" type="TypeName" use="required"/>
    <xsd:attribute name="type" type="TypeQName" use="required"/>
    <xsd:attribute name="value" type="xsd:decimal" use="required"/>
    <xsd:attribute name="comment" type="xsd:string" use="optional"/>
  </xsd:complexType>
  <xsd:simpleType name="ConstantReferenceOrFloatValue">
    <xsd:annotation>
      <xsd:documentation>Use of a constant or of a (float or integer) value.
      </xsd:documentation>
    </xsd:annotation>
    <xsd:restriction base="xsd:string">
      <xsd:pattern
        value="%([A-Za-z][A-Za-z0-9_\.]*:)?[A-Za-z][A-Za-z0-9_]*%|(\+|-)?([0-9]+(\.[0-
9]*)?|\.[0-9]+)([Ee](\+|-)?[0-9]+)?"/>
    </xsd:restriction>
  </xsd:simpleType>
  <xsd:simpleType name="ConstantReferenceOrPositiveIntegerValue">
    <xsd:annotation>
      <xsd:documentation>Use of a constant or of a positive integer value.
      </xsd:documentation>
    </xsd:annotation>
    <xsd:restriction base="xsd:string">
      <xsd:pattern value="%([A-Za-z][A-Za-z0-9_\.]*:)?[A-Za-z][A-Za-z0-9_]*%|[0-9]+"/>
    </xsd:restriction>
  </xsd:simpleType>
  <xsd:simpleType name="ConstantReferenceOrIntegerValue">
    <xsd:annotation>
      <xsd:documentation>Use of a constant or of an integer value.
      </xsd:documentation>
    </xsd:annotation>
    <xsd:restriction base="xsd:string">
```

```
      <xsd:pattern value="%([A-Za-z][A-Za-z0-9_\.]*:)?[A-Za-z][A-Za-z0-9_]*%|(\+|-)?[0-
9]*"/>
    </xsd:restriction>
  </xsd:simpleType>
  <xsd:complexType name="Enum">
    <xsd:annotation>
      <xsd:documentation>Enumerated type</xsd:documentation>
    </xsd:annotation>
    <xsd:sequence>
      <xsd:element maxOccurs="unbounded" name="value" type="EnumValue"/>
    </xsd:sequence>
    <xsd:attribute name="name" type="TypeName" use="required"/>
    <xsd:attribute name="type" type="TypeQName" use="required"/>
    <xsd:attribute name="comment" type="xsd:string" use="optional"/>
  </xsd:complexType>
  <xsd:complexType name="EnumValue">
    <xsd:annotation>
      <xsd:documentation>A possible value of an enumerated type
      </xsd:documentation>
    </xsd:annotation>
    <xsd:attribute name="name" type="TypeName" use="required"/>
    <xsd:attribute name="valnum" type="ConstantReferenceOrIntegerValue"
      use="optional"/>
    <xsd:attribute name="comment" type="xsd:string" use="optional"/>
  </xsd:complexType>
  <xsd:complexType name="FixedArray">
    <xsd:annotation>
      <xsd:documentation>Fixed-size array (size is always equal to max capacity)
      </xsd:documentation>
    </xsd:annotation>
    <xsd:attribute name="itemType" type="TypeQName" use="required"/>
    <xsd:attribute name="maxNumber" type="ConstantReferenceOrPositiveIntegerValue"
      use="required"/>
    <xsd:attribute name="name" type="TypeName" use="required"/>
    <xsd:attribute name="comment" type="xsd:string" use="optional"/>
  </xsd:complexType>
  <xsd:complexType name="Array">
    <xsd:annotation>
      <xsd:documentation>Variable-size (bounded capacity) array
      </xsd:documentation>
    </xsd:annotation>
    <xsd:attribute name="itemType" type="TypeQName" use="required"/>
    <xsd:attribute name="maxNumber" type="ConstantReferenceOrPositiveIntegerValue"
      use="required"/>
    <xsd:attribute name="name" type="TypeName" use="required"/>
    <xsd:attribute name="comment" type="xsd:string" use="optional"/>
  </xsd:complexType>
  <xsd:complexType name="Record">
    <xsd:annotation>
      <xsd:documentation>A record with named fields (Ada record, C struct)
      </xsd:documentation>
    </xsd:annotation>
    <xsd:sequence>
      <xsd:element maxOccurs="unbounded" name="field" type="Field"/>
    </xsd:sequence>
    <xsd:attribute name="name" type="TypeName" use="required"/>
    <xsd:attribute name="comment" type="xsd:string" use="optional"/>
  </xsd:complexType>
  <xsd:complexType name="Field">
    <xsd:attribute name="name" type="NameId" use="required"/>
    <xsd:attribute name="type" type="TypeQName" use="required"/>
    <xsd:attribute name="comment" type="xsd:string" use="optional"/>
  </xsd:complexType>
  <xsd:complexType name="VariantRecord">
    <xsd:annotation>
      <xsd:documentation>A record with variable parts: each "union" exists only
        if the selector has the value given by the "when" attribute.
      </xsd:documentation>
    </xsd:annotation>
    <xsd:sequence>
      <xsd:element maxOccurs="unbounded" minOccurs="0" name="field" type="Field"/>
      <xsd:element maxOccurs="unbounded" name="union" type="Union"/>
    </xsd:sequence>
    <xsd:attribute name="name" type="TypeName" use="required"/>
    <xsd:attribute name="selectName" type="NameId" use="required"/>
    <xsd:attribute name="selectType" type="TypeQName" use="required"/>
    <xsd:attribute name="comment" type="xsd:string" use="optional"/>
  </xsd:complexType>
  <xsd:complexType name="Union">
    <xsd:attribute name="name" type="NameId" use="required"/>
    <xsd:attribute name="type" type="TypeQName" use="required"/>
    <xsd:attribute name="when" type="xsd:string" use="required"/>
    <xsd:attribute name="comment" type="xsd:string" use="optional"/>
  </xsd:complexType>
  <xsd:complexType name="DataTypes">
    <xsd:annotation>
      <xsd:documentation>A set of data type definitions</xsd:documentation>
    </xsd:annotation>
    <xsd:choice maxOccurs="unbounded" minOccurs="0">
      <xsd:element name="simple" type="Simple"/>
      <xsd:element name="record" type="Record">
        <xsd:unique name="field">
          <xsd:selector xpath="tns:field"/>
          <xsd:field xpath="@name"/>
        </xsd:unique>
      </xsd:element>
      <xsd:element name="constant" type="Constant"/>
      <xsd:element name="variantRecord" type="VariantRecord">
        <xsd:unique name="fieldunion">
          <xsd:selector xpath="tns:field|tns:union"/>
          <xsd:field xpath="@name"/>
        </xsd:unique>
      </xsd:element>
      <xsd:element name="array" type="Array"/>
      <xsd:element name="fixedArray" type="FixedArray"/>
      <xsd:element name="enum" type="Enum">
        <xsd:unique name="value">
          <xsd:selector xpath="tns:value"/>
          <xsd:field xpath="@name"/>
        </xsd:unique>
        <xsd:unique name="valnum">
          <xsd:selector xpath="tns:value"/>
          <xsd:field xpath="@valnum"/>
        </xsd:unique>
      </xsd:element>
    </xsd:choice>
  </xsd:complexType>
  <xsd:complexType name="Library">
    <xsd:annotation>
      <xsd:documentation>A set of data types in a library</xsd:documentation>
    </xsd:annotation>
    <xsd:sequence>
      <xsd:element maxOccurs="unbounded" minOccurs="0" ref="use"/>
      <xsd:element name="types" type="DataTypes">
        <xsd:unique name="typename">
          <xsd:selector xpath="*"/>
          <xsd:field xpath="@name"/>
        </xsd:unique>
      </xsd:element>
    </xsd:sequence>
  </xsd:complexType>
  <xsd:element name="library" type="Library"/>
</xsd:schema>
```

8.14 ecoa-project-1.0.xsd

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<xsd:schema xmlns="http://www.ecoa.technology/project-1.0"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  elementFormDefault="qualified" targetNamespace="http://www.ecoa.technology/project-
1.0">
  <xsd:include schemaLocation="ecoa-common-1.0.xsd"/>

  <xsd:element name="ECOAProject" type="EcoaProject"/>


  <xsd:complexType name="EcoaProject">
    <xsd:annotation>
      <xsd:documentation>
        Describes a whole ECOA project
      </xsd:documentation>
    </xsd:annotation>
    <xsd:sequence>
      <xsd:choice maxOccurs="unbounded" minOccurs="0">
        <xsd:element maxOccurs="unbounded" minOccurs="0" name="serviceDefinitions"
          type="Files"/>
        <xsd:element maxOccurs="unbounded" minOccurs="0"
          name="componentDefinitions" type="Files"/>
        <xsd:element maxOccurs="unbounded" minOccurs="0" name="types"
          type="Files"/>
        <xsd:element maxOccurs="unbounded" minOccurs="0" name="initialAssembly"
          type="xsd:anyURI"/>
        <xsd:element maxOccurs="unbounded" minOccurs="0"
          name="componentImplementations" type="Files"/>
        <xsd:element maxOccurs="unbounded" minOccurs="0" name="logicalSystem"
          type="xsd:anyURI"/>
        <xsd:element maxOccurs="unbounded" minOccurs="0" name="deploymentSchema"
```

```
          type="xsd:anyURI"/>
        <xsd:element maxOccurs="unbounded" minOccurs="0" name="outputDirectory"
          type="xsd:anyURI"/>
        <xsd:element maxOccurs="unbounded" minOccurs="0"
          name="implementationAssembly" type="xsd:anyURI"/>
      </xsd:choice>
    </xsd:sequence>
    <xsd:attribute name="name" type="xsd:string" use="required"/>
  </xsd:complexType>

  <xsd:complexType name="Files">
    <xsd:annotation>
      <xsd:documentation>List of files</xsd:documentation>
    </xsd:annotation>
    <xsd:sequence>
      <xsd:element maxOccurs="unbounded" name="file" type="xsd:anyURI"/>
    </xsd:sequence>
  </xsd:complexType>


</xsd:schema>
```

8.15 ecoa-udpbinding-1.0.xsd

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<xsd:schema xmlns="http://www.ecoa.technology/udpbinding-1.0"
xmlns:tns="http://www.ecoa.technology/udpbinding-1.0"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified"
  targetNamespace="http://www.ecoa.technology/udpbinding-1.0">
  <xsd:element name="platform">
    <xsd:complexType>
      <xsd:attribute name="platformId" type="PlatformID" use="required"/>
      <xsd:attribute name="name" type="xsd:string" use="required"/>
      <xsd:attribute default="256" name="maxChannels" type="xsd:positiveInteger"
        use="optional"/>
      <xsd:attribute name="receivingPort" type="xsd:positiveInteger"
        use="required"/>
      <xsd:attribute name="receivingMulticastAddress" type="xsd:string"
        use="required"/>
    </xsd:complexType>
  </xsd:element>
  <xsd:element name="UDPBinding">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element maxOccurs="unbounded" ref="platform"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
  <xsd:simpleType name="PlatformID">
    <xsd:annotation>
      <xsd:documentation>
        PlatformID is used to identify uniquely each platform within
        ELI-UDP exchanges.
        It is assumed that no more than 16 platforms will be connected together.
      </xsd:documentation>
    </xsd:annotation>
    <xsd:restriction base="xsd:unsignedInt">
      <xsd:minInclusive value="0"/>
      <xsd:maxInclusive value="15"/>
    </xsd:restriction>
  </xsd:simpleType>
</xsd:schema>
```

8.16 ecoa-uid-1.0.xsd

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<xsd:schema xmlns="http://www.ecoa.technology/uid-1.0"
xmlns:tns="http://www.ecoa.technology/uid-1.0"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified"
  targetNamespace="http://www.ecoa.technology/uid-1.0">

  <xsd:element name="ID_map" type="ID_map">
    <!-- each "key" attribute shall be unique -->
    <xsd:key name="key">
      <xsd:selector xpath="tns:ID"/>
      <xsd:field xpath="@key"/>
    </xsd:key>
    <!-- each "value" attribute shall be unique -->
    <xsd:key name="value">
      <xsd:selector xpath="tns:ID"/>
      <xsd:field xpath="@value"/>
    </xsd:key>
  </xsd:element>

  <xsd:complexType name="ID_map">
    <xsd:sequence>
      <xsd:element maxOccurs="unbounded" minOccurs="0" name="ID" type="ID"/>
    </xsd:sequence>
  </xsd:complexType>

  <xsd:complexType name="ID">
    <xsd:attribute name="key" type="xsd:string" use="required"/>
    <xsd:attribute name="value" type="xsd:int" use="required"/>
  </xsd:complexType>

</xsd:schema>
```

8.17 sca-1.1-cd06-subset.xsd

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!-- Copyright(C) OASIS(R) 2005,2010. All Rights Reserved. OASIS trademark, IPR and other
policies apply. -->
<!-- This file is a derivative work of the original OASIS XSD file: -->
<!-- sca-1.1-cd06.xsd -->
<!-- In that sense, ECOA is not fully compliant with the OASIS SCA specification. -->
<!-- This file is provided to help users to check more easily their ECOA XML files. -->
<!-- Other verification means may be used. -->
```

```
<schema xmlns="http://www.w3.org/2001/XMLSchema" xmlns:sca="http://docs.oasis-
open.org/ns/opencsa/sca/200912"
  targetNamespace="http://docs.oasis-open.org/ns/opencsa/sca/200912">

  <include schemaLocation="sca-core-1.1-cd06-subset.xsd"/>

  <include schemaLocation="sca-contribution-1.1-cd06-subset.xsd"/>

</schema>
```

8.18 sca-contribution-1.1-cd06-subset.xsd

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!-- Copyright(C) OASIS(R) 2005,2010. All Rights Reserved. OASIS trademark, IPR and other
policies apply. -->
<!-- This file is a derivative work of the original OASIS XSD file: -->
<!-- sca-contribution-1.1-cd06.xsd -->
<!-- In that sense, ECOA is not fully compliant with the OASIS SCA specification. -->
<!-- This file is provided to help users to check more easily their ECOA XML files. -->
<!-- Other verification means may be used. -->

<schema xmlns="http://www.w3.org/2001/XMLSchema"
xmlns:ecoa="http://www.ecoa.technology/sca"
  xmlns:sca="http://docs.oasis-open.org/ns/opencsa/sca/200912"
elementFormDefault="qualified"
  targetNamespace="http://docs.oasis-open.org/ns/opencsa/sca/200912">

  <import namespace="http://www.ecoa.technology/sca" schemaLocation="ecoa-sca-1.0.xsd"/>

  <include schemaLocation="sca-core-1.1-cd06-subset.xsd"/>

  <!-- Contribution -->
  <element name="contribution" type="sca:ContributionType"/>
  <complexType name="ContributionType">
    <complexContent>
      <extension base="sca:CommonExtensionBase">
        <sequence>
          <element maxOccurs="unbounded" minOccurs="0" name="deployable"
            type="sca:DeployableType"/>
        </sequence>
      </extension>
    </complexContent>
  </complexType>

  <!-- Deployable -->
  <complexType name="DeployableType">
    <complexContent>
      <extension base="sca:CommonExtensionBase">
        <sequence>
          <any maxOccurs="unbounded" minOccurs="0" namespace="##other"
            processContents="lax"/>
        </sequence>
        <attribute ref="ecoa:deployment"/>
        <attribute name="composite" type="QName" use="required"/>
      </extension>
    </complexContent>
  </complexType>

</schema>
```

8.19 sca-core-1.1-cd06-subset.xsd

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!-- Copyright(C) OASIS(R) 2005,2010. All Rights Reserved. OASIS trademark, IPR and other
  policies apply. -->
<!-- This file is a derivative work of the original OASIS XSD file: -->
<!-- sca-core-1.1-cd06.xsd -->
<!-- In that sense, ECOA is not fully compliant with the OASIS SCA specification. -->
<!-- This file is provided to help users to check more easily their ECOA XML files. -->
<!-- Other verification means may be used. -->

<schema xmlns="http://www.w3.org/2001/XMLSchema"
xmlns:ecoa="http://www.ecoa.technology/sca"
  xmlns:jxb="http://java.sun.com/xml/ns/jaxb" xmlns:sca="http://docs.oasis-
open.org/ns/opencsa/sca/200912"
  elementFormDefault="qualified" jxb:version="1.0"
  targetNamespace="http://docs.oasis-open.org/ns/opencsa/sca/200912">

  <import namespace="http://www.w3.org/XML/1998/namespace"
schemaLocation="../xml/xml.xsd"/>

  <!-- Workaround to allow within Eclipse the validation of .composite -->
  <import namespace="http://www.ecoa.technology/sca" schemaLocation="ecoa-sca-1.0.xsd"/>

  <!-- Common extension base for SCA definitions -->
  <complexType name="CommonExtensionBase">
    <sequence>
      <element maxOccurs="unbounded" minOccurs="0" ref="sca:documentation"/>
    </sequence>
  </complexType>
  <element name="documentation" type="sca:Documentation"/>
  <complexType mixed="true" name="Documentation">
    <sequence>
      <any maxOccurs="unbounded" minOccurs="0" namespace="##other"
        processContents="lax"/>
    </sequence>
    <attribute ref="xml:lang"/>
  </complexType>
  <!-- Component Type -->
  <element name="componentType" type="sca:ComponentType"/>
  <complexType name="ComponentType">
    <complexContent>
      <extension base="sca:CommonExtensionBase">
        <sequence>
          <choice maxOccurs="unbounded" minOccurs="0">
            <element name="service">
              <complexType>
                <complexContent>
                  <restriction base="sca:ComponentService">
                    <sequence>
                      <sequence>
                        <element ref="ecoa:interface"/>
                      </sequence>
                    </sequence>
                  </restriction>
                </complexContent>
              </complexType>
            </element>
            <element name="reference" type="sca:ComponentTypeReference"/>
            <element name="property" type="sca:Property"/>
          </choice>
        </sequence>
      </extension>
    </complexContent>
  </complexType>
  <!-- Composite -->
  <element name="composite" type="sca:Composite"/>
  <complexType name="Composite">
    <complexContent>
      <extension base="sca:CommonExtensionBase">
        <sequence>
          <choice maxOccurs="unbounded" minOccurs="0">
            <element name="service" type="sca:Service"/>
            <element name="property" type="sca:Property"/>
            <element name="component" type="sca:Component"/>
            <element name="reference" type="sca:Reference"/>
            <element name="wire" type="sca:Wire"/>
          </choice>
        </sequence>
        <attribute name="name" type="NCName" use="required"/>
        <attribute name="targetNamespace" type="anyURI" use="required"/>
      </extension>
    </complexContent>
  </complexType>
  <!-- Contract base type for Service, Reference -->
  <complexType abstract="true" name="Contract">
    <complexContent>
      <extension base="sca:CommonExtensionBase">
        <sequence>
          <element minOccurs="0" ref="ecoa:interface"/>
        </sequence>
        <attribute name="name" type="NCName" use="required"/>
      </extension>
    </complexContent>
  </complexType>
  <!-- Service -->
  <complexType name="Service">
    <complexContent>
      <extension base="sca:Contract">
        <attribute name="promote" type="anyURI" use="required"/>
      </extension>
    </complexContent>
  </complexType>
  <!-- Interface -->
  <element abstract="true" name="interface" type="sca:Interface"/>
  <complexType abstract="true" name="Interface">
```

```
    <complexContent>
      <extension base="sca:CommonExtensionBase"/>
    </complexContent>
  </complexType>
  <!-- Reference -->
  <complexType name="Reference">
    <complexContent>
      <extension base="sca:Contract">
        <attribute name="multiplicity" type="sca:Multiplicity" use="required"/>
        <attribute name="promote" type="sca:listOfAnyURIs" use="required"/>
      </extension>
    </complexContent>
  </complexType>
  <!-- Property -->
  <complexType mixed="true" name="SCAPropertyBase">
    <sequence>
      <any maxOccurs="unbounded" minOccurs="0" namespace="##any" processContents="lax"/>
      <!-- NOT an extension point; This any exists to accept the element-based or complex
        type property i.e. no element-based extension
        point under "sca:property" -->
    </sequence>
    <attribute name="name" type="NCName" use="required"/>
    <attribute name="type" type="QName" use="optional"/>
    <anyAttribute namespace="##other" processContents="lax"/>
  </complexType>
  <complexType mixed="true" name="Property">
    <complexContent mixed="true">
      <extension base="sca:SCAPropertyBase">
        <attribute default="false" name="mustSupply" type="boolean" use="optional"/>
      </extension>
    </complexContent>
  </complexType>
  <complexType mixed="true" name="PropertyValue">
    <complexContent mixed="true">
      <extension base="sca:SCAPropertyBase">
        <attribute name="source" type="string" use="optional"/>
        <attribute name="file" type="anyURI" use="optional"/>
      </extension>
    </complexContent>
  </complexType>
  <!-- WireFormat Type -->
  <element abstract="true" name="wireFormat" type="sca:WireFormatType"/>
  <complexType abstract="true" name="WireFormatType">
    <!-- <anyAttribute namespace="##other" processContents="lax"/> -->
  </complexType>
  <!-- Component -->
  <complexType name="Component">
    <complexContent>
      <extension base="sca:CommonExtensionBase">
        <sequence>
          <element ref="sca:implementation"/>
          <choice maxOccurs="unbounded" minOccurs="0">
            <element name="service" type="sca:ComponentService"/>
            <element name="reference" type="sca:ComponentReference"/>
            <element name="property" type="sca:PropertyValue"/>
            <!-- <element ref="sca:requires"/> -->
            <!-- <element ref="sca:policySetAttachment"/> -->
          </choice>
          <!-- <element ref="sca:extensions" minOccurs="0" maxOccurs="1" /> -->
        </sequence>
        <attribute name="name" type="NCName" use="required"/>
      </extension>
      <!-- <attribute name="autowire" type="boolean" use="optional"/> -->
      <!-- <attribute name="requires" type="sca:listOfQNames" -->
      <!-- use="optional"/> -->
      <!-- <attribute name="policySets" type="sca:listOfQNames" -->
      <!-- use="optional"/> -->
    </complexContent>
  </complexType>
  <!-- Component Service -->
  <complexType name="ComponentService">
    <complexContent>
      <extension base="sca:Contract"/>
    </complexContent>
  </complexType>
  <!-- Component Reference -->
  <complexType name="ComponentReference">
    <complexContent>
      <extension base="sca:Contract">
        <attribute default="1..1" name="multiplicity" type="sca:Multiplicity"
          use="optional"/>
      </extension>
    </complexContent>
  </complexType>
  <!-- Component Type Reference -->
  <complexType name="ComponentTypeReference">
    <complexContent>
      <restriction base="sca:ComponentReference">
        <sequence>
          <element maxOccurs="unbounded" minOccurs="0" ref="sca:documentation"/>
          <element ref="ecoa:interface"/>
        </sequence>
        <attribute name="name" type="NCName" use="required"/>
        <attribute default="1..1" name="multiplicity" type="sca:Multiplicity"
          use="optional"/>
      </restriction>
    </complexContent>
  </complexType>
  <!-- Implementation -->
  <element abstract="true" name="implementation" type="sca:Implementation"/>
  <complexType abstract="true" name="Implementation">
    <complexContent>
      <extension base="sca:CommonExtensionBase"/>
    </complexContent>
  </complexType>
  <!-- Implementation Type -->
  <element name="implementationType" type="sca:ImplementationType"/>
  <complexType name="ImplementationType">
    <complexContent>
      <extension base="sca:CommonExtensionBase">
        <attribute name="type" type="QName" use="required"/>
      </extension>
    </complexContent>
  </complexType>
  <!-- Wire -->
  <complexType name="Wire">
    <complexContent>
      <extension base="sca:CommonExtensionBase">
        <sequence>
          <any maxOccurs="unbounded" minOccurs="0" namespace="##other"
            processContents="lax"/>
        </sequence>
        <attribute name="source" type="anyURI" use="required"/>
        <attribute name="target" type="anyURI" use="required"/>
        <attribute ref="ecoa:rank" use="required"/>
        <attribute ref="ecoa:allEventsMulticasted" use="optional"/>
      </extension>
    </complexContent>
  </complexType>
  <!-- Extensions element -->
  <element name="extensions">
    <complexType>
      <sequence>
        <any maxOccurs="unbounded" minOccurs="1" namespace="##other"
          processContents="lax"/>
      </sequence>
    </complexType>
  </element>
  <!-- Value type definition for property values -->
  <element name="value" type="sca:ValueType"/>
  <complexType mixed="true" name="ValueType">
    <sequence>
      <any maxOccurs="unbounded" minOccurs="0" namespace="##any" processContents="lax"/>
    </sequence>
    <anyAttribute namespace="##any" processContents="lax"/>
  </complexType>
  <!-- Miscellaneous simple type definitions -->
  <simpleType name="Multiplicity">
    <restriction base="string">
      <enumeration value="0..1"/>
      <enumeration value="1..1"/>
      <enumeration value="0..n"/>
      <enumeration value="1..n"/>
    </restriction>
  </simpleType>
  <simpleType name="listOfQNames">
    <list itemType="QName"/>
  </simpleType>
  <simpleType name="listOfAnyURIs">
    <list itemType="anyURI"/>
  </simpleType>
</schema>
```

9 Specifications of the SCA Subset

This section describes in extenso the elements of SCA Assembly not used for the ECOA purpose. Tooling may either explicitly check they are not present in any ECOA XML file or use subsets provided in the schemas pack (sca-1.1-cd06-subset.xsd, sca-core-1.1-cd06-subset.xsd and sca-contribution-1.1-cd06-subset.xsd).

9.1 sca-1.1-cd06.xsd

In this file, apart include elements devoted to "`sca-core-1.1-cd06.xsd`" and "`sca-contribution-1.1-cd06.xsd`", all other include elements are not used and can be commented.

9.2 sca-contribution-1.1-cd06.xsd

The complexType "ContributionType" is used apart the referenced elements `"sca:importBase", "sca:exportBase"` and `"sca:extensions".`

The complexType `"DeployableType"` is used as defined in the original file.

It can be extended with the attribute "`ecoa:deployment`".

The element "`importBase`" and the associated complexType "`Import`" are not used.

The element "`import`" and the associated complexType "`ImportType`" are not used.

The element "`exportBase`" and the associated complexType "`Export`" are not used.

The element "`export`" and the associated complexType "`ExportType`" are not used.

9.3 sca-core-1.1-cd06.xsd

The schema "`sca-policy-1.1-cd04.xsd`" is not used and does not need to be included.

For allowing XML validation with tools such as Eclipse, it may be helpful to import the schema "`ecoa-sca-1.0.xsd`".

The complexType "`CommonExtensionBase`" is used apart its generic attribute element "`anyAttribute`" which can be excluded.

The element "`documentation`" and its associated complexType "`Documentation`" are used as defined in the original file.

The element "`componentType`" is used as defined in the original file.

The complexType "`ComponentType`" is used but :

- Its optional element "`service`" can be restricted to elements of type "`ecoa:interface`",
- The refered element "`sca:extensions`" is useless.

The element “`composite`” is used as defined in the original file.

The complexType “`Composite`” is used apart:

- the referenced elements “`sca:include`”,
- the optional referenced elements “`sca:requires`” and “`sca:policySetAttachment`”,
- the element “`any`”,
- the attributes “`local`”, “`autowire`”, “`requires`” and “`policySets`”.

The complexType “`Contract`” is used apart:

- The referenced elements "`sca:binding`", "`sca:callback`", "`sca:requires`", "`sca:policySetAttachment`" and "`sca:extensions`",
- The attributes “`required`” and “`policySets`”.

The referenced element “`sca:interface`” can be replaced by the referenced element “`ecoa:interface`”.

The complexType “`Service`” is used as defined in the original file.

The element “`interface`” is used as defined in the original file.

The complexType “`Interface` is used apart:

- The optional referenced elements “`sca:requires`” and "`sca:policySetAttachment`",
- The attributes "`remotable`", "`requires`" and "`policySets`".

The complexType “`Reference`” is used apart its attributes “`target`” and “`wiredByImpl`”.

The complexType “`SCAPropertyBase`” is used apart its attributes “`element`”, “`many`”, and “`value`”.

The complexType “`Property`” is used as defined in the original file.

The complexType “`PropertyValue`” is used as defined in the original file.

The element “`binding`” and its associated complexType “`Binding`” are not used.

The element “`bindingType`” and its associated complexType “`BindingType`” are not used.

The element “`wireFormat`” is used as defined in the original file.

The complexType “`WireFormatType`” is used apart its generic attribute element “`anyAttribute`”.

The element “`operationSelector`” and its associated complexType “`OperationSelectorType`” are not used.

The element “`callback`” and its associated complexType “`Callback`” are not used.

The complexType “`Component`” is used apart:

- Its optional referenced elements “`sca:requires`” and "`sca:policySetAttachment`",
- Its referenced elements “`sca:extensions`”
- Its attributes “`autowire`”, “`requires`” and “`policySets`”.

The complexType “`ComponentService`” is used as in the original file.

The complexType “`ComponentReference`” is used apart its attributes “`autowire`”, “`target`”, “`wiredByImpl`” and “`nonOverridable`”.

The complexType “`ComponentTypeReference`” is used apart:

- Its referenced elements "`sca:binding`", "`sca:callback`", "`sca:requires`", "`sca:policySetAttachment`" and "`sca:extensions`"
- Its attributes "`autowire`", "`wiredByImpl`", "`requires`" and "`policySets`"
- Its generic attribute element “`anyAttribute`”.

Its referenced element “`sca:interface`” can be replaced by the referenced element “`ecoa:interface`”.

The element “`implementation`” is used as defined in the original file.

The complexType “`Implementation`” is used apart:

- Its optional referenced elements “`sca:requires`” and "`sca:policySetAttachment`"
- Its attributes “`requires`” and “`policySets`”

The element “`implementationType`” is used as defined in the original file.

The complexType “`Implementation`” is used apart:

- Its sequence of optional anonymous elements (“`any`”),
- Its attributes “`alwaysProvides`” and “`mayProvide`”

The complexType “`Wire`” is used apart its attribute “replace”.

The complexType “`Wire`” is extended with the required attribute “`ecoa:rank`” and the optional attribute “`ecoa:allEventsMulticasted`”.

The element “`include`” and its associated complexType “`Include` are not used.

The element “`extensions`” is used as defined in the original file.

The general attribute “`requires`” is not used.

The general attribute “`callback`” is not used.

The element “`value`” and its associated complexType “`ValueType`” are used as defined in the original file.

The simpleType "`Multiplicity`" is used as defined in the original file.

The simpleType "`OverrideOptions`" is not used.

The simpleTypes "`listOfQNames`" and "`listOfAnyURIs`" are used as they are defined in the original file.

The simpleType “`CreateResource`” is not used.