EODiSP Project

Software Requirements Document

Prepared by P&P Software GmbH with ETH-Zürich
for ESA-Estec under Contract 18833/05/NL/AR

Written By:	I. Birrer (P&P Software GmbH)
	M. Egli (ETH-Zurich)
	A. Pasetti (P&P Software GmbH)
Date:	4 November 2005
Issue	1.1
Reference:	PP-SRD-EOP-0001

Table of Contents

1 Glossary and Acronyms

The table defines the most important technical terms and abbreviations used in the proposal.

Term	Short Definition
Abstract Interface	A definition of the signature and semantics of a set of related operations without any implementation details.
Alternative Flow	Textual description of what can happen in addition to the steps in the main success scenario. This can be an execution branch or exception occurring during execution of the main steps.
AOCS	The Attitude and Orbit Control Subsystem of satellites.
API	Application Programming Interface. A set of definitions of the ways one piece of computer software communicates with another.
Application Instantiation	The process whereby a component-based application is constructed by configuring and linking individual components.
Chsm	Concurrent Hierarchical State Machine
Component	A unit of binary reuse that exposes one or more interfaces and that is seen by its clients only in terms of these interfaces.
Component-Based Framework	A software framework that has components as its building blocks.
Computational Node	A computational resource that has memory and processing capabilities.
CORBA	A widely used middleware infrastructure.
Design Pattern	A description of an abstract design solution for a common
DSL	Domain Specific Language (a language that is created to describe applications or components in a very narrow domain).
DTD	Document Type Definition. It defines the legal building blocks of an XML document. It defines the document structure with a list of legal elements. Its purpose is similar to the one of an XML Schema, although it is not as feature rich and the syntax is different.
EO	Earth Observation
EODiSP	Earth Observation Distributed Simulation Environment (the environment to be developed in this study).
EODiSP Framework	The software framework provided by the EODiSP.
EODiSP Middleware	The middleware selected for the EODiSP.
Federate	An application that may be or is currently coupled with other software applications under a Federation Object Model Document Data (FDD) and a runtime infrastructure (RTI).
Federation	A named set of federate applications and a common Federation Object Model (FOM) that are used as a whole to achieve some specific objective.
Federation Execution	The actual operation, over time, of a set of joined federates that are interconnected by a runtime infrastructure (RTI).
Federation Object Model (FOM)	A specification defining the information exchanged at runtime to achieve a given set of federation objectives. This includes object classes, object class attributes, interaction classes, interaction parameters, and other relevant information.
Framework Domain	The set of functionalities whose implementation is supported by the framework.
Framework Instantiation	The process whereby a framework is adapted to the needs of a specific application within its domain.
Generative Programming	A software engineering paradigm that promotes the automatic generation of an implementation from a set of specifications.
HLA	High Level Architecture. A standard to provide a common architecture for distributed modeling and simulation. Available as IEEE standard 1516.
ISP	Internet Service Provider.
JNI	Java Native Interface, a mechanism for interfacing Java code with non-Java code.
JVM	Java Virtual Machine.
JXTA	A network infrastructure aimed at peer to peer (P2P) networks. The core is a set of specifications for which a Java and a C implementation is available.

Main Success Scenario	Textual description of interactions between the primary actor and the system. The main success scenario only describes one flow of execution, without branches or exceptions. This flow is the execution in most cases for most actors.
Model Owner	The model owner is a person in charge of one or more simulation models. The model owner decides when to make his simulation models available to a simulation and when to terminate their availability. The model owner interacts with the EODiSP through a Model Manager Application.
Object Oriented Framework	A framework that uses inheritance and object composition as its chief adaptation mechanisms.
OBS	The On-Board Software.
Primary Actor	A primary actor is one having a goal requiring the assistance of the system. A use case describes how the primary actor can achieve this goal.
Runtime Infrastructure (RTI)	The software that provides common interface services during a High Level Architecture (HLA) federation execution for synchronizing and data exchange.
Simulation Manager Application	A GUI-based environment. Through this environment, a simulation owner can perform the tasks to overall control a simulation. This includes the control of the configuration and tasks like start, stop or hold a simulation experiment.
Simulation Model Application	A GUI-based environment. Through this environment, a model owner can perform the tasks to overall control the models he is in charge of.
Simulation Object Model (SOM)	A specification of the types of information that an individual federate could provide to High Level Architecture (HLA) federations as well as the information that an individual federate can receive from other federates in HLA federations.
Simulation Experiment	A set of one or more simulation run executed in sequence with different configurations.
Simulation Owner	This is the person who is in overall control of a complete simulation. The simulation owner decides how the simulation models should be configured and when a simulation should start and terminate. The simulation owner interacts with the EODiSP through the Simulation Manager Application.
Simulation Package	A piece of software that implements part of the functionalities required for a simulation run and that is delivered as a single unit.
Simulation Run	A single end-to-end simulation for one particular configuration of a set of simulation packages.
Software Framework	A reusable artifact that captures the commonalities of a set of applications in a specific domain and provides reusable software building blocks to facilitate the instantiation of applications in that domain.
SMP2	Simulation Model Portability, a set of interfaces to support the development of simulation applications.
Use Case	Textual description of how an actor can achieve a desired goal by interacting with the system and how the system reacts to these interactions.
UUID	Universally Unique Identifier. An identifier standard to uniquely identify a ressource or an information. It it 128 Bit long and is usually generated automatically.
XML	Extensible Markup Language. XML documents consist (mainly) of text and tags, and the tags imply a tree structure upon the document. An XML document is said to be valid if it conforms to an XML Schema or a DTD.
XML Schema	The XML Schema language is also refered to as XML Schema Definition (XSD). They provide a means for defining the structure, contents and semantics of XML documents. XML Schemas are written in XML.
XRTI	An implementation of the HLA runtime infrastructure (RTI).
XSD	An XML-based language for defining the structure of an XML document. XML Schemas are normally written in the XSD language.
XSL	An XML-based programming language for defining transformations of XML documents. XSL programs can also be used as code generators.

2 References

[Chsm] Paul. J. Lucas, Concurrent Hierarchical State Machine

[Coc] Alistair Cockburn, Use Cases online resources, http://alistair.cockburn.us/usecases/usecases.html

[Coc00] Alistair Cockburn, Writing Effective Use Cases, 2000

[EaC04] D. P. Donovan et. al., The EarhCARE Simulatior, User Guide and Final Report, ESA Contract No. 15346/01/NL/MM, 12 December 2004

[Har87] D. Harel, Statecharts: A Visual Formalism for Complex Systems, 1987

[Hla00] IEEE Standard For Modeling and Simulation (M&S) High Level Architecture (HLA) - Federate Interface Specification, 2000, ISBN 0-7381-2621-7, E-ISBN 0-7381-2622-5

[Jfc] JFreeChart, a free Java class library for generating charts, http://www.jfree.org/jfreechart/index.php

[Lar02] Craig Larman, Applying UML and Patterns, 2002

[Omt00] IEEE Standard For Modeling and Simulation (M&S) High Level Architecture (HLA) - Object Model Template Specification, 2000, ISBN 0-7381-2623-3, E-ISBN 0-7381-2624-1

[Smpc05] Peter Fritzen, Peter Ellsiepen, Anthony Walsh, SMP 2.0 Component Model, Issue 1 Revision 1, EGOS-SIM-GEN-TN-0101, 2005

[Swt] The Standard Widget Toolkit, http://www.eclipse.org/swt/

[Urd] M. Egli, A. Pasetti, I. Birrer, Concept Definition Phase and User Requirements, PP-TN-EOP-0001, 2005

3 Introduction

This document reports the results of the activities performed in WP 310 of the *Earth Observation Distributed Simulation Platform* (EODiSP) project.

The objective of the EODiSP project is to develop a generic platform to support the development of distributed simulation environments that integrate reusable simulation packages. Within the EODiSP project, the objective of WP 310 is to define the software requirements of the EODiSP.

This document defines and justifies the software requirements for the EODiSP. The EODiSP software requirements are mostly derived from the EODiSP concept and from the EODiSP user requirements specified in reference [Urd]. Major deviations from the requirements of reference [Urd] are discussed in section 4. A complete traceability matrix to the EODiSP user requirements is given in appendix A.

In many cases, software requirements are defined using formalisms such as XML Schemas or the *chsm* language to describe state machines. In those cases, the body of this document only gives an informal and easier to read version of the requirements. The full definition of the requirements using the selected formalism is presented in appendices B and C.

3.1 General Approach

The software requirements translate the user requirements to create a new description of the target application. This new description is intended to drive the software design process. Consequently, it should be expressed in terms of an unambiguous *logical model*. The logical model is a description of the target system in a formal language. Ideally, the logical model should be *executable* (to allow the requirements to be simulated), *verifiable* (to allow properties to be proven at the requirements level), and *compilable* (to allow automatic translation to an implementation).

In practice, it is normally impossible to create a logical model for all parts of a complex application. Parts that are not covered by the logical model must then be expressed using requirements expressed in natural language. In this case, use of a structured approach is the best option to improve the quality of the requirements.

When no structured approach is possible to formulate the software requirements, then conventional UR-like requirements must be used. In this case, the software requirements become a refinement of the user requirements. If the user requirements were already well-defined or if their refinement is either impossible or undesirable, the software requirements may actually coincide with the user requirements.

In summary, during the software requirement definition process, the user requirements can be mapped as follows:

- to a logical model, or
- to requirements in structured natural language, or

- to requirements in unstructured natural language (possibly identical to the user requirements from which they are derived)

In the EODiSP project, all three types of mappings are used. In order to give an overview of the mapping approach, it is useful to divide the EODiSP into four parts:

- The *Graphical User Interface* (GUI) that controls the interaction between the user and the EODiSP,
- The *HLA Core* that implements the subset of the HLA selected for the EODiSP,
- The *JXTA Infrastructure* that implements the distribution services for the EODiSP using the JXTA distribution infrastructure, and
- The *Wrapper Generators* that control the generation of the wrappers for selected kinds of simulation packages to be integrated in the EODiSP.

For the GUI part, use cases are used to define the interaction between the users and the EODiSP and XML Schemas are used to define the structure of the configuration information to be provided by the user for an EODiSP run. The use case approach is semi-formal in the sense that the specification of the use cases follows a structured method but does not use a formal language. The XML Schema approach is more formal because XML Schemas are expressed in the XSD language and can be directly used in the EODiSP implementation to check user inputs or to automatically generate editors for entering user inputs.

The HLA Core part of the EODiSP is defined by a set of state machines. The state machines are specified through a formal language [Chsm]. The state machine definition expressed in chsm can be automatically translated into Java source code that implements the state machines. The chsm state machine description could in principle be made executable and verifiable but this is not in the EODiSP project owing to lack of tool support.

The JXTA part of the EODiSP is specified in natural language. Note that the JXTA infrastructure is essentially “as is” and therefore its associated requirements are not problematic and the need and benefits of a formal approach are correspondingly less felt.

The wrapper part of the EODiSP is specified by using XML Schemas to define the input to the wrapper generator tools. For those simulation packages for which no wrapper code can be generated automatically only an example of the wrapper code is given. In such cases, the structure of the wrappers is specified in natural language.

The EODiSP approach is also illustrated in figure 1. The figures illustrates how the EODiSP user requirements are mapped to three different types of software requirements.

Table 3.1.1 presents the same information in tabular form. It shows the approach that is taken in deriving software requirements for each of the four parts of the EODiSP. The last column in the table points to the sections in this document where the requirements are presented in detail.

Software requirements expressed in unstructured natural language are stated in boxes with the following format:

Ref.	*Requirement*
Sx-y	*<Formulation of the requirement>*

The first column contains an identifier of the requirement. The identifier is formed by the letter 'R' followed by the number 'x' of the section where the requirement is formulated, and by a sequential number 'y' that identifies the requirement within a certain section.

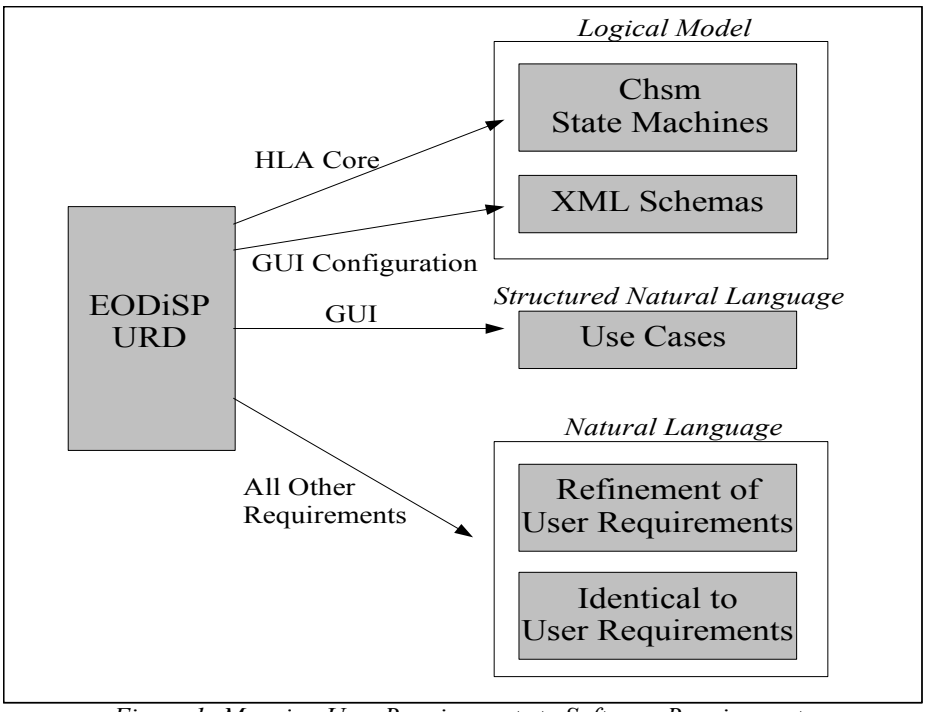

Figure 1: Mapping User Requirements to Software Requirements

Table 3.1.1: Specification Approaches

EODiSP Part	*Approach*	*Section*
GUI	Use Cases (structured) + XML Schemas (logical model)	5, 6
HLA Core	State Machine in chsm language (compilable logical model)	7.1
JXTA Infrastructure	Natural Language	8
Wrappers	XML Schemas (logical model) + Natural Language	9, 10, 11
Residual Req.s	Natural Language	12

3.2 Traceability to User Requirements

All user requirements must be traceable to one or more software requirements. For traceability purposes, software requirements should be identified. This is normally done by assigning

a numerical identifier to each software requirements. Here, and in view of the approach outlined in the previous subsection, four categories of software requirements are recognized with four separate identification policies:

- *Use Case Requirements* are identified by the use case identifier (i.e. one use case is regarded as one single software requirement).
- *State Machine Requirement*s are identified by the name of the state machine (i.e. one state machine is regarded as one single software requirement).
- *XML Schema Requirements* are identified by the name of the XML Schema (i.e. one XML Schema is regarded as one single software requirement).
- *Conventional Requirements* are identified by a numerical identifier of the form Sx-y where 'x' is the section in the present document where the requirement is defined and 'y' is a sequential number that allows the specific requirement within the section to be identified.

As discussed in the previous section, in some cases, user requirements are taken over unchanged and are also used as software requirements. In these cases, and in order to ensure consistency between the URD and the SRD, the text of the requirement is not repeated here. A traceability matrix is given in appendix A. The traceability matrix makes it clear whether a user requirement is mapped to some software requirements or whether it is taken over unchanged and should therefore be considered an integral part of the SRD.

4 Major Deviations from User Requirements

The analyses that led to the definition of the software requirements presented in this document uncovered some inconsistencies in the EODiSP user requirements that make full compliance with the reference [Urd] impossible.

The SRD-level analyses also identified some areas where streamlining and optimization of the requirements of reference [Urd] would be advisable. In these cases, compliance with the EODiSP URD would have been possible but was regarded as unwise.

This section discusses all important cases where the EODiSP software requirements as stated in this document deviate from the user requirements as stated in reference [Urd]. Note that a full traceability matrix from the URD to the SRD can be found in appendix A.

The table below lists in summary form the major deviations from the URD together with a brief statement of their justification and a reference to the subsection in this section where the deviation is discussed in greater detail:

Deviation	***Justification***	***Section***
SOMConfig and FOMConfig files are dropped from the list of simulation manager configuration files.	The function of these two files is incompatible with the use of an HLA-based simulation core.	4.1
Simulation packages cannot ask to be triggered according to a pre-defined schedule.	This functionality cannot be implemented with the set of HLA services baselined for implementation in the EODiSP.	4.2
ExperimentInitConfig and SimulationConfig configuration files are merged.	Optimization of user requirements.	4.3
ModelsConfig and SomSecurityConfig configuration files are merged.	Optimization of user requirements.	4.4
Simulation models cannot exchange data directly without passing through the simulation environment.	This functionality is incompatible with the use of an HLA-based simulation core.	4.5
A description of how the simulation manager application detects the termination of a simulation experiment is missing.	A new section with this description has been added.	4.6

The first, third and fourth deviations do not affect the intrinsic functionality of the EODiSP. They only affect the way the EODiSP is used. The second and fifth deviations instead have an impact on the EODiSP functionality. The last entry is not exactly a deviation in the sense of the other entries in the table, because it is completely new in the SRD. It does not limit or extend the EODiSP functionality, nor does it change its usage.

Finally, it should be stressed that this section has a temporary role only. Eventually, after discussions with ESA, the inconsistencies in the user requirements will have to be removed by updating reference [Urd].

4.1 Definition of a Simulation Configuration

A simulation configuration is defined by defining which simulation models are used in a particular simulation and how they are interconnected.

In the EODiSP user requirements, the definition of a simulation configuration is done through three configuration files – the SimulationConfig, the FOMConfig and SOMConfig configuration files (see section 9 of the URD).

An HLA simulation is constructed as a set of interacting federates. The SimulationsConfig file describes which federates take part in an HLA simulation. Each federate is described by its object classes and attributes. The SOMConfig file is intended to describe how many instances of an object class or an attribute shall be included in a particular simulation execution. The FOMConfig file describes how attribute instances of one object class instance are connected to other attribute instances of another object class instance.

For purposes of illustration, it is useful to consider an example. Consider the case of a user who has a simulation package that models the trajectory of a ballistic rocket. The simulation package is implemented as an executable program that interacts with its environment through input and output files. The input file defines the initial position and velocity of the rocket. The output file (which is refreshed at every simulations step) gives the last computed position of the rocket.

The user wishes to create a simulation where two rockets are launched and their trajectory is recorded by a standard data display package. The data display package must be linked to one or more data sources and simply plots all the data sources on a screen.

With the approach implied by the current EODiSP user requirements, such a simulation would be set up as follows (see Figure 2 for a graphical representation):

- The following two object classes are defined in a FOM: `Rocket` and `DataSource`.
- The rocket simulation package is wrapped as an HLA federate.
- This HLA federate is deployed twice, once for each rocket that is desired to simulate. Each of these federates creates an instance of the object class `Rocket`(*r1* and *r2* in Figure 2).
- The data display package is wrapped as an HLA federate.
- This federate is deployed once. The `DataSource` class is instantiated twice (*ds1* and *ds2* in Figure 2) in this federate, once fore each data source that should be plotted.
- The instantiation policy for the federates is described in the SOMConfig file.The deployment of federates is described in the SimulationsConfig file.

- The four object class instances (two rocket (*r1* and *r2*) and two data source instances (*ds1* and *ds2*) are linked together such that the outputs of the two rocket instances are linked to the inputs of the data source instances.
- The connections among the object instances are described in the FOMConfig file.

The above approach is inspired by the example of Matlab-like simulation environments where users have a palette of pre-defined simulation modules which they can deploy and connect as they wish for each individual simulation.

This approach however is not in line with the HLA spirit. This is because the HLA does not really support the concept of reusable and independently deployable simulation models. An HLA federate is built for a specific simulation and is designed to be embedded within a specific simulation setting (federation). It is normally not possible to change the connections of an HLA federate without changing its implementation.

In the case of the example above, the connection between the rocket federate and the display federate is hardwired in the two federates. It is generally not possible to take the rocket federate, unplug it from the display federate, and plug it into some other federate representing a different simulation model. Note that this swap is not possible even when there is compatibility of number and types of input and output signals.

HLA federates, in other words, are units of encapsulation but they are not units of reuse.

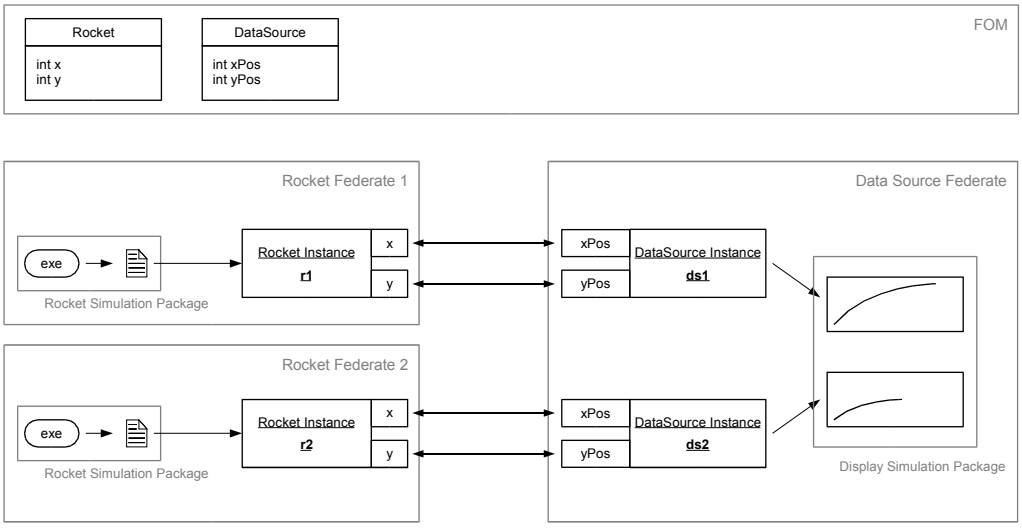

Figure 2: Approach implied by the current EODiSP user requirements

In the EODiSP context, there is an obvious need to have reuse of *simulation packages*. This need is not incompatible with the choice of an HLA-based infrastructure because, in the EODiSP, the simulation packages are transformed into HLA federates through the wrapping process. It therefore becomes possible to have reuse at the level of the *simulation packages* but not at the level of the *HLA federates*.

If the same simulation package is to be reused in two different contexts, then it should be wrapped as two different HLA federates, each adapted to its context. Note that, with this approach, reusability comes to depend on the ease with which simulation packages can be wrapped. This makes the automation of the wrapping process – already foreseen in the EODiSP concept – even more important.

With this second approach, the example simulation with the two rockets would be set up as follows (see Figure 3 for a graphical representation):

- The following object classes are defined in a FOM: `Rocket_1` and `Rocket_2`
- The rocket simulation package is wrapped a first time to generate an HLA federate that exposes object class `Rocket_1`
- This federate is deployed and the generated wrapper automatically creates one instance of the `Rocket_1` object class.
- The rocket simulation package is wrapped a second time to generate a second HLA federate that exposes object class `Rocket_2`
- This federate is deployed and the generated wrapper automatically creates one instance of the `Rocket_2` object class.
- The data display package is wrapped as an HLA federate that is able to receive values from federates that expose either object class `Rocket_1` or `Rocket_2`. The federate itself does not create any instances of object classes.
- This federate is deployed and automatically connects to the object classes `Rocket_1` and `Rocket_2`.

The key difference with respect to the first approach is that federates are only deployed once. Where two instances of the same simulation package are required, two separate federates are generated by two dedicated wrappings of the simulation package, each federate exposing another object class.

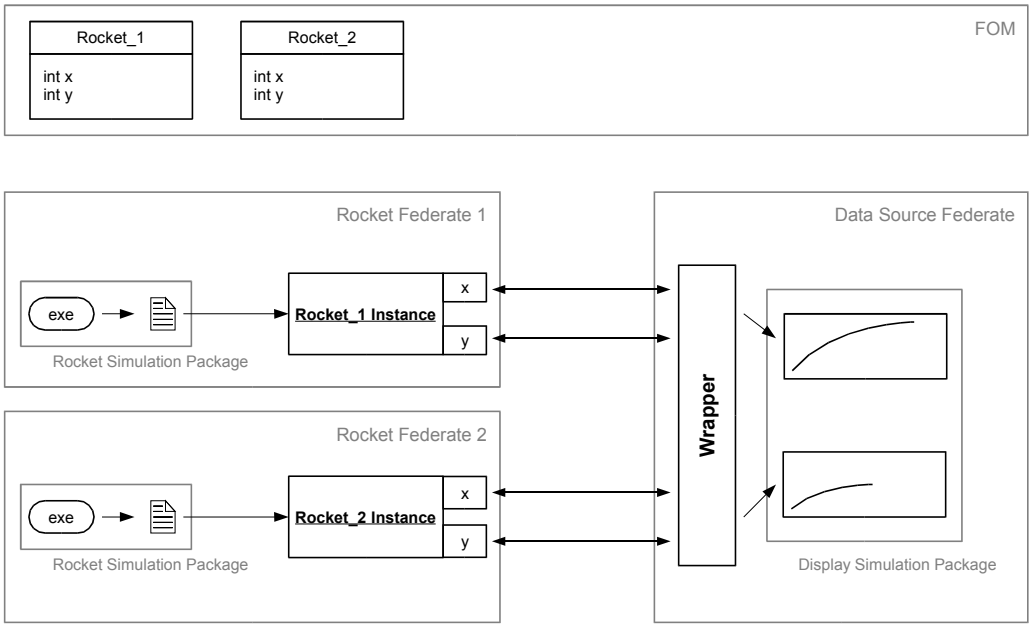

Figure 3: Second Approach without FOMConfig and SOMConfig

A consequence is that there is no need for the SOMConfig and FOMConfig configuration file. The simulation instantiation configuration is now hardcoded in the wrappers of the simulation package. The wrapped simulation package (the HLA federate) is therefore no longer reusable because it is targeted at a specific simulation configuration but the simulation package itself is reusable (but the ease with which it can be reused depends on the ease with which it can be wrapped anew).

In the remainder of this document, the assumption is made that the second approach is adopted.

4.2 Triggering of Simulation Models

In section 4.2 of reference [Urd], a model for the interaction of simulation packages is specified. This model includes the possibility for simulation packages to be triggered according to a pre-defined schedule.

This type of interaction implies that a global simulation time is maintained. This would in turn require the implementation of the HLA Time Management service. This service, however, is currently not supported by the EODiSP (see section 6.7 of [Urd]). There are two basic reasons that led to the decision not to support timing services:

- timing services are especially useful where simulation models can run in parallel but this is not required in the EODiSP, and
- the order of execution of simulation packages in the EODiSP is sequential and packages are triggered not by time but by the arrival of their input values.

In the remainder of this document, the assumption is made that timing services are not implemented by the EODiSP and that therefore the triggering interaction for simulation packages cannot be supported.

The lack of support for a simulation time also means that user requirement R6.4.2-3 cannot be implemented in full. This requirement asks for users to have the ability to predefine the times – either as simulation time or as clock time – when switches from step-by-step to continuous simulation mode and from continuous to step-by-step simulation model should take place. Clearly, the scheduling of the switches based on simulation time is not possible if the HLA timing service is not maintained.

4.3 ExperimentInitConfig and SimulationsConfig Configuration Files

In section 9.5 of reference [Urd], the configuration files for the simulation manager application are specified. Two of these files – the ExperimentInitConfig and SimulationsConfig files – are concerned with the definition of a simulation experiment. They were kept separate because they had two different definition modes (indirect for the SimulationsConfig file, and explicit for the ExperimentInitConfig file).

In order to streamline the configuration process, it has been decided to merge these two files and to have for both the indirect definition mode. The name of the merged file is SimulationManagerProject configuration file.

4.4 ModelsConfig and SomSecuritySetting Configuration Files

In section 9.6 of reference [Urd], the configuration files for the model manager application are specified. Two of these files – the ModelsConfig and SomSecurityConfig files – are concerned with the definition of the simulation models managed by the model manager application.

In order to streamline the configuration process, it has been decided to merge these two files. The name of the merged file is ModelManagerProject configuration file.

4.5 Data Passing Between Simulation Models

Requirement R6.3-1 in reference [Urd] asks for the EODiSP to allow simulation models to exchange data among themselves directly without necessarily passing through the simulation environment.

This requirement is incompatible with the selection of an HLA infrastructure that implements the simulation environment as the RTI and the simulation models as federates. The HLA standard stipulates that all data exchanged between federates must always pass through the RTI. This makes compliance with requirement R6.3-1 impossible.

This introduces a network overhead in the case where two or more models reside on the same node while the RTI is on a remote node. There are three solutions to overcome this problem.

The first offered solution is to run the RTI on the same node as the federates. This is only feasible if the overall network performance is enhanced by this change, which greatly depends on the network topology of the simulation experiment in question. In addition, the per-

son owning the federates has to be the simulation owner as well. If practical, this solution does not violate the HLA standard in any way, but its application is limited.

The second solution is to implement the data transfer between federates as a means of exchanging files. This means that the first federate (the producer) creates a file that includes its calculated values. The second federates (the consumer) reads this values and proceeds the further. Instead of sending the whole file (which can grow large in certain cases) through the RTI, only the information that the value has been updated would be sent to the RTI. This information can also include the path where the file can be read on the file system. The RTI would then only send this information to interested federates. This solution conforms to the HLA standard and is the preferred solution if the former is not applicable. The only restriction of this solution is that it is not applicable when several distributed federates are interested in those values (i.e. in the content of the file). Then, the actual values needs to be sent through the RTI in order to distribute them.

The third solution is not advisable and furthermore more difficult to implement. Therefore it should be considered as a last resort for special cases. The solution would be to wrap all involved simulation packages as a single federate. Firstly, this is a violation to the HLA approach because management data is not sent through the RTI, thus making it impossible to track or log any of the actions that take place between those federates. Furthermore, the data transfer itself has to be implemented by the programmer itself, since wrappers provided by the EODiSP cannot provide this code in any case. As a last point, the wrapping of more than one simulation package as one federate has also be done manually, because a wrapper is always considers only one simulation package.

4.6 Simulation End

A description of one part of a simulation experiment is missing in the URD. This relates to how the simulation manager application is able to detect that a federation execution is completed. The HLA can be used to handle this through two different scenarios.

The first solution is that the *last federate* in a federation execution chain sends an interaction informing all other federates that it has completed its task. This solution has several practical disadvantages. Firstly, the programmer of a federate needs to know in advance that his federate will eventually be the last federate in the execution chain of a certain federation execution. This is often not known at the time when a federate is created. Secondly, it cannot be guaranteed that there is always only one last federate in the chain. Since the EODiSP is a purely data-driven simulation environment, it is possible that two (or more) federates are subscribed to a value from the second-last federate, thus executing in parallel. In such a situation, the decision of which federate shall be responsible of indicating the end of a federation execution can only be guessed. These shortcomings make this solution inapplicable for the EODiSP environment.

The second solution to detect the end of a federation execution is through the use of synchronization points. One federate (or possibly the simulation manager itself for the EODiSP environment) registers a synchronization point within the RTI. Those federates having no opinion about their ending condition achieve the synchronization point immediately. This, for instance, would be the case for some simple federates such as the data transformation feder-

ates, because they would not know how often they are being used throughout a federation execution. Other federates instead know exactly what the ending conditions are. These federates shall only achieve the synchronization point when their ending conditions are fulfilled. The RTI will know that the federation execution is completed when all participating federates have achieved the synchronization point. It will inform all federates to resign from the federation execution in order to destroy it. This approach solves the problems outlined in the first solution above and is therefore preferred.

Using this mechanism also implies that the HLA services concerning synchronization point must be implemented and available. In the URD, the corresponding services are declared as priority 3 (not to be implemented). This needs to be changed for the services listed in table 4.6.1. The first columns lists the section from the HLA standard where the service is defined, and the second column gives the name of the service. These columns are duplicates from the URD. The third column indicates the priority which was assigned to the service by the URD and the last column shows the new priority.

Table 4.6.1: HLA service used to detect federation execution end.

Section	***Service***	***Old Prio.***	***New Prio.***
4.6	Register Federation Synchronization Point	3	1
4.7	Confirm Synchronization Point Registration †	3	1
4.8	Announce Synchronization Point †	3	1
4.9	Synchronization Point Achieved	3	1
4.10	Federation Synchronized †	3	1

The offered solution does not mention the end of a simulation experiment. In fact, only the end of a federation experiment can be detected. This is not a problem, since a simulation experiment consists of one ore more (possibly unrelated) federation executions, whose endings can be determined by the RTI. If all federation executions included in a simulation experiment are completed, the simulation manager knows that the whole experiment is completed.

5 Use Cases

Use cases are one of the techniques that are used to define the EODiSP software requirements. They are in particular used to define the requirements applicable to the GUI part of the EODiSP (see discussion in section 3.1).

This section presents all the use cases defined for the EODiSP GUI. The EODiSP GUI is implemented over three separate applications: the *simulation manager application*, the *model manager application*, and the *support application*. The associated use cases are specified in subsections 5.2, 5.3 and 5.4, respectively.

For readers who are not familiar with the use case concept, subsection 5.1 provides an overview of how they are defined and the purpose they serve.

For convenience, table 5.1 lists all the use cases specified in this document. The first column in the table gives the use case identifier, the second column gives its name, and the last column points to the section where the use case is described.

Table 5.1: List of EODiSP Use Cases.

Use Case ID	***Use Case Name***	***Section***
UC_100	Simulation Manager Application Summary	5.2.1
UC_102	Set-up Simulation Manager Application	5.2.2
UC_104	Configure Simulation Experiment	5.2.3
UC_106	Run Experiment	5.2.4
UC_108	Load or Save Configuration	5.2.5
UC_110	Experiment Abort	5.2.6
UC_112	Configure Simulation Manager Application Log	5.2.7
UC_200	Model Manager Application Summary	5.3.1
UC_202	Set-up Model Manager Application	5.3.2
UC_204	Manage Federates in the Model Manager Application.	5.3.3
UC_208	Configure Model Manager Application Log	5.3.4
UC_302	Generate Wrapper Code for an Excel Workbook	5.4.1
UC_304	Generate Wrapper Code for an SMP2 simulation	5.4.2
UC_306	Generate Wrapper Code for Matlab-Generated Code	5.4.3
UC_308	Generate Wrapper Code for a Matlab Simulation	5.4.4
UC_310	Generate Wrapper Code for Source Code	5.4.5
UC_312	Generate Wrapper Code for a Standalone Executable	5.4.6
UC_314	Generate Wrapper Code a Data Processing Package	5.4.7

Use Case ID	*Use Case Name*	*Section*
UC_316	Creating a New SOM File	5.4.8

The following requirement acts as an umbrella requirement for all the EODiSP use cases. Note that traceability from URD is mostly done in terms of the specific use cases in order to have a better granularity. See also the discussion in section 5.1.1.

S5 -1	*The EODiSP GUI shall support all the use cases listed in table 5.1.*

5.1 Overview of Use Case Concept

Use cases are a good method to describe the interaction between an actor and the system. These entities can be described as follows:

- An actor essentially is the user of the system. This can be a person or another system interacting with the system to achieve a desired goal.
- The system is an entity which reacts to interactions performed by an actor. It is able to achieve a desired goal. The goal which is to be achieved is specified in each use case. In the EODiSP, the system is a piece of runnable software.

A use case describes what happens when the actor interacts with the system. The interactions are performed in a sequential order. Therefore, a use case always starts with an actor interaction, followed by other interactions from the actor or reactions from the system. The use cases used throughout this section are goal-driven, meaning that a use case describes the actions to take (by the actor and the system) to fulfil a desired goal of the actor.

In the EODiSP context, there are three kinds of actors:

- The simulation owner (the person who wishes to use the EODiSP to perform a simulation).
- The model owner (the person who wishes to use the EODiSP to make one or more of his models available to a simulation manager)
- The person who wishes to wrap a simulation package to transform it into an EODiSP-compatible simulation model (this will often be the same as the model owner).

Note that a use case does not describe the 'look and feel' of the graphical user interface. This is an implementation issue and shall be independent of what the system should provide. A use case describes interactions on the level of services. This makes them independent from the implementation. It also makes use cases reusable because they shall apply to every implementation of the system, regardless of how the graphical user interface looks like.

The format of the uses cases in this document is the one which is proposed by Craig Larman [Lar02]. It is closely related to the format proposed by Alistair Cockburn [Coc00], who also

provides good online resources (including an excerpt of his book) about writing use cases [Coc].

Since it is much easier to read short use cases which have clearly defined boundaries than to read one big use case describing every aspect of an application, the dedicated use cases for each application have been split into smaller, more readable use cases. The sum of all use cases belonging to one application makes up the description of interactions between an actor and the system.

Every use case description throughout this document has 3 subsections. The first one is the *Definitions* section. It includes a table defining the attributes of the use case, such as name, level, description, etc. This information is connected with the use case and shall make its intention clear. A detailed description of every attribute can be found in table 5.1.1.

The second subsection of a use case is the *Main Success Scenario*. It describes the interactions between the primary actor and the system to achieve the specified goal. The Main Success Scenario includes neither system failures, nor exceptions, nor alternative interactions which may be supported by the system. Therefore, it describes the interactions performed by most actors in most cases. If the main success scenario is finished, the primary actor's intended goal has been achieved. The description is given in text format. The sequence of interactions is taken into account by using a numbered list. This sequence must be obeyed in most cases in order to achieve the goal.

The third section is the *Alternative Flows*. It describes what can happen in addition to the main success scenario. This can be any system exception or alternative steps the primary actor can choose from. A step in the alternative flow can either occur at any time or at a certain step in the main success scenario. To highlight this, steps in an alternative flow are either marked with a '*' (asterisk) or a number. Steps which can occur at any time are marked with an asterisk. Steps which have a corresponding step in the main success scenario are marked with the same number as the one in the main success scenario plus a letter indicating different alternatives.

Some of the use cases might be very similar or even equal for different applications. This is mostly the case for use cases with level 'Subfunction'. Instead of having these use cases in one place, they are repeated for every application. This approach can be rather verbose but is considered more convenient for the reader.

Table 5.1.1: Description of attributes used in the use cases.

Attribute	***Description***
Number	A number uniquely identifying the use case with the format *UC_<Application><Nr>*. *UC_* is static for every use case. It is used in text to identify a reference as being a use case. The *<Application>* part is a two digit number identifying the type of application for which the use case is written. This can be one of the following:

Attribute	*Description*
	• *1*: Identifying the type of application as simulation manager application • *2*: Identifying the type of application as model manager application • *3*: Identifying the type of application as support application. The <*Nr*> part of the number is a two digit number. This number takes the sequence of the use case into account by advancing it to a higher number. It can leave spaces between numbers to provide a 'slot' for future use cases.
Name	Every use case shall have a name which is given here. The name should be descriptive and, if possible, unique.
Primary Actor	The primary actor is one having a goal requiring the assistance of the system. The use case describes how the primary actor can achieve this goal.
Level	Every use case is assigned a level. This level describes the intention of the use case. 3 levels are defined. • Summary: Use cases with this level represent collections of user goal level use cases. They describe how several use cases can be assembled together and in which order they can be executed. • User Goal: Use cases with this level are of greatest interest. They describe a goal which the actor can achieve by interacting with the system. • Subfunction: Use cases with this level are steps in a user goal use case. This use cases are described as such because they are used in different user goal use cases or because they are too complex to be directly integrated in them.
Description	A high level description of the goal which shall be achieved by the use case. It is a summary of what can be found in more detail in the text of the use case.
Pre-Condition	Describes the states in which the system has to be prior to the start of the use case.
Post-Condition	Describes the state in which the system is after main success scenario of the use case has been run.

5.1.1 Traceability to Code

Use cases can be seen as requirements telling the system what services it shall provide to an actor. In order to keep track of these requirements formulated by use cases, the unique number of each use case is used in the EODiSP code to make a reference to them. If all use cases are referenced from within the code, it is ensured that all use case requirements are implemented.

Another effect is, that if a use case needs to be modified, the impact on the code is easy to assess.

A reference from use cases to the corresponding implementation code is not given because this would be hard to maintain. Furthermore, use cases should be independent of an actual implementation. Referencing code in a use case would break this important feature.

S5.1.1 -1	*The EODiSP GUI code shall contain references to the use cases that is implemented by the code.*

5.2 Simulation Manager Application Use Cases

This section presents the use cases that define the simulation manager application. The simulation manager application is one of the applications that implement the EODiSP GUI. Each use case is defined in a dedicated subsection.

5.2.1 Use Case – Simulation Manager Application Summary

Definitions

Number	UC_100
Name	Simulation Manager Application Summary
Primary Actor	Simulation owner
Level	Summary
Description	Describes how use cases belonging to the simulation manager application are linked together and the sequence in which they shall be used. The description is from the point of view of the actor.
Pre-Condition	None.
Post-Condition	None.

Main Success Scenario

1. Actor sets up the simulation manager application (see use case *UC_102*).
2. Actor configures a simulation experiment (see use case *UC_104*).
3. Actor runs a simulation experiment (see use case *UC_106*).

Alternative Flows

None.

5.2.2 Use Case – Set-up Simulation Manager Application

Definitions

Number	UC_102
Name	Set-up Simulation Manager Application
Primary Actor	Simulation owner
Level	User goal
Description	Describes the steps required to get a running, fully functional simulation manager application. After this, simulation experiments can be configured and run.
Pre-Condition	• The simulation manager application must be installed on the system.
Post-Condition	• The simulation manager application runs. • The EODiSP network infrastructure is initialised. • The simulation manager application is ready to be configured.

Main Success Scenario

1. Actor starts the simulation manager application. Depending on the platform, this can be achieved by using a command on a console or by selecting the appropriate entry in the start menu.
2. Actor chooses the application setting and the network setting to use from a list of already existing application settings and network settings respectively.
3. System loads the 'ApplicationSettings' configuration file for the simulation manager application from a predefined path in the file system.
4. System initialises the simulation manager application with the parameters given in the 'ApplicationSettings' configuration file.
5. System shows the GUI of the simulation manager application to the actor.
6. Actor chooses to initialise the EODiSP network.
7. System initialises the network infrastructure.
8. System displays to the actor whether the simulation manager application has already been registered in the global simulation manager repository.
9. Actor chooses to register, deregister, or update the simulation manager application in the global simulation manager repository.

Alternative Flows

a*. At any time, system detects an internal, unrecoverable error.
1. System signals error.
2. System exits.
3. Actor restarts the simulation manager application.

5-7a. Actor adjusts the general settings.
1. Actor chooses to save the settings.
2. System stores the settings in the 'ApplicationSettings' configuration file.
2a. System detects that it cannot save the file.
1. System signals error.
2. Actor tries to save again or ignores the operation.
3. System displays updated settings.

5-7a Actor adjust the network settings.
1. Actor chooses to save the settings.
2. System stores the settings in the configuration files.
2a. System detects that it cannot save the file.
1. System signals error.
2. Actor tries to save again or ignores the operation.

5-7b. Actor adjusts simulation log settings. See use case *UC_112*.

7*. At any time during this step, system detects a network error during initialisation.
1. System signals error.
2. Actor tries to resolve the network error.
3. Actor chooses to initialise the network again.

7a. The system runs in remote mode (i.e. Internet connection is available):
1. System connects to the global simulation manager repository .

7b. The system runs in local mode (i.e. no Internet connection is available):
1. System does not attempt to connect to a remote host.
2. Use case ends.

9a. Actor chooses not to register the simulation manager application in the global simulation manager repository.
1. System is running and fully functional.

9b. No network connection to the repository available.
1. System displays error and aborts operation.

9c. Register simulation manager application:
1. System connects to the EODiSP simulation manager repository.
1a. Simulation manager application is already registered in the global repository.
1. System displays information.

2. System aborts operation.

2. System transfers information about the simulation manager application to the repository to uniquely identify it.

9d. Deregister simulation manager application:

1. System connects to the EODiSP simulation manager repository.

1a. Simulation manager application is not registered in the global repository.

1. System displays information.

2. System aborts operation.

2. System instructs global repository to deregister the simulation manager application.

9e. Update simulation manager application.

1. System connects to the EODiSP simulation manager repository.

1a. Simulation manager application is not registered in the global repository.

1. System instructs global repository to register the simulation manager application.

1b. System instructs global repository to update registration for the simulation manager application.

5.2.3 Use Case – Configure Simulation Experiment

Definitions

Number	UC_ 104
Name	Configure Simulation Experiment
Primary Actor	Simulation owner
Level	User goal
Description	Describes the configuration which has to be made prior to starting a simulation experiment.
Pre-Condition	• Use case *UC_102* (Set-up simulation manager application) has been successfully completed.
Post-Condition	• A simulation experiment is configured and ready to be run.

Main Success Scenario

1. Actor chooses to find available federates.
2. System presents all federates which are currently available in the EODiSP network. It only shows those federates which have been made available for this simulation manager application.

3. Actor integrates as many federates as he wishes (i.e. as needed for a specific federation execution) from the list of available simulation models into an already existing federation execution.
4. System retrieves information about federates which are to be integrated, including initialisation data template files or a specification of its format, if available.
5. System displays an updated list of federates included in the federation.
6. Actor integrates as many federations as he wishes from the list of available federations into a simulation experiment. These federations are now called federation executions.
7. System locally copies information about a federate into the simulation experiment, including initialisation data template files, if such data is available.
8. System displays an updated list of federation executions included in a simulation experiment.
9. System highlights federates which expect input data.
10. Actor enters initialisation data for all federates expecting input data.

Alternative Flows

a*. At any time, system detects an internal, unrecoverable error.

1. System tries to save unsaved data.
2. System signals error
3. Actor restarts the simulation manager application.
4. System enters a clean state.

b*. At any time, actor chooses to update the list of available federates.

1. System presents an updated list of available federates.

c*. At any time, actor chooses save the current configuration to the file system (see *UC_108*).

1a. Actor chooses to load an already existing 'SimulationManagerProject' file (see *UC_108*).

1. Go to step 8.

1b. Actor chooses to include a federate manually.

1. System presents a file dialogue.
2. Actor chooses an appropriate file from the file system.

2a. The chosen file does not represent a federate or the system encountered errors in the file.

1. System displays error.

3. System adds this federate to the list of available federates.

3a. The federation execution to which the actor wants to add a federate does not yet exist.

1. Actor adds a new federation execution to the list of federation executions and gives it a dedicated name.

2. System displays updated list of federation executions.

4a. The newly integrated federate expects input data.

1. System marks the federate as special federate which expects input data.

6a. The simulation experiment to which the actor wants to add a federation execution does not yet exist.

1. Actor adds a new simulation experiment to the list of simulation experiments and gives it a unique name.

2. System displays updated list of simulation experiments.

10b. No participating federate can be found in the federation execution which is able to publish the attributes needed by a federate which has been integrated.

1. System marks the federate as not fully integrated.

2. System displays an indication that the simulation experiment might not work properly.

1. Actor adds a federate to the federation which is able to publish the needed attributes.

2. System updates the federation execution in the appropriate simulation experiments.

2a. Newly integrated federate is able to publish the needed attributes.

1. System marks the federate as fully integrated (removes visual indication of not fully integrated).

2. System removes indication about the simulation experiment not working properly.

2b. Newly integrated federate is not able to publish the needed attributes.

5.2.4 Use Case – Run Experiment

Definitions

Number	UC_ 106
Name	Run Experiment
Primary Actor	Simulation owner
Level	User goal
Description	Starts a simulation experiment which has been configured. The configuration of a simulation experiment is explained in use case *UC_104.*
Pre-Condition	• Use case *UC_104* (configure simulation experiment) has been successfully completed for the simulation experiment which is selected to run. • No other experiment has been started from the simulation manager application. • All participating models in the simulation experiment are available in

	the EODiSP network. • If no network connection is configured, all federates must be available locally.
Post-Condition	• The simulation experiment has been completed successfully.

Main Success Scenario

1. Actor selects the simulation experiments which he wants to run.
2. Actor chooses to check the availability of the participating federates.
3. Actor chooses to start the simulation experiment.
4. System initialises simulation experiment.
5. System runs simulation experiment.
6. System detects that the simulation experiments has finished executing.
7. System signals end of simulation experiment to the actor.

Alternative Flows

2a. Not all participating federates are currently available in the EODiSP network.

1. Actor checks the availability of federates periodically until all federates are available in the EODiSP network.

4-5a. System detects an internal error during execution of a simulation experiment.

1. System signals error.
2. System tries to recover the error.

2a. System cannot recover error.

1. Actor aborts the executing simulation experiment (see use case *UC_110*).
2. Actor starts the simulation experiment again.

3. System continues to execute the simulation experiment.

4-5b. Actor chooses to abort the currently running simulation experiment (see use case *UC_110*).

4-5c. At any time in continuous operation mode, actor switches to step-by-step operation mode.

1. System waits until the currently running federate has finished its execution.
2. System holds the execution of the simulation experiment before the next step in the federation execution starts.
3. Actor steps forward for one step.
4. System starts the execution of the next federate.

4-5d. At any time in step-by-step operation mode, actor switches to continuous operation mode.

1a. System is currently not on hold.

1b. System is currently on hold waiting for the actor to step forward.

1. System switches mode to continuous operation mode.

2. System automatically advances the simulation experiment without user interaction.

4-5e. System detects a network error.

1a. Network error occurred in the simulation manager application.

1a. The simulation manager can reconnect to the EODiSP network during a certain amount of time.

1b. The simulation manager cannot reconnect to the EODiSP network during a certain time.

1. System signals error.

2. Actor acknowledges error.

3. System stops the simulation experiment and enters a clean state.

4. System is ready to execute another simulation experiment, whenever the network problem has been fixed.

1b. Network error occurred in one of the participating simulation models.

1. System signals error.

2. System informs all model manager applications involved in the simulation experiment about the network error.

3. System tells the model manager applications to reinitialise all federates which participate in the simulation experiment.

4. System stops the simulation experiment and enters a clean state.

5. System is ready to execute another simulation experiment.

5.2.5 Use Case – Load/Save Configuration

Definitions

Number	UC_ 108
Name	Load or Save Configuration
Primary Actor	Simulation owner
Level	Subfunction
Description	Loads an existing 'SimulationManagerProject' configuration file into the simulation manager application. This can be used to load a configuration which has been done in another simulation manager application. When loading such a configuration, there is no guarantee that the simulation models are available for the simulation manager application in which the configuration is loaded.
Pre-Condition	• The simulation manager application is running.
Post-Condition	• The configuration file has been loaded into the simulation manager ap-

	plication. • The configuration file has been saved to a specific location. • No simulation experiment is currently active.

Main Success Scenario

1. Actor opens a configuration file through an 'open file dialogue'.
2. System processes the new configuration file.
3. System clears the currently active project configuration in the application.
4. System adds all federations and simulation experiments which are configured in the document to the current simulation manager application.
5. Actor chooses to save the configuration file.
6. System asks for a location in which the file should be stored.
7. Actor chooses a location and a name for the file.
8. System saves the files to the given location and the given name.

Alternative Flows

2a. The configuration file does not conform to the XML Schema.
   1. System signals error.
   2. Actor chooses another configuration file to load.

8a. System detects that it cannot save the file.
   1. System signals error.
   2. Actor tries to save again or ignores operation.

5.2.6 Use Case – Experiment Abort

Definitions

Number	UC_110
Name	Experiment Abort
Primary Actor	Simulation owner
Level	Subfunction
Description	Aborts a running simulation experiment before it is finished. There is no recovery, once a simulation experiment has been aborted. All simulation data which are not stored are lost.
Pre-Condition	• A simulation experiment which is controlled by this simulation manager application is currently running.

Post-Condition	• Simulation manager is in a clean state and ready to start a simulation experiment.

Main Success Scenario

1. Actor chooses to abort the running simulation experiment.
2. System sends a message to all participating model manager applications asking them to deregister all federates from the simulation experiment and to enter a clean state.
3. System waits until all federates have resigned from the federation.
4. System reinitialises itself and enters a clean state.

Alternative Flows

3a. Not all federates have resigned from the federation after waiting a configurable amount of time.

1. System signals error to actor.
2. Actor acknowledges the error.
3. System reinitializes itself and enters a clean state.

5.2.7 Use Case – Configure Simulation Manager Application Log

Definitions

Number	UC_112
Name	Configure Simulation Manager Application Log
Primary Actor	Simulation owner
Level	Subfunction
Description	The actor defines which messages shall be logged in the application.
Pre-Condition	• Use case *UC_102* (Set-up simulation manager application) has been successfully completed.
Post-Condition	• Log settings are adjusted.

Main Success Scenario

1. Actor chooses to adjust log settings.
2. System displays options to adjust log settings.
3. System displays a list of options which can be enabled or disabled.
4. Actor chooses which log options he wants to enable or disable.
5. System updates the 'ApplicationSettings' configuration file of the simulation manager application..

Alternative Flows

None.

5.3 Model Manager Application Use Cases

This section presents the use cases that define the model manager application. The model manager application is one of the application that implement the EODiSP GUI. It is normally operated by a model owner. Each use case is defined in a dedicated subsection.

5.3.1 Use Case – Model Manager Application Summary

Definitions

Number	UC_200
Name	Model Manager Application Summary
Primary Actor	Model owner
Level	Summary
Description	Describes how the use cases belonging to the model manager application are linked together and in which order they can be used. The description is from the point of view of the actor.
Pre-Condition	None.
Post-Condition	None.

Main Success Scenario

1. Actor sets up the model manager application (see use case *UC_202*)
2. Actor adds federates to the model manager application and configures them (see use case *UC_204*).

Alternative Flows

None.

5.3.2 Use Case – Set-up Model Manager Application

Definitions

Number	UC_ 202
Name	Set-up Model Manager Application
Primary Actor	Model owner
Level	User goal
Description	Describes the steps in order to get a running, fully functional model man-

	ager application.
Pre-Condition	• The model manager application must be installed on the system.
Post-Condition	• The model manager application runs. • The EODiSP network infrastructure is initialised. • Federates can be added to the model manager application.

Main Success Scenario

1. Actor starts the model manager application. Depending on the platform, this can be achieved by using a command in a console or by selecting the appropriate entry in the start menu.
2. Actor chooses the application setting and the network setting to use from a list of already existing application settings and network settings respectively.
3. System loads the 'ApplicationSettings' configuration file for the model manager application from a predefined path in the file system.
4. System initialises the model manager application with the parameters given in the configuration file.
5. System shows the GUI of the model manager application to the actor.
6. Actor chooses to initialise the EODiSP network.
7. System initialises the network infrastructure.
8. System makes already configured federates available to the EODiSP network.

Alternative Flows

a*. At any time, system detects an internal, unrecoverable error.
   1. System signals error.
   2. System exits.
   3. Actor restarts the model manager application.

5-7a. Actor adjusts the general settings.
   1. Actor chooses to save the settings.
   2. System stores the settings in the 'ModelManagerProject' configuration file.
      2a. System detects that it cannot save the file.
         1. System signals error.
         2. Actor tries to save again or ignores the operation.
   3. System displays updated settings.

5-7a Actor adjust the network settings.
   1. Actor chooses to save the settings.

2. System stores the settings in the configuration files.
   2a. System detects that it cannot save the file.
      1. System signals error.
      2. Actor tries to save again or ignores the operation.

5-7b. Actor adjusts simulation log settings. See use case *UC_208*.

7-8a. System detects a network error during initialisation.
1. System signals error.
2. Actor tries to resolve the network error.
3. Actor chooses to initialise the network again.

5.3.3 Use Case – Manage Federates in the Model Manager Application

Definitions

Number	UC_204
Name	Manage Federates in the Model Manager Application
Primary Actor	Model owner
Level	User goal
Description	When a model manager application runs it is ready to integrate additional federates and to make them available to certain simulation manager applications. This use case describes the interactions for this configuration.
Pre-Condition	• Use case *UC_202* (set-up model manager application) has been successfully completed.
Post-Condition	• Some (or all) federates included in the model manager application are ready to be integrated into a simulation experiment.

Main Success Scenario

1. Actor chooses to add a federate.
2. System displays a dialogue which lets the actor choose the federate he wants to integrate.
3. Actor chooses a 'SOM' configuration file which belongs to the federate he wants to integrate.
4. System processes the SOM configuration file.
5. System adds the selected federate to the list of managed federates in the model manager application.
6. Actor selects the newly added federate from the list of managed federates.
7. Actor chooses to configure the security settings for the selected federate.
8. System displays options to specify security settings.

9. Actor selects from a list of registered simulation manager applications to which the federate shall be made available.
10. System automatically updates the 'SOMSecuritySettings' configuration file.
11. Actor chooses to make the federate available to the EODiSP network.
12. System internally adds the federate to the list of available federates for the configured simulation manager application(s) in order to respond to requests coming from them.
13. System displays whether the federate has already been registered in the global federate repository.
14. Actor chooses to register, deregister, or update a federate in the repository holding a list of available federates.

Alternative Flow

3a. Additional configuration files are needed for this type of federate.

1. Actor chooses the additional configuration files to be integrated.
2. System processes additional configuration files.

4a. The configuration file does not conform to the XML Schema.

1. System signals error.
2. Actor chooses another configuration file to load.

6a. Actor selects an already existing federate from the list of managed federates.

9a. No simulation manager application is registered in the EODiSP network.

1. The actor waits until at least one simulation manager application has been registered in the EODiSP network.

9b. No network connection is available to fetch the list of registered simulation manager applications.

1. System displays error.
2. Actor tries to resolve the network error or ignores operation.

9c. Actor chooses to make the federate available to all simulation manager applications in the EODiSP network.

9d. Actor chooses to manually specify a simulation manager application.

1. System presents an open file dialogue.
2. Actor chooses the file which uniquely identifies a simulation manager application. This file can be created by the simulation manager application itself.

2a. System cannot processes the file.

1. System displays error.
2. Actor tries to load the file again or ignores the operation.

12a. System detects a network error.

1a. Network error occurred in the model manager itself.

1a. The model manager can reconnect to the EODiSP network during a certain amount of time.

1b. The model manager cannot reconnect to the EODiSP network during a certain time.

1. System signals error.
2. Actor acknowledges error.
3. System reinitialises the federate.
4. Federate is in a clean state to be used by another simulation experiment.

1b. Network error occurred in one of the other participating simulation models.

1. System reports error in the log file.

2a. System does not receive a message from the simulation manager application informing it to shut down.

2b. System receives a message from the simulation manager application informing it to shut down.

1. System signals error.
2. Actor acknowledges error.
3. System reinitialises the federate.
4. Federate is in a clean state to be used by another simulation experiment.

1c. Network error occurred in the simulation manager application.

1a. The model manager can reconnect to the simulation manager application during a certain amount of time.

1b. The model manager cannot reconnect to simulation manager application during a certain time.

1. System signals error.
2. Actor acknowledges error.
3. System reinitialises the federate.
4. Federate is in a clean state to be used by another simulation experiment.

14a. Actor chooses not to register the federate in the global federate repository.

14b. No network connection to the repository available.

1. System displays error and aborts operation.

14c. Register federate:

1. System connects to the EODiSP federate repository.

1a. Federate is already registered in the global repository.

1. System displays information.
2. System aborts operation.

2. System transfers information about the federate to the repository to uniquely identify it.

14d. Deregister federate:

1. System connects to the EODiSP federate repository.

1a. Federate is not registered in the global repository.

1. System displays information.
2. System aborts operation.

2. System instructs global repository to deregister federate.

14e. Update federate:

1. System connects to the EODiSP federate repository.

1a. System is not registered in the global repository.

1. System instructs global repository to register federate.

1b. System instructs global repository to update the registration of the federate.

5.3.4 Use Case – Configure Model Manager Application Log

Definitions

Number	UC_208
Name	Configure Model Manager Application Log
Primary Actor	Model owner
Level	Subfunction
Description	The actor can configure which messages shall be logged in the application.
Pre-Condition	• Use case *UC_202* (set-up model manager application) has been successfully completed.
Post-Condition	• Log settings are adjusted.

Main Success Scenario

1. Actor chooses to adjust log settings.
2. System displays options to adjust log settings.
3. System displays a list of options which can be enabled or disabled.
4. Actor chooses which log options he wants to enable or disable.
5. System updates the 'ApplicationSettings' configuration file of the model manager application..

Alternative Flow

None.

5.4 Support Application Use Cases

This section presents the use cases that define the support application. The support applications are a set of applications that are provided by the EODiSP to help construct the wrappers that transform the simulation packages into HLA federates.

5.4.1 Use Case – Generate Wrapper Code for an Excel Workbook

Definitions

Number	UC_302
Name	Generate Wrapper Code for an Excel Workbook
Primary Actor	Model owner
Level	User Goal
Description	Describes the steps to generate the code that wraps an existing Excel Workbook as an HLA federate that can be integrated into the EODiSP.
Pre-Condition	• An Excel Workbook is available. • Microsoft Excel is installed on the the model owners computer. • The EODiSP Wrapper macro is installed in Microsoft Excel. • The support application is running.
Post-Condition	• The code that implements a federate wrapper for a given Excel Workbook (federate interface implementation and Excel connector code). • A SOM file that describes the publishing/subscribing facilities of the federate.

Main Success Scenario

1. Actor starts Microsoft Excel and opens the Excel Workbook for which he wants to generate the wrapper code.
2. Actor selects one or more cell ranges as input or output values.
3. System (EODiSP Wrapper macro) highlights input and output ranges.
4. User chooses to generate an *ExcelMapping* file skeleton.
5. System generates an Excel mapping file skeleton. The skeleton only contains the Excel part of the mapping (one Excel element for each input and output range).
6. Actor chooses to load the generated mapping file into the support application.
7. System (support application) displays the mapping file.
8. Actor fills in the missing parts. Namely the mapping of the chosen input and output ranges to HLA object class attributes. The changes can be done either in an external XML editor or directly inside the support application using a text editor.

9. Actor adds mappings from HLA attributes to Excel macros.
10. Actor chooses to generate the SOM file from the mapping file.
11. System generates the SOM file.
12. Actor chooses to generate the complete wrapper code.
13. System generates the federate interface wrapper code from the SOM and the connector code using the information in the mapping file.

Alternative Flows

5a. No input or output ranges selected.
1. System signals error.
2. Go to step 2.

11a. Mapping file does not validate against the *ExcelMapping* XML Schema.
1. System signals error.
2. Actor corrects the mapping file and chooses to regenerate the SOM file.

13a. SOM file or the *ExcelMapping* file do not validates against their XML Schemas.
1. System signals error.
2. Actor corrects the mapping file and regenerates the SOM file (step 9) if necessary.

5.4.2 Use Case – Generate Wrapper Code for an SMP2 simulation

Definitions

Number	UC_304
Name	Generate Wrapper Code for an SMP2 simulation
Primary Actor	Model owner
Level	User Goal
Description	Describes the steps to generate the code that wraps an existing SMP2 simulation as a federate that can be integrated into the EODiSP.
Pre-Condition	• An SMP2 catalogue file is available. • An SMP2 assembly file is available. • The support application is running.
Post-Condition	• The code that implements a federate wrapper for a given SMP2 simulation (federate interface implementation and SMP2 connector code). • A SOM file that describes the publishing/subscribing facilities of the federate.

Main Success Scenario

1. Actor chooses to load an SMP2 catalogue file.
2. Actor chooses to load an SMP2 assembly file.
3. System displays model instances and their features (properties, fields etc.) in a tree.
4. Actor selects one or more features in the tree and chooses to generate an *SMP2Mapping* file skeleton.
5. System generates an SMP2Mapping file skeleton. The skeleton only contains the SMP2 part of the mapping (one feature element for each selected feature).
6. System displays the mapping file and highlights incomplete parts.
7. Actor fills in the missing parts. Namely the mapping of the chosen features to HLA object class attributes. The changes can be done either in an external XML editor or directly inside the support application using a text editor.
8. Actor chooses to generate the SOM file from the mapping file.
9. System generates the SOM file.
10. Actor chooses to generate the wrapper code.
11. System generates the federate interface wrapper code from the SOM and the connector code using the information in the mapping file.

Alternative Flows

3a. Catalogue or Assembly files do not validate against XML Schema.
   1. System signals error.
   2. Actor chooses to another Assembly or Catalogue file.

5a. No features chosen.
   1. System signals error.
   2. Go to step 4.

9a. Mapping file does not validate against the *SMP2Mapping* XML Schema.
   1. System signals error.
   2. Actor corrects the mapping file and chooses to regenerate the SOM file.

11a. SOM file or the *SMP2Mapping* file do not validates against their XML Schemas.
   1. System signals error.
   2. Actor corrects the mapping file and regenerates the SOM file (step 8) if necessary.

5.4.3 Use Case – Generate Wrapper Code for Matlab-Generated Code

Definitions

Number	UC_306

Name	Generate Wrapper Code for Matlab-Generated Code
Primary Actor	Model owner
Level	User Goal
Description	Describes the steps to generate the code that wraps an existing Matlab-generated code as a federate that can be integrated into the EODiSP.
Pre-Condition	• Matlab-generated code is available. • The support application is running.
Post-Condition	• The code that implements a federate wrapper for a given Matlab-generated code (federate interface implementation and connector code). • A SOM file that describes the publishing/subscribing facilities of the federate.

Main Success Scenario

1. Actor wraps the Matlab-generated code as an SMP2 model using the Mosaic tool and integrates it into SimSat 2000.
2. Actor generates wrapper code for this SMP2 simulation (see use case *UC_304*).

Alternative Flows

None.

5.4.4 Use Case – Create Wrapper Code for a Matlab Simulation

Definitions

Number	UC_308
Name	Generate Wrapper Code for a Matlab Simulation
Primary Actor	Model owner
Level	User Goal
Description	Describes the steps to create the code that wraps an existing Matlab simulation as a federate that can be integrated into the EODiSP.
Pre-Condition	• A Matlab simulation is available. • The support application is running.
Post-Condition	• The code that implements a federate wrapper for a given Matlabsimulation (federate interface implementation and connector code). • A SOM file that describes the publishing/subscribing facilities of the federate.

Main Success Scenario

1. Actor creates a new SOM file (see use case *UC_316*).
2. Actor chooses to generate the federate interface implementation code from the SOM.
3. System generates the federate interface implementation code.
4. Actor implements the connector code that connects the federate interface implementation code with the Matlab simulation using the COM interface of the Matlab simulation. COM calls are implemented using the SWT COM/Java bridge [Swt].

Alternative Flows

5a. SOM file does not validates against its XML Schema.

1. System signals error.

2. Actor corrects the SOM file and chooses to regenerate the federate interface implementation (step 4).

5.4.5 Use Case – Create Wrapper for Source Code

Definitions

Number	UC_310
Name	Generate Wrapper Code for Source Code
Primary Actor	Model owner
Level	User Goal
Description	Describes the steps to create the code that wraps an existing executable as a federate that can be integrated into the EODiSP.
Pre-Condition	• Some source (C/C++/Fortran/Java) code is available that can be compiled to an executable program. • The support application is running.
Post-Condition	• The code that implements a federate wrapper for some given source code (federate interface implementation and connector code). • A SOM file that describes the publishing/subscribing facilities of the federate.

Main Success Scenario

1. Actor creates a new SOM file (see use case *UC_316*).
2. Actor chooses to generate the federate interface implementation code from the SOM.
3. System generates the federate interface implementation code.

4. Actor implements the connector code that connects the federate interface implementation code with the source code.

Alternative Flows

5a. SOM file does not validates against its XML Schema.

1. System signals error.

2. Actor corrects the SOM file and chooses to regenerate the federate interface implementation (step 4).

5.4.6 Use Case – Create Wrapper for a Standalone Executable

Definitions

Number	UC_312
Name	Generate Wrapper Code for a Standalone Executable
Primary Actor	Model owner
Level	User Goal
Description	Describes the steps to create the code that wraps an existing executable with a defined in- and/or output as a federate that can be integrated into the EODiSP.
Pre-Condition	• The input and output of a standalone executable is known. • The support application is running.
Post-Condition	• The code that implements a federate wrapper for the standalone executable (federate interface implementation and connector code). • A SOM file that describes the publishing/subscribing facilities of the federate.

Main Success Scenario

1. Actor creates a new SOM file (see use case *UC_316*).
2. Actor chooses to generate the federate interface implementation code from the SOM.
3. System generates the federate interface implementation code.
4. Actor implements the connector code that connects the federate interface implementation code with the standalone executable.

Alternative Flows

5a. SOM file does not validates against its XML Schema.

1. System signals error.

2. Actor corrects the SOM file and chooses to regenerate the federate interface implementation (step 4).

5.4.7 Use Case – Create Wrapper for a Data Processing Package

Definitions

Number	UC_314
Name	Generate Wrapper Code a Data Processing Package
Primary Actor	Model owner
Level	User Goal
Description	Describes the steps to create the code that wraps an existing data processing package as a federate that can be integrated into the EODiSP.
Pre-Condition	• The interface of a data processing package is known. • The support application is running.
Post-Condition	• The code that implements a federate wrapper for the dta processing package (federate interface implementation and connector code). • A SOM file that describes the publishing/subscribing facilities of the federate.

Main Success Scenario

1. Actor creates a new SOM file (see use case *UC_316*).
2. Actor chooses to generate the federate interface implementation code from the SOM.
3. System generates the federate interface implementation code.
4. Actor implements the connector code that connects the federate interface implementation code with the data processing package.

Alternative Flows

5a. SOM file does not validates against its XML Schema.

1. System signals error.

2. Actor corrects the SOM file and chooses to regenerate the federate interface implementation (step 4).

5.4.8 Use Case – Creating a New SOM File

Definitions

Number	UC_316
Name	Creating a New SOM File

Primary Actor	Model owner
Level	Subfunction
Description	Describes the steps to create a new SOM file from scratch.
Pre-Condition	• The support application is running.
Post-Condition	• A SOM file that defines the desired object classes and their attributes and the ability of the federate for publishing them or subscribing to them.

Main Success Scenario

1. Actor chooses to create a new SOM file.
2. System creates a valid SOM file with no object class definitions and displays the file in a text editor.
3. Actor edits the SOM and adds the desired object class and attribute definitions to the SOM.

Alternative Flows

None.

6 GUI Configuration Files

This section defines the configuration files that are associated to the GUI part of the EODiSP. The EODiSP GUI is implemented over three separate applications: the *simulation manager application*, the *model manager application*, and the *support application*.

The GUI configuration files formally and fully define the type of information that users must provide when they use the EODiSP. They are specified as XML Schema expressed in the XSD language. Since the configuration files are specified in a formal language, their specification is used in the EODiSP implementation to check that users actually enter all the information that is required from them and that the structure of this information complies with its specification. The schema information is also used to automatically generate part of the editors where users enter their inputs.

Subsection 6.1 describes the approach taken in this document to specifying the XML Schemas. Subsections 6.2 and 6.3 discuss the configuration files of the simulation manager and model manager applications. The configuration files for the support applications have a more idiosyncratic character because they depend on the kind of wrappers supported by the EODiSP and are therefore discussed separately in section 9.

6.1 XML Schema Documentation

The GUI configuration files are formally specified through XML Schemas written in the XSD language. The XML Schemas for the EODiSP GUI configuration files are given in their entirety in appendix C. This section presents them using a more informal (but more easily readable) graphical and textual representation.

The representation format is shown in Figure 4. Each box in this figure represents an XML element. Boxes are connected with an arrow line that shows the parent/child relationship between XML element. These connection lines can be annotated by a cardinality.

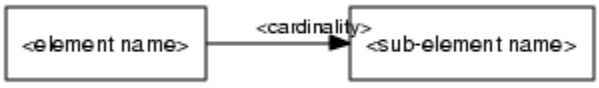

Figure 4: Graphical representation of XML Schemas

Non-essential elements which have no attributes and only serve as container elements are not described in more detail. Other elements are described in dedicated tables. Each table describes one XML element and its attributes. The format of these description tables is as follows:

Name:	<The name of the element>
Description:	<A detailed description of the element>
Parent element:	<Parent element of the described element or 'none' if it is a root element>
Child element(s):	<A comma separated list of child elements. Set to 'none' if this

	element cannot have any child elements>
Attributes:	A list of attributes with the following form: <attribute type> <**attribute name>**

Where appropriate, each attribute is described in detail in the text that follows the table.

The element description tables are automatically constructed by processing the XML Schema themselves. This guarantees that the description given in this section is consistent with the formal XML Schemas given in appendix C.

6.2 Simulation Manager Application Configuration Files

Table 6.2.1 lists all the EODiSP GUI configuration files associated with the *simulation manager application*. For each configuration file a brief description of their role is given. More details can be found in [Urd]. The last column gives an identifier for the XML Schema. The identifier is used in the traceability matrix as a concise way to refer to the XML Schemas.

Note that the list of configuration files given in table 6.2.1 differs from the list of configuration files of the simulation manager application as it is given in the URD. The deviations are discussed in sections 4.1 and 4.3.

The XML Schemas are described in detail in dedicated subsections in this section. The following requirement acts as an umbrella requirement that mandates the applicability of the XML Schemas. However, as discussed in section 3.2, traceability to user requirements is sometimes done in terms of the individual XML Schemas.

S6.2 -1	*The user inputs to be provided through the simulation manager application shall conform to the XML Schemas listed in table 6.2.1.*

Table 6.2.1: Configuration files of the simulation manager application

File Name	***Description***	***ID***
SimulationManager-Project	Stores the configuration tree from an EODiSP simulation manager application. This file can be reused to load a complete configuration at a later time. Note that this file is obtained by merging the SimulationsConfig and ExperimentInitConfig files of the URD.	XS_001
FOM	Defines object classes, attributes, etc. and the information they exchange at runtime.	HLA DTD
ApplicationSettings	Stores general settings configured in the simulation manager application.	XS_010

6.2.1 SimulationManagerProject Configuration File

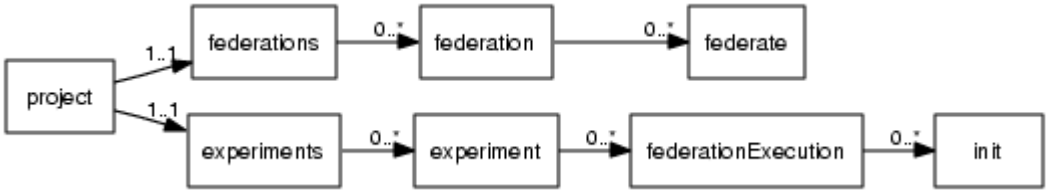

Figure 5: Structure of the SimulationManagerProject XML Schema

A project configuration for the simulation manager application defines two things, federations and experiments. Each federation is assembled from one or more federates. Each experiment consist of one or more federation executions. The interfaces of the federates are defined in SOM files which are located in the model manager application.

Essential XML elements are described in the sections below:

Element <project>

Name:	***project***
Description:	The root element of the SimulationManagerProject file.
Parent element:	`none`
Child element(s):	`federations, experiments`
Attributes:	`xs:string` **name**

Attribute "name"

- *type:* `xs:string`
- *use:* required

The name of the simulation manager project

Element <federation>

Name:	***federation***
Description:	Describes a federation configuration that can be executed in an experiment.
Parent element:	`federations`
Child element(s):	`federate`
Attributes:	`xs:ID` **name**

Attribute "name"

- *type:* `xs:ID`
- *use:* required

The name of the federation configuration. Must be unique.

Element <federate>

Name:	***federate***
Description:	References a federate present in a model manager application.
Parent element:	`federation`
Child element(s):	`none`
Attributes:	`xs:anyURI` **uri**

Attribute "uri"

- *type:* `xs:anyURI`
- *use:* required

References a federate on a model manager application. The exact format of this URI is not defined yet but it shall reference a particular federate located in a model manager application. The authority part of the URI is probaly an identification of the model manager where the path part points to a particular federate (e.g: `mymodelmanager/federate_1`).

A federate is defined by its id (attribute `id` of a federate element in the *ModelManagerProject* file).

(see http://www.ietf.org/rfc/rfc2396.txt for the definition of an URI)

Element <experiment>

Name:	***experiment***
Description:	An experiment combines several federation executions which can be executed in one run.
Parent element:	`experiments`
Child element(s):	`federationExecution`
Attributes:	`xs:string` **name**

Attribute "name"

- *type:* `xs:string`
- *use:* required

The name of the experiment

Element <federationExecution>

Name:	***federationExecution***
Description:	A federation execution is a reference to a federation with the necessary additional information to initialize and execute a federation.

Parent element:	`experiment`
Child element(s):	`init`
Attributes:	`xs:IDREF` **federation**

Attribute "federation"

- *type:* `xs:IDREF`
- *use:* required

References a federation name defined in a federation element.

Element <init>

Name:	***init***
Description:	Each federate can have one or more initialization resources attached. The init element defines the locations of these resources and defines to which federate this resource belongs to. A resource can be a file but also any other resource that is referencable by an URI (e.g. a database).
Parent element:	`federationExecution`
Child element(s):	`none`
Attributes:	`xs:string` **federate** `xs:anyURI` **resource** `xs:string` **type**

Attribute "federate"

- *type:* `xs:string`
- *use:* required

Specifies the federate to which the initialization data is assigned to. The exact format has yet to be decided. It could refercence a federate by using an XPath like expression (i.e. `//federation.0/federate.0` for the first federate appearing in the first federation).

Attribute "resource"

- *type:* `xs:anyURI`
- *use:* required

Specifies a resource that contains the initialization data for this federate. The type of this attribute is a URI. Currently the only supported protocol is `'file:'` (i.e.`file:///C:/initData/init.xml`). Relative paths are allowed and are always relative to the location of the project file (i.e. `file://../init.xml`).

Attribute "type"

- *type:* `xs:string`
- *use:* required

Defines the type of the initialzation data. Possible types will be defined in a later phase.

6.2.2 FOM

The FOM file structure is defined by a DTD. The full DTD is part of the HLA standard and defined in Annex C of [Omt00].

6.2.3 Application Settings

Application settings for the *simulation manager application* are stored in a set of key/value pairs. A standard XML Schema is used for this purpose and is described in section 6.4. Table 6.2.2 lists valid keys and a description of the valid values for this key for the simulation manager application:

Table 6.2.2: A list of keys defined for the Simulation Manager Application Settings

Key	***Description***
RecentProjectFiles	A list of the locations of recently used SimulationManager-Project files.
NetworkImpl	Specifies the network infrastructure implementation. Currently, the only meaningful value is `jxta`. If other network infrastructures become available this key can be used to choose between different network implementation.
NetworkMode	Defines in which network mode the EODiSP shall operate. The choices are 'local' or 'remote'. If in local mode, the EODiSP will make no attempt to make a connection to the Internet.
JxtaConfig	An absolute or relative path to the directory where JXTA configuration files reside or shall be saved respectively. If a relative path is given, it is relative to the path given in the environment variable 'EODISP_SIM_CONFIG'.

One special variable which is mentioned in table 6.2.2 is the environment variable 'EODISP_SIM_CONFIG'. It specifies a path to a directory where the all configuration files for the simulation manager application shall be stored. If the environment variable is omitted (i.e. not specified), the path `{user_home}/.simConfig` will be used as default value. The value {user_home} is the home directory of the user currently logged in.

The directory specified with this environment variable will hold all configuration files related to the simulation manager application, including JXTA configuration files.

6.3 Model Manager Application Configuration Files

Table 6.3.1 lists all the EODiSP GUI configuration files associated with the *model manager application*. For each configuration file a brief description of their role is given. More details can be found in [Urd]. The last column gives an identifier for the XML Schema. The identifier is used in the traceability matrix as a concise way to refer to the XML Schemas.

Note that the list of configuration files given in table 6.3.1 differs from the list of configuration files of the model manager application as it is given in the URD. The deviations are discussed in sections 4.4.

The XML Schemas are described in detail in dedicated subsections in this section. The following requirement acts as an umbrella requirement that mandates the applicability of the XML Schemas. However, as discussed in section 3.2, traceability to user requirements is sometimes done in terms of the individual XML Schemas.

S6.3 -1	*The user inputs to be provided through the model manager application shall conform to the XML Schemas listed in table 6.3.1.*

Table 6.3.1: Configuration files of the model manager application

File Name	***Description***	***ID***
ModelManager-Project	Stores the configuration tree of federates which are included in the model manager application and the path to find them. Note that this file is obtained by merging the *ModelsConfig* and *SomSecurityConfig* files of the URD.	XS_002
SOM	Stores information about object classes and object class attributes of a single federate.	HLA DTD
ApplicationSettings	Stores general settings configured in the model manager application.	XS_011

6.3.1 ModelManagerProject Configuration File

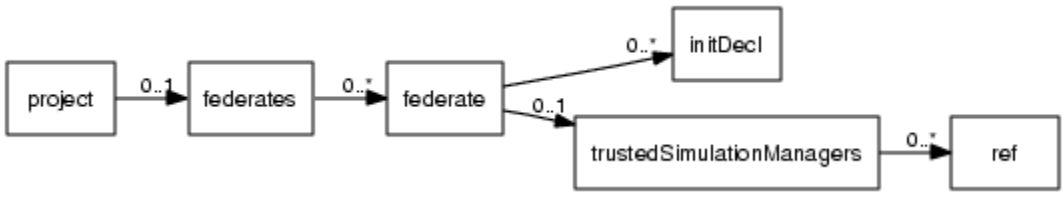

Figure 6: Structure of the ModelManagerProject XML Schema

A model manager manages several federates and possibly makes them available to simulation manager applications. Each federate is formally described in a SOM. The project configuration file specifies which SOM files are loaded in the model manager application and which of

these SOMs are available on the network. Finer grained settings for the publication of federates can be specified in the SOMSecuritySettings file.

Essential XML elements are described in the sections below:

Element <project>

Name:	***project***
Description:	The root element of the ModelManagerConfig file.
Parent element:	`none`
Child element(s):	`federates`
Attributes:	`xs:string` **name**

Attribute "name"

- *type:* `xs:string`
- *use:* required

The name of the model manager project

Element <federate>

Name:	***federate***
Description:	Describes a federate available on the model manager node and controls its publication to the eodisp network.
Parent element:	`federates`
Child element(s):	`initDecl, trustedSimulationManagers`
Attributes:	`xs:ID` **id** `xs:anyURI` **som** `xs:boolean` **public**

Attribute "id"

- *type:* `xs:ID`
- *use:* required

Each federate is identified by a unique id. The format used is a Universal Unique Identifier (UUID). See http://www.ietf.org/rfc/rfc4122.txt for a definition of it.

Attribute "som"

- *type:* `xs:anyURI`
- *use:* required

The URI that identifies a Simulation Object Model (SOM) file. Currently the only supported protocol is `'file:'` (i.e. `file:///C:/soms/mySom.xml`). Relative paths are allowed and are always relative to the location of the project file (i.e. `file://../soms/mySom.xml`).

Attribute "public"

- *type:* `xs:boolean`
- *use:* required

Specifies if this federate should be available to simulation manager applications (`true/false`). Finer grained security settings for a publication of a federate can be set in the SOMSecuritySettings configuration file.

Not that if this attribute is set to true it does not mean that the federate is available to simulation manager applications. It only declares the federate as being published. *If* the federate is accessible by a simulation manager application is defined in the SOMSecuritySettings configuration file.

Element <initDecl>

Name:	***initDecl***
Description:	An initialization declaration specifies the kind of initialization data a federate expects. Additionally it can provide a template resource which will be delivered to the simulation manager to support the simulation owner in creating the initialization data resource.
Parent element:	`federate`
Child element(s):	`none`
Attributes:	`xs:string` **type** `xs:anyURI` **template** `xs:anyURI` **spec** `xs:boolean` **required**

Attribute "type"

- *type:* `xs:string`
- *use:* required

Defines the type of the data expected by the federate. Possible types will be defined in a later phase.

Attribute "template"

- *type:* `xs:anyURI`
- *use:* optional

Specifies a resource that contains a template for the data expected by the federate.Currently the only supported protocol is `'file:'` (i.e. `file://C:/myTemplateFile.xml`). Relative paths are allowed and are always relative to the location of the project file (i.e. `file://../myTemplateFile.xml`).

Attribute "spec"

- *type:* `xs:anyURI`
- *use:* optional

Specifies an optional resource which gives a more detailed description of the initialization data expected by a federate. The kind of resource which is specified in this attribute depends on the type of the initialization. It could for example point to an XML Schema to define the format of initialization data encoded in XML. But it can point to any other resource which specifies the format of the initialization data. The format of the URI is restricted in the same way as in the template attribute.

Attribute "required"

- *type:* `xs:boolean`
- *use:* required

If set to `false` the federate can be started without specifying any initialization data otherwise initialization data has to be provided by the simulation manager.

Element <trustedSimulationManagers>

Name:	***trustedSimulationManagers***
Description:	Lists the simulation manager applications which are trusted by the model manager, that is, the simulation managers that are allowed to make this federate a part of a federation execution and therefore run this federate on the model manager node.
Parent element:	`federate`
Child element(s):	`ref`
Attributes:	`none`

Element <ref>

Name:	***ref***
Description:	References a simulation manager application node.
Parent element:	`trustedSimulationManagers`
Child element(s):	`none`
Attributes:	`xs:anyURI` **uri**

Attribute "uri"

- *type:* `xs:anyURI`
- *use:* optional

References a simulation manager application. The exact format of this URI is not defined yet but it shall reference unambiguously a specific simulation manager application.

(see http://www.ietf.org/rfc/rfc2396.txt for the definition of an URI)

6.3.2 SOM

The SOM file structure is defined by a DTD. The full DTD is part of the HLA standard and defined in Annex C of [Omt00].

6.3.3 Application Settings

Application settings for the model *manager application* are stored in a set of key/value pairs. A standard XML Schema is used for this purpose and is described in section 6.4. Table 6.3.2 lists valid keys and a description to the valid values for this key for the simulation manager application:

Table 6.3.2: A list of keys defined for the Simulation Manager Application Settings

Key	***Description***
RecentProjectFiles	A list of the locations of recently used ModelManagerProject files.
NetworkImpl	Specifies the network infrastructure implementation. Currently, the only meaningful value is `jxta`. If other network infrastructures become available this key can be used to choose between different network implementation.
NetworkMode	Defines in which network mode the EODiSP shall operate. The choices are 'local' or 'remote'. If in local mode, the EODiSP will make no attempt to make a connection to the Internet.
JxtaConfig	An absolute or relative path to the directory where JXTA configuration files reside or shall be saved respectively. If a relative path is given, it is relative to the path given in the environment variable 'EODISP_MOD_CONFIG'.

One special variable which is mentioned in table 6.3.2 is the environment variable 'EODISP_MOD_CONFIG'. It specifies a path to a directory where the all configuration files for the model manager application shall be stored. If the environment variable is omitted (i.e. not specified), the path `{user_home}/.modConfig` will be used as default value. The value {user_home} is the home directory of the user currently logged in.

The directory specified with this environment variable will hold all configuration files related to the model manager application, including JXTA configuration files.

6.4 General format of Application Settings

All applications defined for the EODiSP use the standard Java mechanism to store general application settings. This mechanism is provided by the Java `Properties` class and allows to save settings a set of of key/value pairs. The DTD of the XML file to which the data is stored to is defined by the `Properties` class. Figure 7 shows the structure of this DTD.

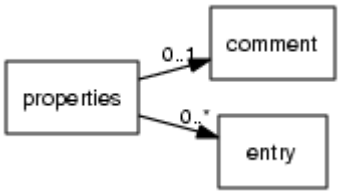

Figure 7: Structure of the DTD used by Java's Properties class

The entry element defines an attribute `key`. Its content type is PCDATA which means it can have any text content. For convenience and because the DTD is very short it is printed here in full:

```
<?xml version="1.0" encoding="UTF-8"?>
    <!ELEMENT properties ( comment?, entry* ) >
    <!ATTLIST properties version CDATA #FIXED "1.0">
    <!ELEMENT comment (#PCDATA) >
    <!ELEMENT entry (#PCDATA) >
    <!ATTLIST entry key CDATA #REQUIRED>
```

7 HLA Core

This section defines the requirements that are applicable to the HLA Core part of the EODiSP. The requirements are stated in terms of the HLA services that are supported by the EODiSP. These services are defined in the EODiSP URD. The services themselves are defined in the HLA standard using a logical model based on state machines.

This section specifies which of the state machines defined by the HLA standard are implemented in the EODiSP. Since the standard includes an extensive and accurate description for each state machine, no further description of these state machines is given in this document. However, references to the appropriate sections and descriptions in the HLA standard are given.

Section 7.1 gives an overview of the state machine concept and it presents the modelling approach used in the EODiSP. Section 7.2 identifies the HLA state machines that are supported by the EODiSP and section 7.3 presents the model for selected HLA state machines supported by the EODiSP. Note that section 7.3 only give an overview of the state machine models. The full definition can be found in appendix B.

7.1 State Machines

A state machine models behaviour composed of states, transitions and actions. A state stores information about the past, i.e. it reflects the input changes from the system start to the present moment. A transition indicates a state change and is described by a condition that would need to be fulfilled to enable the transition. An action is a description of an activity that is to be performed at a given moment. Several action types can be performed

- Entry action: Execute the action when entering a state.
- Exit action: Execute the action when exiting a state.
- Input action: execute the action dependant on present state and input conditions.
- Transition action: execute the action when performing a transition.

The HLA standard describes state machines in terms of state diagrams. State diagrams are graphical representations of (finite) state machines. The most common form is a directed graph, which is also used by the HLA.

In this document, for the more complex HLA state machine, a more formal description is used. This is based on the concurrent hierarchical state machine [Chsm] modelling approach. This approach uses its own language to represent a state machine. This language is a further abstraction of a visual representation of a state machine. CHSM provides a compiler to use with this language. The compiler output is Java code representing a state machine. This code can be integrated into the EODiSP project. The use of CHSM makes it therefore possible to transfer the visual representation in form of state diagrams in the HLA standard over a specialised language defined by CHSM into Java code.

Section 7.3 includes the documentation part of each CHSM file which has been created for the EODiSP project. Every CHSM file represents one state machine defined in the HLA

standard. The documentation part of a CHSM file is only a description of what HLA services are covered by this state machine as well as a reference to the appropriate section in the HLA standard document. The whole CHSM definitions are included in the Appendix B of this document.

This section discusses the CHSM language in which state machines are defined. It shall give enough information to understand the listings in Appendix B where the whole CHSM representation of the state charts in the EODiSP projects are included. However, in case of interest, more information about the CHSM language can be found on its website [Chsm].

Every CHSM definition file includes the definition for one state machine. Since in the EODiSP project, multiple state machines need to be defined, there are multiple CHSM definition files. Each of this file is named after the state machine's name in the HLA standard plus a file extension. The file extension is '.chsmj', meaning that the file includes a CHSM definition for a state chart and is intended to output Java Code (and therefore probably has Java code included as well).

The structure of a single .chsmj file must follow the proposed structure of CHSM and always has the following 3 sections (in this order):

- Declarations:
  This is the first section. It includes general purpose documentation and Java code which can be used in the description section. The integrated Java code can be virtually anything supported by the Java language. It will be used to integrate the resulting state machine code into the EODiSP project.

- Description:
  This is the second section. It includes the actual representation of the state machine, all states, transitions, guards, etc. In other words, it defines the whole state machine in terms of the CHSM language. As mentioned before, this section has access to the declarations section. This opens the possibility to interact with code outside the definition of this state machine (e.g. to check the state of another state machine).

- User Code:
  This is the third section. It includes user written Java code which is accessible from outside the generated state machine code (i.e. outside the generated classes). Another possible way to access methods from other classes is to mark the generated classes as public. Since this makes it obsolete to define a complete API to access the state machine, this is the preferred way for the EODiSP project. Therefore, this section remains unused.

The declarations section is not much of interest for interpreting the chsmj files. It is merely used to integrate the state machine into the rest of the EODiSP code. The interesting part is the description section, where the actual definition of the state machine takes place.

The description section consists of everything needed to generate Java code representing a single state machine. The concept for defining a state chart is the same for the HLA standard and the CHSM language. Both use the state chart definition developed by David Harel

[Har87] (which has also become part of UML). This makes it easy to map the state charts from the visual representation in the HLA standard to the textual representation in CHSM.

As stated before, a state machine consists of states, transitions and actions. In addition to this, states can be grouped into clusters. A cluster can contain states or one or more clusters. Another characteristic of clusters is that they can run simultaneously. CHSM also defines a class, which is the container for all clusters.

Therefore, we have the following hierarchy

```
1. class -> state | cluster
2. cluster -> state | cluster
```

The CHSM keyword for the main class is `chsm`, the keyword for clusters is `cluster` and for states its `state`.

Within a state, one or more transitions can be specified. The format of a transition is `eventName -> changeTo;` Whenever the event with the name `eventName` is risen, the state machine changes to the state or cluster with the name `changeTo`.

Section 7.3 describes the state machines which are converted from the HLA standard to the CHSM language for the EODiSP project. For conciseness, only the documentation section of the '.chsmj' files are given. The whole files including all sections of the chsm definition, as used in the EODiSP project, are given in B.

7.2 State Machines in the EODiSP

The HLA defines are rich set of state machines, among which only a few are automatically generated by using the CHSM approach. There are two main reasons for not using CHSM for a state machine in the project. Firstly, some of the state machines are simply too trivial. It would increase the implementation complexity if CHSM was used, without increasing stability. These state machines are implemented directly using the Java programming language. Secondly, some of the state machines are not applicable because they refer to a part of the HLA standard which is not intended to be implemented in the EODiSP (i.e. HLA services with priority 3).

Table 7.2.1 lists all state machines defined by the HLA. The first column gives an identifier of the state machine. This is a unique name given for each state machine. The name is not defined by the HLA, but should give an idea for the state machine's purpose. The second column gives a reference to the section in the HLA standard document in which the state machine is defined. The third column states if, and how it is implemented. The choices 'chsm', 'java' and 'no' are possible. 'Chsm' means that the state machine is used in the EODiSP and will be automatically generated by using the CHSM infrastructure. 'Java' means that the state machine will be used in the EODiSP and will be implemented directly by using the Java programming language. 'No' means that this state machine is not applicable to the EODiSP because it refers to a part of the HLA standard which will not be part of the EODiSP.

The HLA standard document acts as formal definition for other state machines. For those state machines which are implemented using 'chsm' (see column 3 in the table above), a CHSM definition file will be given in addition to the HLA standard.

Table 7.2.1: List of all state machines defined by the HLA standard

Identifier	***HLA ref.***	***Implementation***
Federation Lifetime	4.1	chsm
Federate Lifetime	4.1.1	chsm
Normal Activity Permitted	4.1.1	java
Sync Point	4.1.3	java
Global Synch Label	4.1.3	java
Awaiting Synchronization	4.1.3	java
Local Synch Label	4.1.3	java
Object Class	5.1.4	java
Class Attribute	5.1.4	java
Class Attribute Privilege To Delete Object	1.5.4	java
Interaction Class	1.5.4	java
Object Instance Known	6.1.1	java
Implications of Ownership of Instance Attribute	6.1.1	java
Establishing Ownership of Instance Attribute	7.1	no
Temporal State	8.1.6	no

S7.2 -1	*The EODiSP shall implement the HLA state machines as listed in table 7.2.1.*

7.3 State Machine Descriptions

7.3.1 State Machine Federation Lifetime

The following documentation is taken from the file *FederationLifetime.chsmj* in the source code of the EODiSP project. The complete file is delivered separately and is part of this SRD.

```
//*****documentation*****
/**
* This document is a representation of a state machine defined by the HLA.
```

```
* The original definition can be found in the IEEE 1516.1 specification document
* in section 4.1, figure 1--Basic states of the federation execution.
*
* The names of the transitions used in this state machine representation are taken
* from the HLA specification. They do not correspond to the names of the services.
*
* This state machine handles the following hla services related to federations.
* <ul>
* <li> Create Federation Execution (section 4.2)
* <li> Destroy Federation Execution (section 4.3)
* <li> Join Federation Execution (section 4.4) -- also in other state machine
* <li> Resign Federation Execution (section 4.5) -- also in other state machine
* </ul>
*/
```

7.3.2 State Machine Federate Lifetime

The following documentation is taken from the file *FederateLifetime.chsmj* in the source code of the EODiSP project. The complete file is delivered separately and is part of this SRD.

```
//*****documentation*****
/**
* This document is a representation of a state machine defined by the HLA.
* The original definition can be found in the IEEE 1516.1 specification document
* in section 4.1.1, figure 3--Lifetime of a federate.
*
* The names of the transitions used in this state machine representation are taken
* from the HLA specification. They do not correspond to the names of the services.
*
* This state machine handles the following hla services related to the declaration management.
* <ul>
* <li> Join Federation Execution (section 4.4) -- also in other state machine
* <li> Resign Federation Execution (section 4.5) -- also in other state machine
* <li> Request Federation Save (section 4.11)
* <li> Initiate Federation Save † (section 4.12)
* <li> Federate Save Begun (section 4.13)
```

```
* <li> Federate Save Complete (section 4.14)
* <li> Federation Saved † (section 4.15)
* <li> Request Federation Restore (section 4.18)
* <li> Confirm Federation Restoration Request † (section 4.19)
* <li> Federation Restore Begun † (section 4.20)
* <li> Initiate Federate Restore † (section 4.21)
* <li> Federate Restore Complete (section 4.22)
* <li> Federation Restored † (section 4.23)
* </ul>
*/
```

8 The JXTA Infrastructure

The JXTA infrastructure is used as transport layer in the EODiSP to support communication between simulation packages over a network. The infrastructure as such can be integrated into the EODiSP without modification. Therefore, most of the requirements defined in the user requirements document will be taken over to this document 'as is' (see the traceability matrix in appendix A). However, the JXTA infrastructure needs to be configured for each application. This configuration data will be incorporated into the dedicated application configuration document. The next section gives an overview of the possible configurations.

8.1 Configuration Properties

JXTA needs some configuration, mostly network-related, to work properly. The properties which can be configured are defined by the JXTA infrastructure itself and cannot be changed by the EODiSP.

JXTA reads a configuration file upon initialization and configures the whole infrastructure appropriately. To facilitate the configuration task for the JXTA infrastructure, a Graphical User Interface will be built. This GUI module will be part of both the simulation manager and the model manager application. The settings configured through this GUI will be stored in a dedicated file for each application instance (i.e. for each simulation manager application and for each model manager application) separately. How the files are managed and stored is explained in more detail in section 8.2.

Table 8.1.1 introduces a list with properties applicable to both the simulation manager and the model manager application. The list is not complete in the sense of JXTA, because some options will be set automatically by the EODiSP framework. However, the list is complete from the point of view of the user. Therefore, the list specifies the properties which are to be configured by the user of either the simulation manager or the model manager application.

Table 8.1.1List of configuration properties applicable to the JXTA infrastructure

Property	***Description***
Id	A unique UUID which identifies the application. This value will be automatically generated upon the first start up and should never be changed.
Name	The name of the peer. This should be a short string identifying the application.
Description	A description of the peer. This should be a text describing the peer (e.g. if it is a simulation or model manager application).
IP Address	The IP address of the network device which is used to communicate with other peers. This can also be the localhost address (127.0.01) if the EODISP works in local mode.
TCP	Specifies if the TCP protocol shall be enabled.
TCP incoming	Specifies if incoming TCP connections shall be enabled. If they are

Property	*Description*
	enabled, the appropriate port must not be blocked (e.g. by a fire-wall).
TCP outgoing	Specifies if outgoing TCP connections shall be enabled.
TCP port	Specifies the port for incoming TCP connections if TCP incoming is enabled.
TCP proxy	Specifies if a proxy service for the TCP protocol shall be used.
TCP proxy address	Specifies the IP address of the proxy service.
HTTP	Specifies the HTTP protocol shall be enabled.
HTTP incoming	Specifies incoming HTTP connections shall be enabled. If they are enabled, the appropriate port must not be blocked (e.g. by a fire-wall).
HTTP outgoing	Specifies outgoing HTTP connections shall be enabled.
HTTP port	Specifies the port for incoming HTTP connections if HTTP incoming is enabled.
HTTP proxy	Specifies if a proxy service for the HTTP protocol shall be used.
HTTP proxy address	Specifies the IP address of the proxy service.
Rendezvous	Specifies if a rendezvous server shall be used for communication.
Rendezvous address	Specifies the IP address and port of the rendezvous server.
Relay	Specifies if a relay server shall be used.
Relay address	Specified the IP address and port of the relay server.

S8.1 -1	*The EODiSP Model Manager and Simulation Manager applications shall provide a GUI-based interface to allow users to set the JXTA properties listed in table 8.1.1.*

8.2 Dedicated Configurations

As stated in the previous section, each single instance of a simulation manager application or a model manager application needs its own JXTA configuration file. The main reason for this is that each application instance needs to specify its own listening ports for incoming TCP or HTTP connections.

The main place to store the configuration file for JXTA is defined by the 'JxtaConfig' property in the application configuration file for the simulation manager application or the model manager application (see section 6.2.3 and 6.3.3 respectively). This, however, is just the main directory to store all JXTA related settings for all application instances. To be able to distin-

guish different configurations, a dedicated subdirectory will be created for each application instance which is being started. The name of the directory will be the UUID which is used to uniquely identify an application instance (see the discussion about the configuration files in chapter 6 for more information). Upon start up, the user will be presented a list of possible network (i.e. JXTA) configurations to choose from. If no configuration is present, a new one will be created. Note that this type of interaction with the EODiSP is covered by use cases UC_102 and UC_202.

A direct consequence of this configuration is, that it is not possible to run 2 instances of the same application with the same network configuration. Although this might look like a drawback, it is the normal behaviour of every network capable software. The reason is, that if one application acquires a port to listen on, no other application can acquire the exact same port. Therefore, the second application needs its own configuration.

S8.2 -1	*The EODiSP Model Manager and Simulation Manager applications shall allow users to save their JXTA configuration.*
S8.2 -2	*The EODiSP Model Manager and Simulation Manager applications shall allow users to load a previously saved JXTA configuration.*

9 General Wrapper Structure

In the EODiSP context, wrappers are used to integrate simulation packages within the EODiSP environment. They transform a generic simulation package into an EODiSP-compatible HLA federate.

There is much variety in wrapper types and structure. This section defines the requirements that apply to all EODiSP wrappers. The next two sections – sections 10 and 11 – define requirements that are specific to commonly used types of wrappers.

9.1 Wrapping Approach

In general, an EODiSP wrapper can serve two purposes:

- *Language Bridges*: the EODiSP is Java-based but most simulation packages are expected to be implemented in other languages (notably COM, C/C++ and Fortran). The wrappers are used to allow non-Java packages to be integrated in the EODiSP.
- *HLA Bridges*: the EODiSP is based on the HLA concept but simulation packages are not necessarily implemented as HLA federates. The wrappers are used to adapt their interface to conform to the interface specified by the HLA standard.

In the EODiSP, these two functions of a wrapper are implemented in three separate elements. The resulting wrapper architecture is shown in figure 8. Block 3 and 4 in the figure implement the HLA bridging function and block 2 implements the language bridging function.

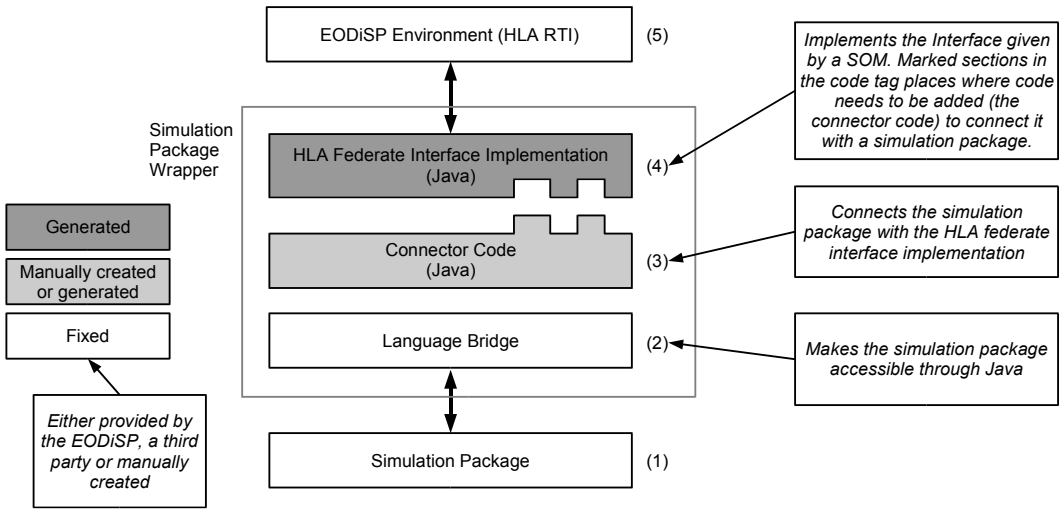

Figure 8: HLA Federate Wrapper

The choice of the architecture of figure 8 is driven by the desire to, as far as possible, automate the wrapper generation process. The part of the HLA bridge that communicates with the EODiSP environment (Block 4) needs to implement the HLA Federate Interface Specification [Hla00]. This code is the same for all federates and only depends on the SOM the feder-

ate wants to expose. Therefore, the code implementing the federate interface can be generated in the same way for any HLA federate and much of it can be generated automatically. Markers can be added in the code to tag the places where code needs to be added to connect this implementation to a simulation package (this is indicated by the two "holes" in block 4 of Figure 8).

The inputs for the generation of the block 4 code are the object class attributes the federate wants to publish or subscribe to. A SOM contains exactly this information and therefore serves as the input to the code generator.

The code that needs to be inserted in the generated code is shown in Figure 8 as block 3. This code is either inserted manually or, for some specific simulation packages, generated automatically.

In order to assist users in the construction of the wrappers for their simulation packages, the EODiSP provides three kinds of facilities:

- *HLA Federate Interface Implementation Generator*: this is a program that automatically generates the HLA federate interface implementation of a wrapper (block 4 in figure 8). This program is incorporated in the EODiSP support application.
- *Wrapper Generators*: these are programs that automatically generate the entire wrapper (the SOM and the connector code, where the former is used to generate the HLA federate interface implementation). These programs are only provided for commonly used kinds of simulation packages with a well-defined structure. The wrapper generators are incorporated in the EODiSP support application.
- *Sample Wrappers*: these are complete wrappers for particular simulation packages. They are intended to be used as blueprints for the construction of user-specific wrappers. The sample wrappers are stand-alone pieces of code that are not integrated in the EODiSP support application.

The *HLA federate interface implementation generator* is specified in subsection 9.2 below. The wrapper generators are specified in section 10. The sample wrappers are specified in section 11.

S9.1 -1	*The EODiSP wrapper generators shall be implemented as two cooperating elements: HLA federate interface implementation generator (for all wrappers) and wrapper generators (for selected kinds of wrappers).*
S9.1 -2	*The EODiSP wrapper generators shall be controlled through one single support application..*

9.2 HLA Federate Interface Implementation Generator

The HLA federate interface implementation generator generates parts of the code of an HLA federate. The input is its interface description in the form of a SOM.

Table 9.2.1 lists the types of services for which code can be generated:

Table 9.2.1: Service for which code is generated

Service	***Description***
Startup	A startup method is generated that calls appropriate HLA services to initialize the federate and join it to a federation execution.
Shutdown	A shutdown method is generated that calls appropriate HLA services to cleanly shutdown and resign the federate from a federation execution.
Publishing	For each HLA object class with sharing set to "Publish" a helper method is generated that publishes all its attributes.
Subscription	For each HLA object class with sharing set to "Subscribe" a helper method is generated that subscribes to all its attributes.
Update Attributes	For each attribute with sharing set to "Publish" an update method is generated to update a particular attribute in the federation.
Reflect Attributes	For each attribute with sharing set to "Subscribe" a callback method is generated that is invoked whenever the attribute value is updated by the federation.

S9.2 -1	*The EODiSP support application shall allow users to automatically generate an HLA federated interface implementation skeleton from a SOM.*
S9.2 -2	*The HLA federate interface skeleton generate by the EODiSP support application shall mark the points where package-specific code has to be provided by the user.*

9.3 Predefined Data Conversions

It has been specified in the URD that the EODiSP will provide a set of predefined simulation models. Each of this models should take care of one clearly defined data conversion. A more detailed analysis of the concept led to the decision that the concept itself should be changed for certain data conversions, such that it should rather be included directly into the process of generating the wrapper code.

The reason for this is that a federate needs to know its input and output values, including their formats, prior to generating the wrapper code. Data conversions can be applied to both, input and output values of a federate.

The connector code, which is generated by a support application, will provide the facility to convert values for some predefined type of conversions. The specification of the conversion is done in the support application itself.

The concept is therefore very similar to the one proposed in the URD, only that the implementation of the conversions does not reside in a dedicated federate but in the wrapper code of the federate in question itself.

This is a practical solution for some of the data conversions. Namely those, which can be clearly identified and whose conversion can be implemented in a generic way. For those data conversions which cannot be known by the EODiSP, the proposed solution in the SRD is still kept, thus EODiSP will provide sample code to implement those data conversions.

As said before, only a limited and clearly defined set of data conversions is supported by the EODiSP. Table 9.3.1 lists all types of conversions which are provided. Column 3 (Supported By) shows how the data transformation will be supported. The value can be either 'Wrapper', meaning that the data conversion will be directly supported by the wrapper code generation process, or 'Federate', meaning that a dedicated federate can be used to achieve the desired data conversion.

Table 9.3.1: List of predefined conversion types supported by the EODiSP

Conversion Type	***Description***	***Supported By***
Scaling	Application of a constant factor to a value.	Wrapper
Bias	Adding a constant to a value.	Wrapper
Type Casting	Integer to float	Wrapper
Application of Calibration Curves	Look-up table based mapping of values, interpolation	Federate
Matrix Operations	extraction of sub-matrix, transposition	Federate
Data Structure Modification	Extraction of substructures or reformatting of the structure.	Federate

S9.3 -1	*The EODiSP support application shall allow users to generate wrapper skeletons that implement the data conversions that are identified as 'Supported By Wrapper' in column 3 listed in table 9.3.1.*
S9.3 -2	*The EODiSP environment shall provide skeleton code to facilitate the implementation of the data conversions that are identified as 'Supported By Federate' in column 3 listed in table 9.3.1.*

9.4 Java/COM Bridge

Many simulation packages targeted at the MS-Windows operating system are provided with a COM interface. It is therefore useful to offer a means to interface the EODiSP to COM applications. In the prototyping activities done during the user requirement definition phase, no public domain Java/COM bridge was found that was capable of handling COM events.

During the software requirements phase it was instead found that a tweaked version of SWT [Swt] makes it possible to subscribe to COM events raised by Microsoft Excel. The next release of SWT (3.2) will have this functionality fully integrated. The SWT will therefore be used as the Java/COM bridge in the EODiSP.

10 Wrapper Generators

This section defines the requirements that apply to the wrapper generators. Wrapper generators are provided for selected kinds of simulation package whose structure allows the wrapper to be completely generated automatically. The wrapper generators are incorporated in the EODiSP support application.

The simulation packages for which wrapper generators are provided are:

- Microsoft Excel Workbooks
- SMP2 Simulations. Namely SMP2 simulations running on SimSat 2000.
- Matlab-generated Code

The input to the wrapper generation process is a mapping from the data of the simulation package to HLA object class attributes. A code generator then generates two artefacts from this input. Firstly a SOM file that can be used to generate the HLA federate interface implementation (see section 9.2) and, secondly, the *connector code* that connects the *HLA federate interface implementation* with the simulation package. This process is shown in Figure 9.

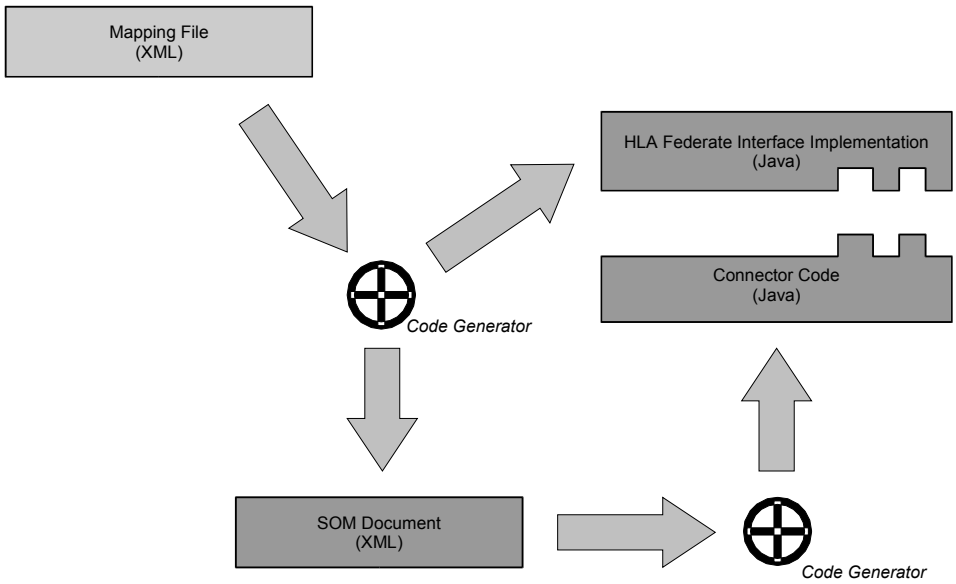

Figure 9: Wrapper generation process for predefined wrapper types

Subsections 10.1.1 and 10.2.1 describe the mapping files for each of the simulation packages named above.

The mapping files are defined through XML Schemas. They are given in their full form in appendix C. The subsections present them using a more informal (but more easily readable) graphical and textual representation (See section 6.1 for a description of this representation).

Table 10.1 lists the mapping files that are provided by the EODiSP. The last column gives an identifier for the XML Schema. The identifier is used in the traceability matrix as a concise way to refer to the XML Schemas.

Table 10.1: XML Schemas for the Wrapper Mapping Files

Schema Name	***Description***	***ID***
ExcelMapping	Mapping file for excel workbooks.	XS_003
SMP2Mapping	Mapping file for SMP2 model through SimSat 2000.	XS_004

S10 -1	*The wrapper generation process shall be implemented as it is described in figure 9.*
S10 -2	*The EODiSP wrapper generation process shall be driven by the wrapper mapping files listed in table 10.1.*

10.1 Microsoft Excel Workbook Wrapper

The EODiSP wrapper for an Excel Workbook is divided into several elements. These elements are shown in Figure 10 as block 2 to 5.

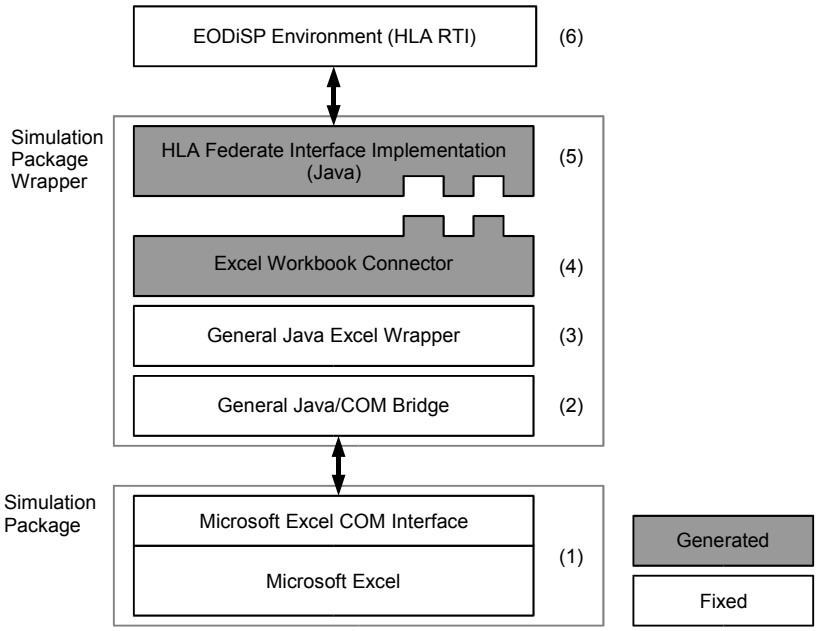

Figure 10: Structure of the EODiSP Wrapper for Excel

The fixed elements (block 2 and 3) provide easy access from Java to the Excel COM interface. This part is provided by the EODiSP as fixed code or Java library. The bridge between

Java and COM is implemented using a general Java/COM bridge. Specific methods of the Excel COM interface will be made available through dedicated Java interfaces (block 3). Among others, these include starting and stopping an Excel instance, opening and closing a workbook, setting and getting values of cells in a Worksheet as well as subscribing to events caused by a value change in a Excel Worksheet cell.

Block 4 in Figure 10 implements the connector code between the HLA federate interface implementation and the General Excel Wrapper. Namely, if there's a request to publish an HLA attribute it reads the value from the Excel Workbook and publishes it by using the methods from the HLA federate interface implementation (Block 5). Vice versa, if an HLA attribute is updated by another federate, it receives the value from the HLA federate interface implementation and writes the value to a specific cell in the Excel Workbook. Alernatively, an update of an HLA attribute can trigger a VBA macro in the Excel Workbook.

S10.1 -1	*The EODiSP support application shall provide the means the automatically generate a wrapper for an excel workbook.*
S10.1 -2	*The structure of the excel workbook wrapper shall be as in figure 10.*

10.1.1 Mapping File – ExcelMapping.xsd

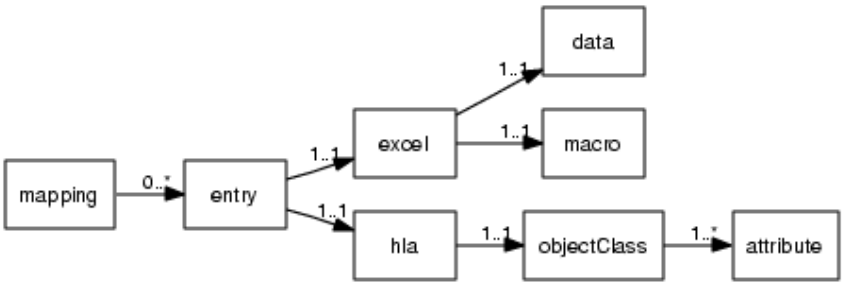

Figure 11: Structure of the ExcelMapping XML Schema

In order to automatically generate the whole HLA wrapper code for Microsoft Excel the generator needs as its input a definition of the mapping from an Excel workbook to an HLA Federation Object Model (FOM). The mapping is such that a range of cells in an Excel workbook is mapped to an attribute of a particular HLA object class in the FOM. Furthermore, the code generator needs to know the data type of the range.

Essential XML elements are described in the sections below:

Element <mapping>

Name:	***mapping***
Description:	
Parent element:	`none`

Child element(s):	`entry`
Attributes:	`xs:anyURI` **fom**

Attribute "fom"

- *type:* `xs:anyURI`
- *use:* required

References a Federation Object Model (FOM) using a Unified Resource Identifier (URI). All object classes referenced in `hla` elements need to be defined in this FOM.

Element <entry>

Name:	***entry***
Description:	An entry element describes the mapping from a range in an Excel worksheet to an attribute of an HLA object class.
Parent element:	`mapping`
Child element(s):	`excel, hla`
Attributes:	`none`

Element <excel>

Name:	***excel***
Description:	The `excel` element describes the Excel part of the Excel to HLA mapping. It either specifies a cell region (range) or a VBA macro.
Parent element:	`entry`
Child element(s):	`data, macro`
Attributes:	`none`

Element <data>

Name:	***data***
Description:	The data (cell ranges) to be mapped to an HLA attribute.
Parent element:	`excel`
Child element(s):	`none`
Attributes:	`xs:string` **worksheet** `xs:string` **range** `xs:string` **type**

Attribute "worksheet"

- *type:* `xs:string`
- *use:* required

The name of the worksheet the range is contained in.

Attribute "range"

- *type:* `xs:string`
- *use:* required

The range of cells that holds the mapped value(s). A range is specified using the Excel VBA (Visual Basic for Applications) notation for a range property. A range is specified by two cells separated by a colon. A cell is defined by its row and column position on a worksheet (e.g. `A1` for column "A", row 1). The following table shows a few examples:

Range	***Description***
`A3:A3`	Specifies the range containtin only the cell in column "A", row 3.
`A3`	Specifies the exact same cell as example above. A range specifying only one cell can ommit the second cell after the colon.
`A3:A6`	The range containing the cells `A3, A4, A5 and A6`.
`XPosition`	A range named `XPosition` in Excel.

See Excel VBA Language Reference (http://msdn.microsoft.com/library/default.asp?url=/library/en-us/vbaxl11/html/xlobjRange1_HV05204333.asp) for detailed documentation.

If the range specifies more than one cell it must be mapped to an HLA attribute of type array.

Attribute "type"

- *type:* `xs:string`
- *use:* required

Specifies the data type of the values in the range. Only one data type can be specified per range and, consequentially, all cells in the specified range need to be of the same data type.

Valid values are:

`VT_R4`	Single-precision floating-point
`VT_R8`	Double-precision floating-point
`VT_BOOL`	Boolean
`VT_I2`	Integer
`VT_I4`	Long
`VT_BSTR`	String
`VT_DATE`	Date

VT_CY	Currency

A detailed description of each data type can be found at http://msdn.microsoft.com/library/default.asp?url=/library/en-us/automat/htm/chap6_7zdz.asp.

Element <macro>

Name:	***macro***
Description:	Specifies a macro that is invoked when the corresponding HLA attribute is updated.
Parent element:	`excel`
Child element(s):	`none`
Attributes:	`xs:string` **name**

Attribute "name"

- *type:* `xs:string`
- *use:* required

Identifies the macro through its fully qualified name.

Element <hla>

Name:	***hla***
Description:	The `hla` element describes the HLA part of the Excel to HLA mapping. It specifies an attribute of an object class defined in the Federate Object Model (FOM). The FOM is given by the `fom` attribute of the root element.
Parent element:	`entry`
Child element(s):	`objectClass`
Attributes:	`none`

Element <objectClass>

Name:	***objectClass***
Description:	The HLA object class
Parent element:	`hla`
Child element(s):	`attribute`
Attributes:	`xs:string` **name**

Attribute "name"

- *type:* `xs:string`
- *use:* required

The fully qualified name of an object class (E.g. `HLARootObject.Rocket`).

Element <attribute>

Name:	***attribute***
Description:	The attribute to which the Excel range is mapped to. If more than one attribute element is given, only part of the wrapper code can be generated where other parts need to be implemented manually.
Parent element:	`objectClass`
Child element(s):	`none`
Attributes:	`xs:string` **name** `xs:string` **sharing**

Attribute "name"

- *type:* `xs:string`
- *use:* required

The name of the HLA object class attribute

Attribute "sharing"

- *type:* `xs:string`
- *use:* required

Valid values are: `publish`, `subscribe` and `publishSubscribe`. The following table describes each of them:

Value	***Description***
`publish`	`Publish` indicates that the attribute will be published to the HLA simulation environment. It is published whenever its corresponding value has changed in the Excel worksheet.
`subscribe`	`Subscribe` indicates hat the corresponding Excel range can be updated from the HLA simulation environment, namely by another federate that publishes this attribute.
`publishSubscribe`	The federate can both publish and subscribe to this attribute. Whenever the corresponding range is updated by the HLA environment it is, in response, being published to the environment.

10.2 SMP2 Simulation Wrapper

The EODiSP wrapper of an SMP2 simulation is divided into several elements. These elements are shown in Figure 12 as block 2 to 5.

The fixed components (block 2 and 3) provide easy access from Java to the SimSat SMP2 COM Adapter interface. This part is provided by the EODiSP as fixed code or Java library. The bridge between Java and COM is implemented using a general Java/COM bridge (block 2).

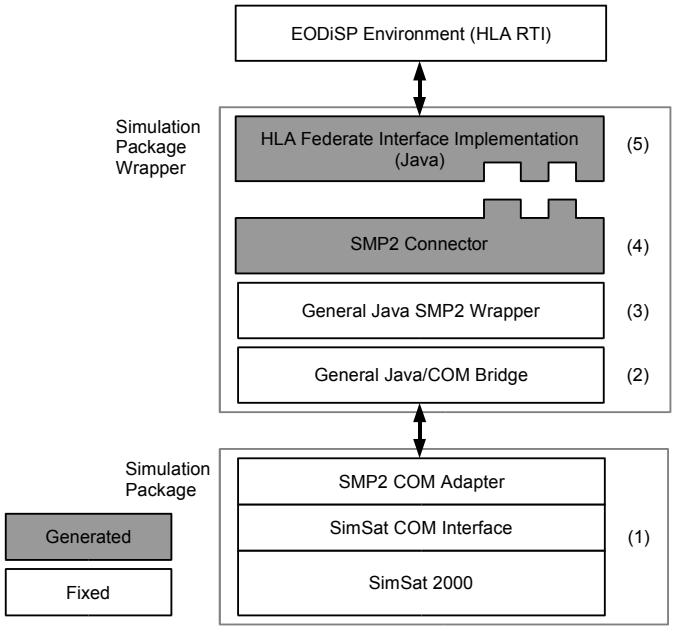

Figure 12: Structure of the EODiSP Wrapper of an SMP2 simulation

Block 4 in Figure 12 implements the connector code between the HLA federate interface implementation and the g*eneral Java SMP2 wrapper*. Namely, if there's a request to publish an HLA attribute it reads the value from an SMP2 model instance and publishes it by using the methods from the HLA federate interface implementation (Block 5). Vice versa, if an HLA attribute is updated by another federate, it receives the value from the HLA federate interface implementation and writes the value to a specific SMP2 model instance.

In order to access any data from an SMP2 model instance running in an SMP2 simulation it must fulfil the following requirements:

- The model is defined in a catalogue file
- The name of the model is known at the time the wrapper is generated. This is the case if the model instantiation is specified in an SMP2 assembly file.
- The model implements the IDynamicInvocation Interface

- The model implements the IManagedModel interface

S10.2 -1	*The EODiSP support application shall provide the means the automatically generate a wrapper for an SMP2 model controlled through a SimSat 2000 application.*
S10.2 -2	*The structure of the SMP2 wrapper shall be as in figure 10.*

10.2.1 Mapping File - SMP2Mapping.xsd

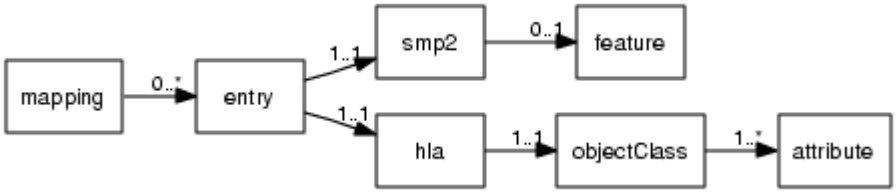

Figure 13: Structure of the SMP2Mapping XML Schema

In order to generate the HLA wrapper code for an SMP2 simulation environment a mapping from SMP 2 model instance features (fields, properties, operations etc.) to HLA Object Class attributes is needed. It is important to note that model instances (in contrast to a definition of a model) are mapped. This SMP2Mapping XML Schema defines the format of this mapping.

Essential XML elements are described in the sections below:

Element <mapping>

Name:	***mapping***
Description:	
Parent element:	`none`
Child element(s):	`entry`
Attributes:	`xs:anyURI` **fom**

Attribute "fom"

- *type:* `xs:anyURI`
- *use:* required

References a Federation Object Model (FOM) using a Unified Resource Identifier (URI). All object classes referenced in `hla` elements need to be defined in this FOM.

Element <entry>

Name:	***entry***
Description:	An entry element describes the mapping from an SMP2 model feature to one or more attributes of an HLA object class.
Parent element:	`mapping`
Child element(s):	`smp2, hla`
Attributes:	`none`

Element <smp2>

Name:	***smp2***
Description:	The `smp2` element describes the SMP2 part of the SMP2 to HLA mapping. It can specify a feature of a model instance which is going to be mapped to an object class attribute.
Parent element:	`entry`
Child element(s):	`feature`
Attributes:	`none`

Element <feature>

Name:	***feature***
Description:	
Parent element:	`smp2`
Child element(s):	`none`
Attributes:	`xs:string` **modelInstance** `xs:string` **type**

Attribute "modelInstance"

- *type:* `xs:string`
- *use:* required

Specifies an SMP2 model instance of a running SMP2 simulation to which the feature belongs to with an absolute path. The syntax of the path is described in the SMP 2.0 Component Model specification [Smpc05] in section 4.2.1.2.

Attribute "type"

- *type:* `xs:string`
- *use:* optional

SMP2 defines 10 different features for a model. Only for a few features the wrapper code can be generated automatically. The following table describes for each feature the code that can be generated. The following structure is used throughout the tables:

- Column 1: Lists the name of the SMP2 model feature
- Column 2: Indicates how much of the wrapper code can be generated automatically. The following entries are possible:
  - `complete`: The complete wrapper code can be generated automatically.
  - `~complete`: The complete wrapper code can be generated automatically if special conditions are met. The conditions are explained in column 3.
  - `skeleton`: Parts of the wrapper code can be generated automatically. For others parts only skeleton code will be generated
- Column 3: Remarks and exceptions

Feature	***Gen***	***Remarks/Exceptions***
Property	~complete	The complete wrapper code can only be generated automatically if the type of the property is one of the following: • `SimpleType` • `String` • Arrays with an item type of either `SimpleType` or `String` For more information on SMP2 types, see Chapter 4, "Core Types", in the SMP2 Metamodel Documentation [SMP2META]. Note that read-only properties need to be mapped to an HLA attribute that has a sharing type of `publish`, and `subscribe` for write-only properties respectively.
EntryPoint	complete	An entry point must be mapped to an HLA attribute with its sharing-type set to `subscribe`
Field	~complete	The complete wrapper code can only be generated automatically if the type of the property is one of the following: • `SimpleType` • `String` • Arrays with an item type of either `SimpleType` or `String` For more information on SMP2 types, see Chapter 4, "Core Types", in the SMP2 Metamodel Documentation [SMP2META].
Operation	skeleton	-

EventSource/Event-Sink	skeleton	-
NestedType	skeleton	-
Reference	skeleton	-
Container	skeleton	-
Association	skeleton	-

Element <hla>

Name:	***hla***
Description:	The `hla` element describes the HLA part of the Excel to HLA mapping. It specifies an attribute of an object class defined in the Federate Object Model (FOM). The FOM is given by the `fom` attribute of the root element.
Parent element:	`entry`
Child element(s):	`objectClass`
Attributes:	`none`

Element <objectClass>

Name:	***objectClass***
Description:	The HLA object class
Parent element:	`hla`
Child element(s):	`attribute`
Attributes:	`xs:string` **name**

Attribute "name"

- *type:* `xs:string`
- *use:* required

The fully qualified name of an object class. E.g: `HLARootObject.Rocket`

Element <attribute>

Name:	***attribute***
Description:	The attribute to which the SMP2 feature is mapped to. If more than one attribute element is given, only part of the wrapper code can be generated where other parts need to be implemented manually.
Parent element:	`objectClass`

Child element(s):	`none`
Attributes:	`xs:string` **name** `xs:string` **sharing**

Attribute "name"

- *type:* `xs:string`
- *use:* required

The name of the HLA object class attribute

Attribute "sharing"

- *type:* `xs:string`
- *use:* required

Valid values are: `publish`, `subscribe` and `publishSubscribe`. The following table describes each of them:

Value	*Description*
`publish`	`Publish` indicates that the attribute will be published to the HLA simulation environment. It is published whenever its corresponding value has changed in the Excel worksheet.
`subscribe`	`Subscribe` indicates hat the corresponding Excel range can be updated from the HLA simulation environment, namely by another federate that publishes this attribute.
`publishSubscribe`	The federate can both publish and subscribe to this attribute. Whenever the corresponding range is updated by the HLA environment it is, in response, being published to the environment.

10.3 Matlab-Generated Code Wrapper

Matlab-generated code is transformed to an SMP2 simulation model by the Mosaic tool and then integrated (by hand) into an SMP2 SimSat simulation. The wrapper for a so created SMP2 simulation can then be wrapped using the wrapper generator for SMP2 simulations (see section 10.2).

11 Sample Wrappers

This section defines the requirements for the sample wrappers. The sample wrappers are complete wrappers that are provided as blueprints to help users construct their own wrappers. Table 11.1 summarizes these sample wrappers.

Table 11.1: Summary of sample wrappers

Sample Wrapper	***Description***
Matlab Simulation Sample Wrapper	An EODiSP wrapper for a simple Simulink block that simulates the trajectory of a rocket.
Fortran Source Code Sample Wrapper	An EODiSP wrapper for a simple Fortran program that simulates the trajectory of a rocket.
C++ Source Code Sample Wrapper	A wrapper for a simple C++ class that simulates the trajectory of a rocket.
Standalone Executable Sample Wrapper	A sample wrapper for the EarthCARE simulator.
Data Processing Package Sample Wrapper	A sample wrapper for the Java library JFreeChart - a library for generating charts.

S11 -1	*The EODiSP shall provide sample wrappers for the simulation packages listed in table 11.1.*

11.1 Matlab Simulation Sample Wrapper

The sample wrapper for Matlab simulations is a wrapper for a simple Simulink block that simulates the trajectory of a rocket. Figure 14 shows the in- and outputs of this block.

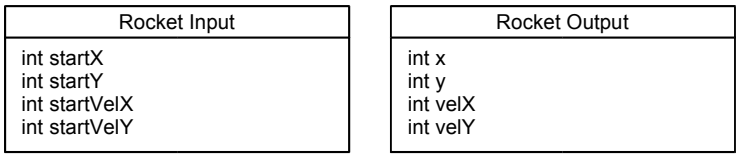

Figure 14: In-/Outputs of the Rocket Simulink block

These in- and outputs are manually translated into one federate exposing two HLA object classes. The object classes defined in the resulting SOM file are shown in Figure 15.

The behaviour of the wrapper is described in the following list:

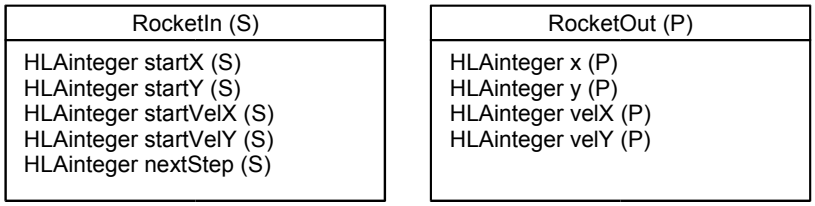

Figure 15: UML of the Rocket object classes

1. The federate wrapper registers an object class instance of type `RocketOut`. The name of the instance is set to `rocketOut`.
2. The federate subscribes to all attributes defined in the object class `RocketIn`.
3. The federate waits for new values of the `startX, startY, startVelX, startVelY` and `nextStep` attributes.
4. Whenever the federate is informed of an update of these attributes it sets the appropriate input variable in Matlab/Simulink.
5. If the `nextStep` attribute is updated it calculates the Simulink block and subsequently updates all attributes in the `rocketOut` object by reading the appropriate output values of the Simulink block.

11.2 Fortran Source Code Sample Wrapper

The sample wrapper for Fortran source code is a wrapper for a simple Fortran program that simulates the trajectory of a rocket. Figure 16 shows the subroutines of this Fortran program.

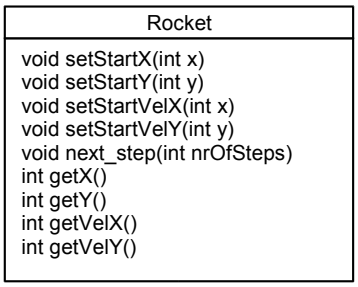

Figure 16: Subroutines of the Fortran Rocket program

The subroutines are manually translated into one federate exposing two HLA object classes which will make the interface of the Fortran program available on the federate. The object classes defined in the resulting SOM file are shown in Figure 17.

The behaviour of the wrapper is described in the following list:

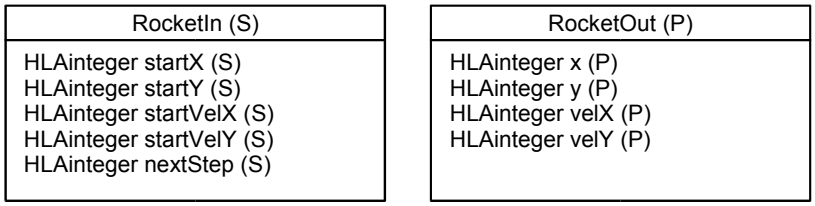

Figure 17: UML of the Rocket object classes

1. The federate wrapper registers an object class instance of type `RocketOut`. The name of the instance is set to `rocketOut`.
2. The federate subscribes to all attributes defined in the object class `RocketIn`.
3. The federate waits for new values of the `startX, startY, startVelX, startVelY` and `nextStep` attributes.
4. Whenever the federate is informed of an update of these attributes it calls the appropriate subroutine in the Fortran program (e.g. `setStartX()` when `startX` is updated).
5. If the `nextStep` attribute is updated it calls the `nextStep()` subroutine and subsequently updates all attributes in the `rocketOut` object by using getter subroutines (e.g: `getX()`).

11.3 C++ Source Code Sample Wrapper

The sample wrapper for C++ source code is a wrapper for a simple C++ class that simulates the trajectory of a rocket. Figure 18 shows the interface of this C++ class in a UML diagram.

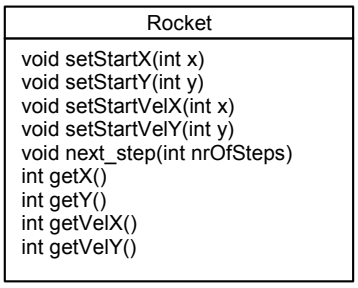

Figure 18: UML of the C++ Rocket class which shall be wrapped

The class is manually translated into one federate exposing two HLA object classes which will make the the interface of the C++ Rocket class available on the federate. The object classes defined in the resulting SOM file are shown in Figure 19.

The behaviour of the wrapper is described in the following list:

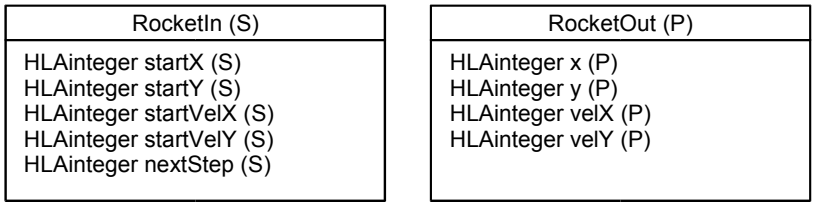

Figure 19: UML of the Rocket object classes

1. The wrapper instantiates the `Rocket` C++ class once.
2. The federate wrapper registers an object class instance of type `RocketOut`. The name of the instance is set to `rocketOut`.
3. The federate subscribes to all attributes defined in the object class `RocketIn`.
4. The federate waits for new values of the `startX, startY, startVelX, startVelY` and `nextStep` attributes.
5. Whenever the federate is informed of an update of these attributes it calls the appropriate method on the C++ Rocket instance (e.g. `setStartX()` when `startX` is updated).
6. If the `nextStep` attribute is updated it calls the `nextStep()` C++ method and subsequently updates all attributes in the `rocketOut` object by using getter methods on the C++ rocket instance (e.g: `getX()`).

11.4 Standalone Executable Sample Wrapper

The sample wrapper for a standalone executable will be done using the EarthCARE simulator [EaC04] as the simulation package. The simulation shall provide an example of wrapping the sequential execution of the EarthCARE executables `lid_filter`, `lidar` and `lidar_ret1`.

Figure 20 shows the execution order and their inputs and outputs. The input that is exported to the federate interface are two specific values of the `lidar.in` file, namely the `horizonzalResolution` and `verticalResolution`. The output is the values extracted from the file `OUTFILE_ray_para_lr.dat`. This is reflected by the SOM that makes up the interface of the federate wrapper (see Figure 20).

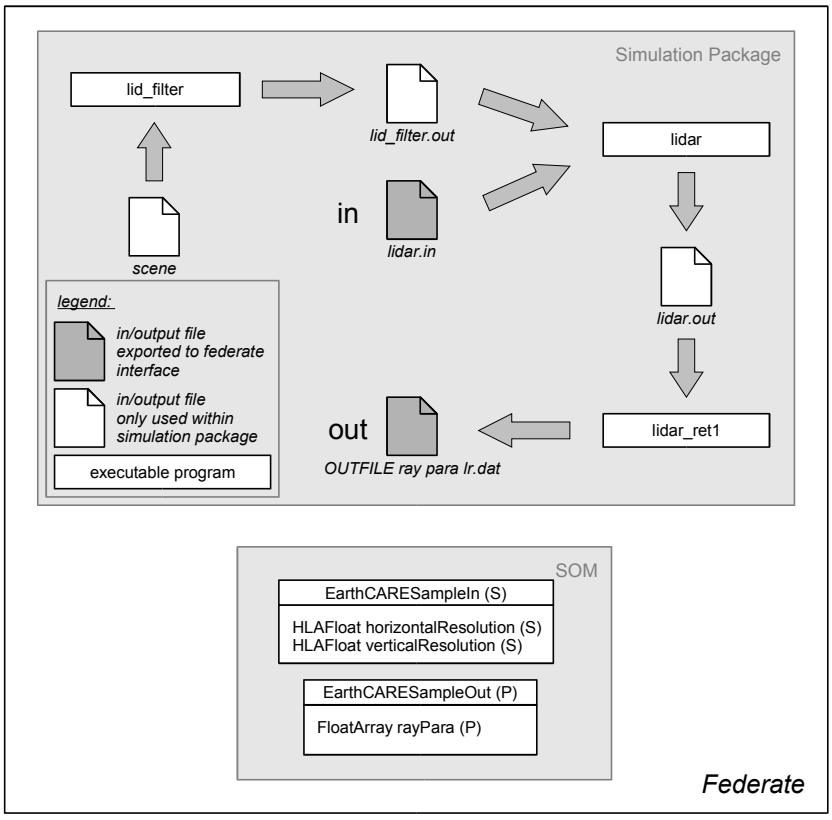

Figure 20: EarthCARE

The behaviour of the wrapper is described in the following list:

1. The federate registers an object class instance of type `EarthCARESampleOut`. The name of the instance is set to `earthCARESampleOut`.
2. The federate subscribes to the object class attributes `horizonzalResolution` and `verticalResolution`.
3. The federate waits for new values of the `horizonzalResolution` and `verticalResolution` attributes.
4. Whenever the federate is informed of an update of these attributes it writes these new values to the `lidar.in` file and starts the sequence of execution outlined in Figure 20 (`lid_filter > lidar > lidar_ret1`) and waits until the output file `OUTFILE_ray_para_lr.dat` is available.
5. The federate reads the new values from the output file and updates the `rayPara` attribute of the `earthCARESampleOut` instance.

11.5 Data Processing Package Sample Wrapper

The sample wrapper for the data processing package uses JFreeChart [Jfc] to visualize the trajectory of a rocket. It is considered that the trajectory data is provided by a federate that exposes the object class shown in Figure 21.

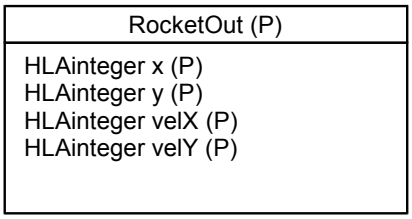

Figure 21: Object class providing the values for the JFreeChart federate

The behaviour of the wrapper is described in the following list:

1. The federate subscribes to all attributes defined in the object class `RocketOut`.
2. Whenever the federate wrapper is informed of an update of these attributes it uses the functions of JFreeChart to draw a line to the actual location of the rocket.

12 General Requirements

This section defines the software requirement that impact the EODiSP as a whole. The structure of this section mirrors that of the corresponding sections in the EODiSP URD. In general, the software requirements in this section are obtained by refining or specializing the user requirements from which they are derived.

12.1 Target Operating System

The EODiSP will be implemented in Java and will be targeted at the version 1.5 of the Java Virtual Machine.

The user requirements left the door open to the use of the Eclipse as an OS-like platform upon which to develop the GUI part of the EODiSP. The use of Eclipse was especially attractive in view of the possibility of rapidly creating editors for editing complex configuration information. This type of complex configuration information was expected to occur in the case of SOMConfig and FOMConfig files. However, as discussed in section 4.1, these two files are no longer used and hence the advantage of basing the EODiSP GUI on Eclipse are correspondingly lessened.

Given the potential drawbacks for non-specialist users arising from the need to install the Eclipse environment, the decision has been taken not to require Eclipse as a support for the EODiSP.

Note that this is a user-level decision only. Eclipse will still be used to implement some parts of the EODiSP core but this usage will be invisible to the end users and hence need not be considered at requirements level.

S12.1 -1	*The EODiSP shall be built to run on a Java Virtual Machine version 1.5.*	*T*

12.2 Licensing Requirements

The EODiSP licencing requirements are dictated by the need to produce an end application that can be made available under a GPL licence.

Table 12.2.1 lists the third-party software packages that are used in the EODiSP. The first column in the table gives the name of the package. The second column gives a brief description of the package. The last column gives the licence for the package. It is important to stress that all the above software packages are available on licence that GPL-compatible.

Table 12.2.1: List of configuration properties applicable to the JXTA infrastructure

Name	***Description***	***Licence***
JXTA	A peer-to-peer Network layer.	Sun Project JXTA Software License
SWT	Standard Widget Library.	Eclipse Public License Version 1.0
Saxon	An XSLT 2.0 Processor.	Mozilla Pubic Licence 1.0
EMF	The Eclipse Modeling Framework.	Eclipse Public License Version 1.0

Name	*Description*	*Licence*
JFreeChart	Java class library for generating charts.	LGPL

S12.2 -1	*The EODiSP shall incorporate the third-party software listed in table 12.2.1.*

12.3 Installation Requirements

The installation requirements are still TBD.

12.4 Predefined HLA Federates

There are some services that are mandated by the EODiSP URD but which are not covered by the HLA standard. As a general principle, these services are provided by predefined federated. Users who wish to have access to these services will need to deploy the predefine federates and will then have to link up to the appropriate input or output data.

At present, only one predefined federate is foreseen that will provide the services listed in table 12.4.1.

Table 12.4.1: List of services to be provided by the EODiSP Predefined Federate

Service	*Description*
Active Model	Provides an identifier of the currently active model. The active model is the next model that is due for execution (recall that in the EODiSP models are executed in sequence.
Message Logging	Allows other models to ask for string messages to be logged.

S12.4 -1	*The EODiSP shall provided a predefined federate offering the services listed in table 12.4.1.*

Appendix A: Traceability Matrix

The table in this appendix shows the traceability matrix for software requirements defined in this document against the user requirements defined in reference [Urd]. The traceability matrix shows how user requirements are mapped to software requirements. The mapping approach is discussed in section 3.1.

The first column in the table contains the reference to all the user requirements defined [Urd]. The second column contains the reference to the software requirements to which the user requirement is mapped. The software requirement reference can be one of the following (see also section 3.2):

- a reference to a use case, or
- a reference to an XML schema, or
- a reference to a state machine, or
- a reference to a software requirement

The third column contains the mapping status for the use requirement. This can be one of the following: 'FM', 'PM', or 'TO'. The value 'FM' is used for user requirements that are fully mapped to one or more software requirements. The value 'PM' is used for user requirements that are only partially mapped to software requirements. This is usually due to deliberate deviations from the requirements baseline or to lack of information at the time this document is issued. Finally, the value 'TO' designates user requirements that are taken over unchanged . These requirements are not reproduced in the present document but should be considered to be an integral part of it.

The last column in the table contains various remarks or justifications of any deviations from the requirement baseline.

UR Ref.	***SR Ref.***	***Status***	***Remarks***
R4.1-1	*S10.1-1* *S10.2-1* *S11-1*	*FM*	*Recall that Matlab generated models will be mapped through the Mosaic tool.*
R4.1-2	*TBD*	*TBD*	*At the time of writing, the Mosaic tool was not yet available to make an example for the type Matlab-generated simulation package.*
R4.1-3	*S10.1-1*	*FM*	
R4.2-3	*SOM DTD* *S12.4-1*	*PM*	*Access to general simulation information will be through the predefined federate of section 12.4*
R6.1-1	*None*	*TO*	
R6.1-2	*None*	*TO*	
R6.1-3	*None*	*TO*	
R6.2-1	*S6.2-1*	*FM*	*The EODiSP is configurable by the means of editing configuration files. Sec-*

UR Ref.	SR Ref.	Status	Remarks
			tion 6.2 specifies these configuration files in detail.
R6.2-2	S9.1-2 S10.1-1 S10.2-1 S11-1	FM	
R6.3-1	None		*This user requirement is incompatible with the use of an HLA-based infrastructure. See section 4.5.*
R6.4.1-1	UC_104 UC_106	FM	*Note that the terms 'simulation run' is a synonym for 'federation execution'.*
R6.4.2-1	UC_106	FM	
R6.4.2-2	UC_106	FM	
R6.4.2-3	UC_106	PM	*Mapping is only partial because the simulation time will not be maintained in the EODiSP since the HLA timing service is not baselined for implementation. See section 4.2.*
R6.4.2-4	UC_106	FM	
R6.4.3-1	UC_106	FM	*Note that the EODiSP does not directly provide the means to start/stop a simulation model. This must be supported by a simulation model. The simulation experiment, however, will not abort upon stopping a simulation model if in step-by-step mode. It is not guaranteed that the simulation experiment remains in a consistent state if using this feature.*
R6.4.4-1	UC_112 UC_208	FM	
R6.4.4-2	None	TO	
R6.4.4-3	None	TO	
R6.4.4-4	None	TO	
R6.4.4-5	None	TO	
R6.4.5-1	S9.3-1	FM	*The URD asks for the data conversions to be implemented in pre-defined models. The SRD specifies that they be implemented in the wrappers. There is no impact on the functionality seen by the user*
R6.5-1	None	TO	
R6.7-1	S7.2-1	FM	
G6.7-1	None	TO	
R7.2-1	None	TO	
R7.2-2	S8.1-1	FM	
R7.2-3	None	TO	
R7.2-4	None	TO	
G7.2-5	None	TO	
R7.2-6	None	TO	
R7.3-1	S8.1-1	FM	
R7.6-1	None	TO	

UR Ref.	*SR Ref.*	*Status*	*Remarks*
R8.1-1	*S9.1-2* *S9.2-1*	*FM*	
R8.1-2	*S9.2-1* *S10-1*	*FM*	*The URD asks for the wrapper generation to be XSL-based. The SRD relaxes this requirements and only refers to "code generators". There is no impact on the user.*
R8.1-3	*S9.2-2*	*FM*	
R8.1-4	*S9.2-1* *S10-2*	*FM*	
R8.1-5	*S10-2*	*PM*	*Fully automatic generation of wrappers only possible for some kinds of simulation packages. In some cases, sample wrappers are provided instead of wrapper generators.*
R8.2-1	*S10.1-2*	*FM*	*The COM wrapper is included in the excel wrapper. See figure 10.*
R8.2-2	*S10.2-1*	*FM*	*The mapping of SMP2 models is limited to models that are inside the SimSat 2000 application.*
R8.2-3	*S11-1*	*FM*	*The selected data processing package is JFreeChart. This choice is subject to agreement from ESA.*
R8.2-4	*S11-1*	*FM*	
R8.2-5	*S11-1*	*FM*	
R8.2.2-1	*S10.2-1*	*FM*	
R9.2-1	*S12-1*	*FM*	
R9.2-2	*S12-1*	*FM*	
R9.2-3	*None*		*The use of Eclipse as a support for the GUI is not baselined in the EODiSP. See section 12.1.*
R9.2-4	*None*		*The use of Eclipse as a support for the GUI is not baselined in the EODiSP. See section 12.1.*
R9.3-1	*S12-2*	*FM*	
R9.3-2	*S12-2*	*FM*	
R9.4-1	*UC_100*	*FM*	
R9.4-2	*UC_200*	*FM*	
R9.4-3	*UC_300*	*FM*	
R9.5-1	*UC_104* *S6.2-1*	*FM*	
R9.5-2	*UC_106*	*FM*	
R9.5.1-1	*None*	*TO*	
R9.5.1-2	*S6.2-1*	*FM*	
R9.5.1-3	*XS_001* *XS_010*	*FM*	
R9.5.1-4	*S6.2-1*	*PM*	*Some of the configuration files in table 9.5.1-1 have been merged and others have been dropped. See sections 4.1 and 4.3.*
R9.5.1-5	*UC_108*	*FM*	

UR Ref.	*SR Ref.*	*Status*	*Remarks*
R9.5.2-1	*S5-1*	*FM*	
R9.6-1	*UC_204*	*FM*	
R9.6-2	*UC_204*	*FM*	
R9.6.1-1	*None*	*TO*	
R9.6.1-2	*S5-1*	*FM*	
R9.6.1-3	*XS_002* *XS_020*	*FM*	
R9.6.1-4	*S6.3-1*	*PM*	*Some of the configuration files in table 9.6.1-1 have been merged. See sections 4.4.*
R9.6.1-5	*UC_206*	*FM*	
R9.6.2-1	*S5-1*	*FM*	
R9.7-1	*UC_302*	*FM*	
R9.7-2	*UC_302*	*FM*	
R9.7-3	*UC_302*	*FM*	
R9.7.1-1	*None*	*TO*	
R9.7.1-2	*None*	*TO*	
R9.7.1-3	*None*	*TO*	
R9.7.1-4	*S9.2-1* *S10-2*	*FM*	
R9.7.1-5	*None*	*TO*	
R9.7.2-1	*None*	*TO*	
G11.1.1-1	*UC_106*	*FM*	
G11.1.1-1	*UC_106*	*FM*	*The reference of this user requirement should be changed to G11.1.1-2.*

Appendix B: State Machines

This appendix presents the complete definition of the HLA state machines that are modelled in chsm. The background to their usage in the EODiSP can be found in section 7.

B.1 FederationLifetime

```
package org.eodisp.common.smcgen;

import org.eodisp.common.sm.EodispStateMonitor;

//*****documentation*****

/**
* This document is a representation of a state machine defined by the HLA.
* The original definition can be found in the IEEE 1516.1 specification document
* in section 4.1, figure 1--Basic states of the federation execution.
*
* The names of the transitions used in this state machine representation are taken
* from the HLA specification. They do not correspond to the names of the services.
*
* This state machine handles the following hla services related to federations.
* <ul>
* <li> Create Federation Execution (section 4.2)
* <li> Destroy Federation Execution (section 4.3)
* <li> Join Federation Execution (section 4.4) -- also in other state machine
* <li> Resign Federation Execution (section 4.5) -- also in other state machine
* </ul>
*
* Generation of the code can be done using the chsm compiler for java (chsm2java)
* using the following command:
* {@code $chsm2java [ options ] source-file definition-file}
*
* For the EODiSP framework, this file will automatically build by the ant build
script.
*/


//*****declarations*****

/**
*
* This classe is used to monitor the enter and exit events of states
* in the state machine.
*/
class FederationLifetimeMonitor extends EodispStateMonitor {
   public FederationLifetimeMonitor(CHSM_STATE_ARGS) {
      super(CHSM_STATE_INIT);
   }
}


%%
//*****description*****
// The main class. Contains all clusters and states defined by this state machine.
public chsm FederationLifetime is {
```

```
/**
* A federate have sent a request to resign from the currently running
* federation execution.
* We do not check the resigning federate, but we check if it is the last
* supporting federate in the execution. If this is the case, we change
* the state to 'noJoinedFederates', otherwise, we keep the state
* of 'supportingJoinedFederates'.
*/
event resignFederate %{
   //check if it is the last federate which resigns.
   return false;
%};

   // Cluster: 'FederationLifetimeRoot'.
   // The root cluster for this state machine. This is only a container including alls
   // clusters and states specified in the state machine.
   cluster FederationLifetimeRoot(notExist, Exist) {
   } is {

      // State: 'notExist'. Initial state in this cluster.
      // It means that no federation execution has yet been created.
      // This state is kind of virtual, because there is no execution to work with.
      state notExist {

         // Event: 'createFederationExecution'.
         // Change to cluster: 'Exists'
         // Defined in the IEEE 1516.1 specification document in section 4.2.
         createFederationExecution -> Exist;
      }

      // Cluster: 'Exists'.
      // When entered, this is the default mode of operation of an federation
      // execution. As long as an execution exists, one of the two states
      // within this cluster must be active.
      cluster Exist(noJoinedFederates, supportingJoinedFederates) {
      } is {

         // State: 'noJoinedFederates'. Initial state in this cluster.
         // It means that the federation execution is running, but there are
         // currently no joined federates.
         state noJoinedFederates {

            // Event: 'federateJoined'.
            // Change to state: 'supportingJoinedFederates'
            // Defined in the IEEE 1516.1 specification document in section 4.4.
            federateJoined -> supportingJoinedFederates;

            // Event: 'destroyFederationExecution'.
            // Change to state: 'notExists'
            // Defined in the IEEE 1516.1 specification document in section 4.3.
            destroyFederationExecution -> FederationLifetimeRoot.notExist;
         }

         // State: 'noJoinedFederates'.
         // It means that the federation execution is running and there are
         // joined federates in the execution. This is default state when
         // a simulation is being performed.
         state supportingJoinedFederates {
```

```
            // Event: 'resignFederate'.
            // Change to state: 'noJoinedFederates'
            // Defined in the IEEE 1516.1 specification document in section 4.5.
            resignFederate -> noJoinedFederates;

            // Event: 'joinFederate'
            // Change to state: no Change
            // Defined in the IEEE 1516.1 specification document in section 4.4.
            joinFederate %{
               // We are already in the state of supporting joined federates.
               // If a new federate joines the execution, we stay in this state,
               // without leaving the state.
            %};
         }
      }
   }
}
%%
//*****user code*****
```

B.2 FederateLifetime

```
package org.eodisp.common.smcgen;

import org.eodisp.common.sm.EodispStateMonitor;
import org.eodisp.common.sm.StateConditions;
import org.eodisp.common.sm.StateConditions.ConditionsEnum;

//*****documentation*****

/**
* This document is a representation of a state machine defined by the HLA.
* The original definition can be found in the IEEE 1516.1 specification document
* in section 4.1.1, figure 3--Lifetime of a federate.
*
* The names of the transitions used in this state machine representation are taken
* from the HLA specification. They do not correspond to the names of the services.
*
* This state machine handles the following hla services related to the declaration
management.
* <ul>
* <li> Join Federation Execution (section 4.4) -- also in other state machine
* <li> Resign Federation Execution (section 4.5) -- also in other state machine
* <li> Request Federation Save (section 4.11)
* <li> Initiate Federation Save † (section 4.12)
* <li> Federate Save Begun (section 4.13)
* <li> Federate Save Complete (section 4.14)
* <li> Federation Saved † (section 4.15)
* <li> Request Federation Restore (section 4.18)
* <li> Confirm Federation Restoration Request † (section 4.19)
* <li> Federation Restore Begun † (section 4.20)
* <li> Initiate Federate Restore † (section 4.21)
* <li> Federate Restore Complete (section 4.22)
* <li> Federation Restored † (section 4.23)
* </ul>
*
```

```
* Generation of the code can be done using the chsm compiler for java (chsm2java)
* using the following command:
* {@code $chsm2java [ options ] source-file definition-file}
*
* For the EODiSP framework, this file will automatically build by the ant build
script.
*/

//*****declarations*****

/**
* This classe is used to monitor the enter and exit events of states
* in the state machine.
*/
class FederateLifetimeMonitor extends EodispStateMonitor {
   public FederateLifetimeMonitor(CHSM_STATE_ARGS) {
      super(CHSM_STATE_INIT);
   }
}


class Conditions {
   public static int condition = 0;

   public static int getCondition() {
      return condition;
   }
}

class FederateLifetimeConditions {

   public static boolean getConsrainedCondition() {
      return StateConditions.getCondition(ConditionsEnum.FederateLifetimeConstrained);
   }

   public static boolean getTimeAdvancingCondition() {
      return
StateConditions.getCondition(ConditionsEnum.FederateLifetimeTimeAdvancing);
   }
}

%%
//*****description*****

// The main class. Contains all clusters and states defined by this state machine.
public chsm FederateLifetime is {

/**
* We check, whether a request to restore a federate fails or not. In case
* of failure, we return to state 'active', in case of success we change
* to state 'waitingforRestoreToBegin'.
*/
event requestFederationRestore() %{
   //TODO: check whether the request fails --> return false in case of failure.
   return true;
%};

   // Cluster: 'FederateLifetimeRoot'.
```

```
   // The root cluster for this state machine. This is only a container including all
   // clusters and states specified in the state machine.
   cluster FederateLifetimeRoot(initialization, JoinedFederate, stopped) {
   } is {

      // State: 'initialization'. Initial state in this cluster.
      // It means that a federate has been initialize and want to join the
      // federation execution.
      state<FederateLifetimeMonitor> initialization {

         //Event: 'joinFederationExecution'.
         // Change to cluster: 'JoinedFederate'
         // Defined in the IEEE 1516.1 specification document in section 4.4.
         joinFederationExecution -> JoinedFederate;
      }

      // Cluster: 'JoinedFederate'.
      // This cluster is a container for all cluster and states for a federate
      // when this federate has joined the federation execution.
      // A federate stays in one of this states until it resigns from the
      // execution.
      cluster JoinedFederate(concurrentComponents) {

            // Event: 'resignFederationExection'
            // Change to state: 'stopped'
            // Defined in the IEEE 1516.1 specification document in section 4.5.
            // At any time and in any state, a federate can resign from the
            // currently joined federation execution. In this case, we change
            // the state immediately to 'stopped'.
            resignFederationExection -> FederateLifetimeRoot.stopped;
      } is {

         // Set: 'concurrentComponents'.
         // A set of concurrently existing clusters. It is a container containing
         // clusters and states. These cluster can operate independently from each
other.
         set concurrentComponents(FederateActivity, FederatePermissions) {
         } is {

            // Cluster: 'FederateActivity'.
            // This cluster defines the states concerning the activity of a
            // federate while it joins a federation execution.
            cluster FederateActivity(ActiveFederate, SaveFederate, RestoreFederate) {
            } is {

               // Cluster 'ActiveFederate'. Initial cluster within this cluster.
               // This cluster contains the states while a federation is active.
               // This is the normal mode of a federate when it has joined a
               // federation execution. It can leave the active states for saving
               // or restoring.
               cluster ActiveFederate(active, restoreRequestPending) history {

                  // Event: 'initiateFederateSave †'.
                  // Change to state: 'instructedToSave'.
                  // Defined in the IEEE 1516.1 specification document in section
4.12.
                  // From any state in this cluster, a save can be initiated. Upon
returning to this
```

```
                // cluster, it will continue in the state in which it was before
(history).
                initiateFederateSave -> SaveFederate.instructedToSave;
            } is {


                // State: 'active'. Initial state in this cluster.
                // It means that the federate has joined the federation execution
                // and is active. This is the normal state for a running federate.
                state<FederateLifetimeMonitor> active {

                    // Event: 'requestFederationRestore'.
                    // Change to state: 'restoreRequestPending'.
                    // Defined in the IEEE 1516.1 specification document in section
4.18.
                    requestFederationRestore -> restoreRequestPending;

                    // Event: 'federationRestoreBegun †'.
                    // Change to state: 'preparedToRestore'.
                    // Defined in the IEEE 1516.1 specification document in section
4.20.
                    federationRestoreBegun -> RestoreFederate.preparedToRestore;

                }

                // State: 'restoreRequestPending'.
                // It means that the federate sent out a request for saving its
                // state. The RTI will response to this request by either success
                // or failure. Depending on this response, the state will be changed
                // appropriately.
                state<FederateLifetimeMonitor> restoreRequestPending {

                    // Event: 'confirmFederationRestoratonRequest †'.
                    // Change to state: 'waitingforRestoreToBegin'.
                    // Defined in the IEEE 1516.1 specification document in section
4.19.
                    // If the response of the RTI is success, this change will take
place and
                    // the state of the federate will be saved.
                    confirmFederationRestoratonRequest ->
RestoreFederate.waitingForRestoreToBegin;

                    // Event: 'confirmFederationRestoratonRequest †'.
                    // Change to state: 'active'.
                    // Defined in the IEEE 1516.1 specification document in section
4.19.
                    // If the response of the RTI is failure, this change will take
place and
                    // the state of the federate will not be saved.
                    confirmFederationRestoratonRequest -> active;
                }
            }

            // Cluster: 'SaveFederate'.
            // This cluster contains the states while a federate's state
            // is being saved. When the federate is saved, this cluster
            // will be left and cluster 'ActiveFederate' will be reentered.
            cluster SaveFederate(instructedToSave, waitingForFederationToSave,
saving) {
```

```
                  // Event: 'federationSaved †'.
                  // Change to cluster: 'ActiveFederate'.
                  // Defined in the IEEE 1516.1 specification document in section
4.15.
                  // Informs the federate, that the federation save process is
complete.
                  // An indication whether it was successful or not can be given with
                  // the save-success indicator.
                  federationSaved -> ActiveFederate;
               } is {

                  // State: 'instructedToSave'. Initial state of this cluster.
                  // It means that the RTI has initiated the federation save process.
                  // Upon entering this state, the federate will change it state to
                  // proceed with the save request.
                  state<FederateLifetimeMonitor> instructedToSave {

                     // Event: 'federateSaveBegun'.
                     // Change to state: 'saving'.
                     // Defined in the IEEE 1516.1 specification document in section
4.13.
                     federateSaveBegun -> saving;
                  }

                  // State: 'saving'.
                  // It means that a federate is in the process of saving. The RTI
                  // shall be notified that the federate is beginning to save its
sate.
                  state<FederateLifetimeMonitor> saving {

                     // Event: 'federateSaveComplete'.
                     // Change to state: 'waitingForFederationToSave'.
                     // Defined in the IEEE 1516.1 specification document in section
4.14.
                     federateSaveComplete -> waitingForFederationToSave;
                  }

                  // State: 'waitingForFederationToSave'.
                  // This notifies the RTI that the federate has completed its attempt
to
                  // save its state. It can indicate either if the attempt has
succeded or
                  // faild by the save-sucess indicator.
                  state<FederateLifetimeMonitor> waitingForFederationToSave {
                     // We don't change the state in here, therefore, no events are
present.
                     // Whenever the RTI sends the 'federationSaved' Event, this
cluster will
                     // be left. This is defined in the head of this cluster.
                  }
               }

               // Cluster: 'RestoreFederate'.
               // This cluster contains the states while a federate's state is being
               // restored. When the federate has restored (whether successful or
not),
               // this cluster will be left and the "ActiveFederate' cluster will be
               // reentered in its last state.
               cluster RestoreFederate(waitingForFederationToRestore,
waitingForRestoreToBegin, preparedToRestore, restoring) {
```

```
                // Event: 'federationRestored †'.
                // Change to cluster: 'ActiveFederate'.
                // Defined in the IEEE 1516.1 specification document in section
4.23.
                // If the restoration process has finished, this cluster is left.
                // The RTI can indicate the end of the process by this event.
                federationRestored -> ActiveFederate;

             } is {

                // State: 'waitingForRestorToBegin'. Initial state in this cluster.
                // It means that the federate has requested a federation restoration
and
                // that the response was successful.
                state<FederateLifetimeMonitor> waitingForRestoreToBegin {

                   // Event: 'federationRestoreBegun †'.
                   // Change to state: 'preparedToRestore'.
                   // Defined in the IEEE 1516.1 specification document in section
4.20.
                   federationRestoreBegun -> preparedToRestore;
                }

                // State: 'preparedToRestore'.
                // It means that the RTI has informed the federate that a
restoration is imminent.
                // The federate stops providing information to the RTI immediately.
                state<FederateLifetimeMonitor> preparedToRestore {

                   // Event: 'initiateFederateRestore †'.
                   // Change to state: 'restoring'.
                   // Defined in the IEEE 1516.1 specification document in section
4.21.
                   initiateFederateRestore -> restoring;
                }

                // State: 'restoring'.
                // It means that a the federate has been instructed to return to a
                // previously saved state. In this state, the federate restores
                // that state.
                state<FederateLifetimeMonitor> restoring {

                   // Event: 'federateRestoreComplete'.
                   // Change to state: 'waitingForFederationToRestore'.
                   // Defined in the IEEE 1516.1 specification document in section
4.22.
                   federateRestoreComplete -> waitingForFederationToRestore;
                }

                // State: 'waitingForFederationToRestore'.
                // It means that the RTI has been notified that the federate has
completed
                // the requested attempt to restore a previous state.
                state<FederateLifetimeMonitor> waitingForFederationToRestore {
                   // We don't change the state in here, therefore, no events are
present.
                   // Whenever the RTI sends the 'federationRestored' Event, this
cluster will
                   // be left. This is defined in the head of this cluster.
                }
```

```
               }
            }

            // Cluster: FederatePermissions.
            // This cluster defines the states concerning permissions to perform
events. This
            // cluster runs concurrently to the 'FederateActivity' cluster.
            cluster FederatePermissions(normalActivityPermitted,
normalActivityNotPermitted) {
            } is {

            // State: normalActivityPermitted. Initial state of this cluster.
            // It means, that if the federate is in any active state, permissions for
activities are granted.
            // This is the normal case if a federate is running. Whenever the state
machine indicates a leave from
            // the 'ActiveFederate' cluster, the permissions will be withdrawn by
changing the state to 'normalActivityNotPermitted'.
            state<FederateLifetimeMonitor> normalActivityPermitted{
exit(FederateActivity.ActiveFederate) -> normalActivityNotPermitted; }

            // State: 'normalActivityNotPermitted'.
            // It means that the federate is not in an active state and the
permissions for activities are not granted.
            // Whenever the state machine reenters the 'ActiveFederate' cluster, the
state will be changed.
            state<FederateLifetimeMonitor> normalActivityNotPermitted{
enter(FederateActivity.ActiveFederate) -> normalActivityPermitted; }
         }
         }
      }

      //The federate has resigned from the joined federate state and is in its end
state (stopped).
      // State: stopped.
      // It means that the federate has left the federation execution. It is therefore
no longer present
      // for the simulation. For the state machine, the federate resides in the
stopped state.
      // The federate cannot leave this state. It is the end state.
      state<FederateLifetimeMonitor> stopped {
         // End state.
      }
   }
}

%%
//*****user code*****
```

Appendix C: XML Schemas

This appendix lists all XML Schema files defined in this document. Documentation elements (xs:annotation) are omitted to increase readability. For each file a reference name, the name of the XML Schema file and the chapter describing its documentation is given.

C.1 SimulationManagerProjectFile.xsd

Reference:	XS_001
Name:	SimlationManagerProjectFile.xsd
Documentation:	Section 6.2.1

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns="http://www.w3.org/1999/xhtml"
xmlns:xs="http://www.w3.org/2001/XMLSchema">
 <xs:element name="project">
  <xs:complexType>
   <xs:sequence>
    <xs:element name="federations" minOccurs="1">
     <xs:complexType>
      <xs:sequence>
       <xs:element name="federation" maxOccurs="unbounded" minOccurs="0">
        <xs:complexType>
         <xs:sequence>
          <xs:element name="federate" minOccurs="0" maxOccurs="unbounded">
           <xs:complexType>
            <xs:attribute name="uri" type="xs:anyURI" use="required">
            </xs:attribute>
           </xs:complexType>
          </xs:element>
         </xs:sequence>
         <xs:attribute name="name" type="xs:ID" use="required">
         </xs:attribute>
        </xs:complexType>
       </xs:element>
      </xs:sequence>
     </xs:complexType>
    </xs:element>
    <xs:element name="experiments" minOccurs="1">
     <xs:complexType>
      <xs:sequence>
       <xs:element name="experiment" maxOccurs="unbounded" minOccurs="0">
        <xs:complexType>
         <xs:sequence>
          <xs:element name="federationExecution" maxOccurs="unbounded" minOccurs="0">
           <xs:complexType>
            <xs:sequence>
             <xs:element name="init" maxOccurs="unbounded" minOccurs="0">
              <xs:complexType>
               <xs:attribute name="federate" type="xs:string" use="required">
               </xs:attribute>
               <xs:attribute name="resource" type="xs:anyURI" use="required">
               </xs:attribute>
               <xs:attribute name="type" type="xs:string" use="required">
               </xs:attribute>
              </xs:complexType>
             </xs:element>
            </xs:sequence>
```

```
           <xs:attribute name="federation" type="xs:IDREF" use="required">
           </xs:attribute>
          </xs:complexType>
         </xs:element>
        </xs:sequence>
        <xs:attribute name="name" type="xs:string" use="required">
        </xs:attribute>
       </xs:complexType>
      </xs:element>
     </xs:sequence>
    </xs:complexType>
   </xs:element>
  </xs:sequence>
  <xs:attribute name="name" type="xs:string" use="required">
  </xs:attribute>
 </xs:complexType>
</xs:element>
</xs:schema>
```

C.2 ModelManagerProjectFile.xsd

Reference:	XS_002
Name:	ModelManagerProjectFile.xsd
Documentation:	Section 6.3.1

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns="http://www.w3.org/1999/xhtml"
xmlns:xs="http://www.w3.org/2001/XMLSchema">
 <xs:element name="project">
  <xs:complexType>
   <xs:sequence maxOccurs="1" minOccurs="1">
    <xs:element name="federates" minOccurs="0" maxOccurs="1">
     <xs:complexType>
      <xs:sequence>
       <xs:element name="federate" maxOccurs="unbounded" minOccurs="0">
        <xs:complexType>
         <xs:sequence>
          <xs:element name="initDecl" minOccurs="0" maxOccurs="unbounded">
           <xs:complexType>
            <xs:attribute name="type" type="xs:string" use="required">
            </xs:attribute>
            <xs:attribute name="template" type="xs:anyURI" use="optional">
            </xs:attribute>
            <xs:attribute name="spec" type="xs:anyURI" use="optional">
            </xs:attribute>
            <xs:attribute name="required" type="xs:boolean" use="required">
            </xs:attribute>
           </xs:complexType>
          </xs:element>
          <xs:element name="trustedSimulationManagers" minOccurs="0" maxOccurs="1">
           <xs:complexType>
            <xs:sequence>
             <xs:element name="ref" minOccurs="0" maxOccurs="unbounded">
              <xs:complexType>
               <xs:attribute name="uri" type="xs:anyURI">
               </xs:attribute>
              </xs:complexType>
             </xs:element>
            </xs:sequence>
```

```
          </xs:complexType>
         </xs:element>
        </xs:sequence>
        <xs:attribute name="id" type="xs:ID" use="required">
        </xs:attribute>
        <xs:attribute name="som" type="xs:anyURI" use="required">
        </xs:attribute>
        <xs:attribute name="public" type="xs:boolean" use="required">
        </xs:attribute>
       </xs:complexType>
      </xs:element>
     </xs:sequence>
    </xs:complexType>
   </xs:element>
  </xs:sequence>
  <xs:attribute name="name" type="xs:string" use="required">
  </xs:attribute>
 </xs:complexType>
</xs:element>
</xs:schema>
```

C.3 ExcelMapping.xsd

Reference:	XS_003
Name:	ExcelMapping.xsd
Documentation:	Section 10.1.1

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns="http://www.w3.org/1999/xhtml">
 <xs:element name="mapping">
  <xs:complexType>
   <xs:sequence>
    <xs:element name="entry" minOccurs="0" maxOccurs="unbounded">
     <xs:complexType>
      <xs:sequence>
       <xs:element name="excel">
        <xs:complexType>
         <xs:choice>
          <xs:element name="data">
           <xs:complexType>
            <xs:attribute name="worksheet" type="xs:string" use="required">
            </xs:attribute>
            <xs:attribute name="range" type="xs:string" use="required">
            </xs:attribute>
            <xs:attribute name="type" use="required">
             <xs:simpleType>
              <xs:restriction base="xs:string">
               <xs:enumeration value="VT_R4"/>
               <xs:enumeration value="VT_R8"/>
               <xs:enumeration value="VT_BOOL"/>
               <xs:enumeration value="VT_I2"/>
               <xs:enumeration value="VT_I4"/>
               <xs:enumeration value="VT_BSTR"/>
               <xs:enumeration value="VT_DATE"/>
               <xs:enumeration value="VT_CY"/>
              </xs:restriction>
             </xs:simpleType>
```

```
          </xs:attribute>
         </xs:complexType>
        </xs:element>
        <xs:element name="macro">
         <xs:complexType>
          <xs:attribute name="name" type="xs:string" use="required">
          </xs:attribute>
         </xs:complexType>
        </xs:element>
       </xs:choice>
      </xs:complexType>
     </xs:element>
     <xs:element name="hla">
      <xs:complexType>
       <xs:sequence>
        <xs:element name="objectClass">
         <xs:complexType>
          <xs:sequence>
           <xs:element name="attribute" minOccurs="1" maxOccurs="unbounded">
            <xs:complexType>
             <xs:attribute name="name" type="xs:string" use="required">
             </xs:attribute>
             <xs:attribute name="sharing" use="required">
              <xs:simpleType>
               <xs:restriction base="xs:string">
                <xs:enumeration value="publish"/>
                <xs:enumeration value="subscribe"/>
                <xs:enumeration value="publishSubscribe"/>
               </xs:restriction>
              </xs:simpleType>
             </xs:attribute>
            </xs:complexType>
           </xs:element>
          </xs:sequence>
          <xs:attribute name="name" type="xs:string" use="required">
          </xs:attribute>
         </xs:complexType>
        </xs:element>
       </xs:sequence>
      </xs:complexType>
     </xs:element>
    </xs:sequence>
    <xs:attribute name="fom" type="xs:anyURI" use="required">
    </xs:attribute>
   </xs:complexType>
  </xs:element>
</xs:schema>
```

C.4 SMP2Mapping.xsd

Reference:	XS_004
Name:	SMP2Mapping.xsd
Documentation:	Section 10.2.1

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns="http://www.w3.org/1999/xhtml" id="SMP2Mapping">
 <xs:element name="mapping">
  <xs:complexType>
   <xs:sequence>
    <xs:element name="entry" minOccurs="0" maxOccurs="unbounded">
     <xs:complexType>
      <xs:sequence>
       <xs:element name="smp2">
        <xs:complexType>
         <xs:sequence>
          <xs:element name="feature" minOccurs="0" maxOccurs="1">
           <xs:complexType>
            <xs:attribute name="modelInstance" type="xs:string" use="required">
            </xs:attribute>
            <xs:attribute name="type">
             <xs:simpleType>
              <xs:restriction base="xs:string">
               <xs:enumeration value="property"/>
               <xs:enumeration value="entryPoint"/>
               <xs:enumeration value="field"/>
               <xs:enumeration value="operation"/>
               <xs:enumeration value="eventSource"/>
               <xs:enumeration value="eventSink"/>
               <xs:enumeration value="nestedType"/>
               <xs:enumeration value="reference"/>
               <xs:enumeration value="container"/>
               <xs:enumeration value="associaton"/>
              </xs:restriction>
             </xs:simpleType>
            </xs:attribute>
           </xs:complexType>
          </xs:element>
         </xs:sequence>
        </xs:complexType>
       </xs:element>
       <xs:element name="hla">
        <xs:complexType>
         <xs:sequence>
          <xs:element name="objectClass">
           <xs:complexType>
            <xs:sequence>
             <xs:element name="attribute" minOccurs="1" maxOccurs="unbounded">
              <xs:complexType>
               <xs:attribute name="name" type="xs:string" use="required">
               </xs:attribute>
               <xs:attribute name="sharing" use="required">
                <xs:simpleType>
                 <xs:restriction base="xs:string">
                  <xs:enumeration value="publish"/>
                  <xs:enumeration value="subscribe"/>
                  <xs:enumeration value="publishSubscribe"/>
                 </xs:restriction>
                </xs:simpleType>
               </xs:attribute>
              </xs:complexType>
             </xs:element>
            </xs:sequence>
            <xs:attribute name="name" type="xs:string" use="required">
            </xs:attribute>
           </xs:complexType>
          </xs:element>
```

```
        </xs:sequence>
       </xs:complexType>
      </xs:element>
     </xs:sequence>
    </xs:complexType>
   </xs:element>
  </xs:sequence>
  <xs:attribute name="fom" type="xs:anyURI" use="required">
  </xs:attribute>
 </xs:complexType>
</xs:element>
</xs:schema>
```