A New Standard for Simulation Model´s Portability and its Implementation in Simulus

Nuno Sebastiao 1*,and N Di Nisio2†
[1]*European Space Agency, European Space Operations Centre, Darmstadt, Germany.*

[2]*TERMA GmbH, Darmstadt, Germany.*

I. Introduction

The main purpose of Simulation Model Portability (SMP2) is to promote platform independence, interoperability and reuse of simulation models across simulations and simulation infrastructures. This is achieved by applying the following principles:

- Platform Abstraction: The SMP2 standard based on the concepts outlined by the Model Driven Approach (MDA) which defend the clear separation of modelling of components between the actual modelling and the mapping to a given programming language. All SMP2 models must be built using common high-level concepts addressing fundamental modelling issues. This enables the development of models on an abstract level, which is essential for platform independence and reuse of models.
- Common Type System: All SMP2 models must be built upon a common type system. This enables different models to have a common understanding of the syntax and semantics of basic types, which is essential for interoperability between different models.

The implementation of the SMP2 standard at the European Space Operations Centre (ESOC) is done in the scope of the Simulator Infrastructure (SIMSAT 4.0) development project and in the EGOS Modelling Framework (EGOS-MF). This work can be segmented into two main areas:

- Tooling to support SMP2 Models Development: Inline with the platform abstraction goals, SMP2 defines the Simulation Model Definition Language (SMDL) which is used to define catalogues of models in a platform Independent manner. It is then necessary to provide the necessary tooling support that allows the creation, validation, assembly, packaging and scheduling of models.
- The upgrades to the Simsat Kernel: SMP2 mandates that any given infrastructure must implement and provide a series of services and interfaces in order to be able to host simulations based on the standard.

This paper will detail how the different aspects of the SMP2 standard are being implemented in SIMSAT 4.0 and in EGOS-MF. Lastly this paper will attempt to provide some outlook to the future evolution of both the standard and Simsat itself.

This paper will also provide some insight to the use cases where ESOC infrastructure intends to leverage on SMP2 as a platform for the development of reference spacecraft simulation architecture and contribution to the development of generic/re-usable SMP-2 model libraries.

* Software Engineer, D/OPS-GIC, Nuno.Sebastiao@esa.int.
† Software Engineer, Space, nin@terma.com.

II. Supporting the full simulation life cycle

As depicted in Figure 1, the SMP2 standard and its implementation at ESOC provides the necessary tooling

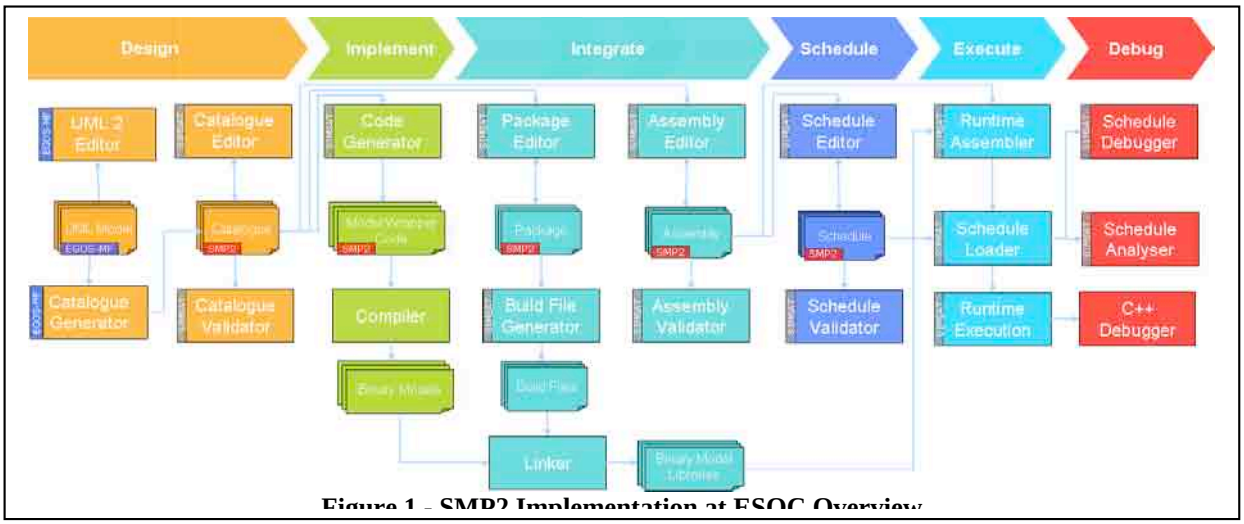

Figure 1 - SMP2 Implementation at ESOC Overview

support throughout the entire simulation development life cycle, since the design phase of simulation models to the runtime execution and debugging of the same models instances.

A. Simulation Design

The simulation design phase represents the Platform Independent Model (PIM) of the simulated system, i.e. the specification level of SMP2 models. The main concern in this phase is to define model specifications (in one or more Catalogue file) and to ensure their validity against the SMP2 Standard.

A catalogue document holds individual simulation components that are organised within nested namespaces, where each namespace can contain a collection of Models, Class, Structure, and SMP2 specific types. A catalogue can also reference other catalogues to group certain sub system level functionality within a single document.

Simsat 4.0 supports the simulation design phase by providing a Catalogue Editor and Catalogue *Validator*, see Figure 2, which leverage the modeller of having to interact directly with the underlying XML specification of the SMDL language.

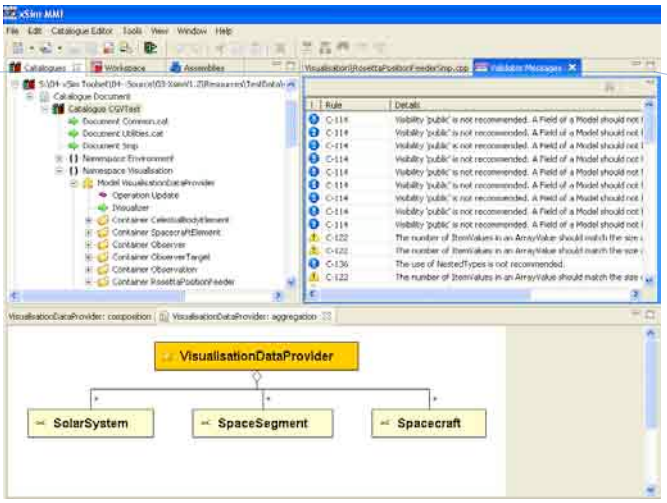

Figure 2 - Simsat 4.0 Catalogue Editor and Code Generator

The catalogue editor allows editing of catalogue files that confirm to the SMDL specification through a well-defined schema[4]. The editor acts as a view onto the underlying schema data by means of a set of graphical viewer components. Each of the viewer components are linked with the underlying schema document structure to permit changes.

The catalogue *validator* is mainly used as an extension to the catalogue editor to support catalogue validation. Validation can be performed on a complete catalogue or a section of a catalogue depending on the selection context and background validation settings. Two levels of validation are provided by Simsat 4.0.

- *Primary Validation*: known as *well-formed validation*, occurs during the loading and caching of the source document. At this stage the document is validated against its XML Schema Document (**XSD**) to verify it its well-formed and complies with the schema. Well-formed validation is required due to the hierarchical nature of the document and to allow the tooling to represent the document visually.
- *Secondary Validation*: known as *semantic validation*, occurs under various scenarios. During a semantic validation the content of the data is validated against the SMDL standards. The identifiable

content sections include Class/Interface/Field *References* as well as *Links* to *Required* and *Provided* Interfaces. The secondary validation occurs under the *user-initiated* validation scenario. The purpose for semantic validation is to prevent errors during code generation and compilation stages as well as to maintain a standards compliant source document.

The representation of the output produced by the integrated validator includes an error level, error description and possible solutions and location of where in the document the error occurred. The error entry itself can be selected with the ability to navigate to the appropriate entry within the catalogue tree.

EGOS-MF support for SMP2 Modelling

The EGOS-Modeling Framework (EGOS-MF) provides a framework that supports the modelling of space data systems and that helps improving efficiency and consistency in European Ground Operations System (EGOS) software development.

Regarding simulation modeling and particularly SMP2 model design, the EGOS-MF will be used to make the model driven development process and associated productivity enhancements accessible to simulation model developments targeting SMP2 while still keeping the modeling only at the UML level.

EGOS-MF will support SMP2 catalogue development by providing the following capabilities:

- support for simulation modeling in UML conforming with SMP2 by using UML profiles,
- support for generating SMP2 Catalogue(s) from the UML model(s)
- support for reverse-engineering of SMP2 catalogues into UML models
- integration with SIMSAT, especially with the SIMSAT Catalogue Validator to allow validating generated SMP2 catalogues using the SIMSAT validation engine.

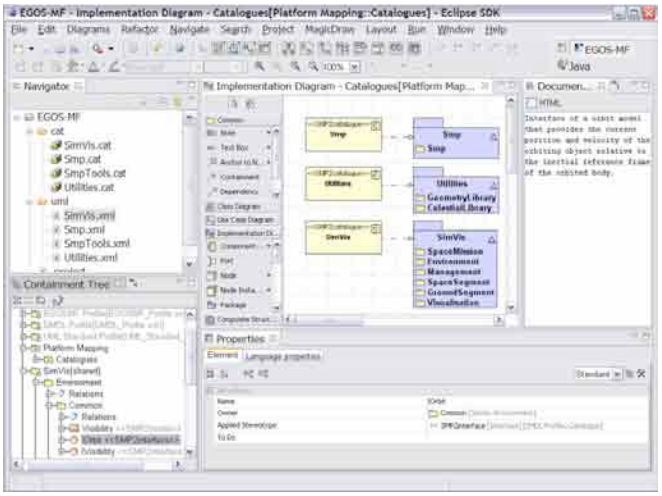

Figure 3 - EGOS-MF UML editing of SMP2 Catalogues

B. Simulation Implementation

The SMP2 standard defines a clear mapping between the platform independent specification held in the catalogue and the C++ programming language[2]. The responsibility of translating the information held in the model catalogues into ANSI C++ compliant source is within the code generator who must parse the SMDL documents containing the catalogues and model specification and produce compilable source code.

SMP2 currently only defines and provide mechanisms for the way a given model or element interacts with other models or elements and what information does a model supplies to the outside world. However it does not specify the internal behavioural mechanisms of a model, i.e. what operations does it do with the information data it receives and this needs to be done directly in the source produced by the source code generator.

Simsat 4.0 addresses this phase by providing a fully SMP2 Compliant C++ code generator, that maps the catalogue information into compilable C++ source code. The code generator provided in Simsat 4.0 will support only forward engineering, i.e. it will not support the reverse engineering of already existing model source code into catalogue information. Additionally, it will not support code merging due to the complicated comparison rules that would be required for this functionality.

C. Simulation Integration

The simulation Integration phase represents the assembly of both the catalogue components and the assembly of the produced binary code.

This phase has a two fold objective, where on one side it defines what will be the runtime structure of a simulation and on the other side it defines what will be the physical packaging of models according to the target platform, e.g. static of shared objects in Linux or Dynamic Link Libraries (DLL) in the Windows platform.

1. Model Assembly

An Assembly document, as specified in the SMP2 Standard, mainly references the catalogue documents in order to identify model instances that should be created in memory during runtime and the relationships between the different models in terms of event sources/sinks and interface providers/consumers.

The assembly editor provides simulation integrators with support to build integrated simulations that consist of model instances, links, and event definitions. This includes mechanisms for creating links between model instances, namely between references, and implemented interfaces, between event sources and event links, and between output and input fields.

Together with the Assembly editor, there is an assembly *validator* that will be used as an extension to allow validation of assembly documents while they are being edited. The validation rules of an assembly document are identical to those of a catalogue document from both a primary and secondary validation points of view, but applicable to rules defined for assemblies in the SMP2 SMDL specification.

Simsat 4.0 provides the same approach and support for the creation and edition of assembly's in the Assembly editor as for the Catalogue Editor depicted in Figure 2.

2. Model Packaging

A Package is a Document that holds an arbitrary number of Implementation elements. Each of these implementations references a type in a catalogue that shall be implemented in the package. In addition, a package may reference other packages as a Dependency.

Simsat 4.0 provides tooling, as for the Catalogue and Assembly Editor, for the creation, edition and validation of Package document conforming to the SMP2 Metamodel definition[4]. In addition it also provides:

- Package *Validator* component providing a service that performs validation of a package document against the SMP2 standard.
- Build File Generator which is responsible for the generation of the build files that will allow the linking of the generated and compiled source code in the form of shared objects that will be loaded in the Simsat runtime environment.

D. Simulation Scheduling

Schedules reference the assemblies to co-ordinate scheduling policies for model execution entry points. A schedule defines how the model instances and field links of an assembly are to be scheduled. It includes timed events and cyclic events, and tasks triggered by these events. Additionally, it may specify the origins of epoch time and mission time via its *EpochTime* and *MissionStart* elements, respectively.

Simsat 4.0 provides the same level of tooling as for the catalogue and assembly document management which can be summarised as:

- Creation, Loading and Editing of SMP2 conformant documents[4].
- Loading of catalogue and assembly documents that are contained within the schedule document as links.
- On demand cross-document link resolving between different types of documents when browsing the schedule document.
- Graphical timeline view of selected schedule documents. Each single event of the Schedule Document being edited is displayed according to the schedule data and the timeline scale.
- A Validation mechanism providing a service that performs validation of a schedule document against the SMP2 standards[4].

E. Simulation Execution

The simulation execution deals with the runtime aspect of a simulation as opposed to the previous sub sections that were strictly related to the development and build of a simulation.

In this phase the simulation information, in the form of compiled objects together with the assembly and schedule information, is loaded in the simulation runtime kernel. In order to facilitate the inter-operability between SMP2 compliant simulation execution environments (i.e. run-time simulation kernels), several Simulation Services are defined in the SMP2 specification. For the sake of clarity the main data types, interfaces and services that a runtime SMP2 Compliant platform format must provide are summarised below:

- Data Types

In order to allow the interoperability between simulation environments it is necessary to have a common set of data types consistently available and supported by every simulation runtime that is to be SMP2 compliant as they are

the basic mechanism used to represent data and upon which complex data types are constructed. If these data types are not defined and used consistently it will not be possible to exchange models between simulation environments.

SMP2 standardizes on types definition for Boolean types, integer types, floating types, data and time types, Strings, Universal Unique Identifier (UUID) and Collections of elements. In order to avoid the need of having to map the different SMP2 Types to the different programming languages and processors where an SMP2 Execution Environment might exist, the mapping is done against the CORBA IDL type definition. This eases the responsibility of the SMP2 Mapping as the CORBA data types mapping to the different programming languages is already standardised.

- Mandatory Interfaces and services

The SMP2 specification distinguishes between mandatory and optional interfaces. A *mandatory* interface is one that every component shall support in order to inter-operate with other components (models or services). An *optional* interface is one that a component may implement. Here the main mandatory interfaces and services are highlighted.

The main interface that a Simulation Environment must implement is the *ISimulator* interface[1] so that it can give access to the models and services. This is the interface responsible for the transition between the different simulator states, with well defined state transition methods between these states as described in Figure 4.

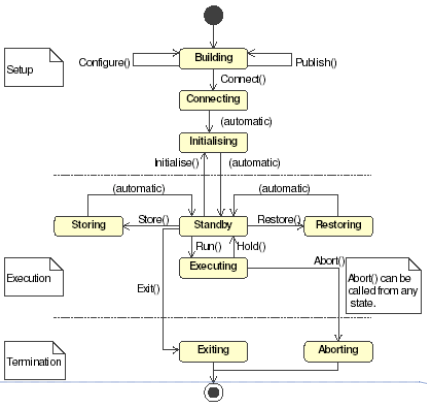

Figure 4 - Simulation Environment State Diagram with State Transition Methods

IDynamicSimulator: This interface extends the *ISimulator* interface and adds methods to dynamically create components (typically models) from component factories. It makes use of the *IFactory* interface for component factories.

IPublication: Provide functionality to allow publishing members, including fields and operations. As part of the initialisation, every model needs to be given access to a publication receiver to publish its fields and operations.

- Mandatory Services

Logger: The logger service provides a method to send a log message to the simulation log file.

Time Keeper: SMP2 supports four different kinds of time. The time managed by the Time Keeper simulation service is called Simulation Time. The service keeps track of simulation time and puts it into relation with epoch time and mission time. Further, the service provides Zulu time based on the clock of the computer.

Scheduler: The scheduler service calls entry points of models based on events triggered by one of the four time kinds.

Event Manager: The event manager service provides a global notification mechanism.

Simsat 4.0 supports this phase by providing an SMP2 Adapter that allows loading and executing simulations that conform to the SMP2 Standard. More on this can be found in section I - The Simsat 4.0 implementation of the SMP2 Runtime Specification.

F. Simulation Debugging

The notion of Simulation debugging is split into two main concepts:

- Scheduling Debugger

SIMSAT provides a facility to debug scheduled events. In order to do this, the Scheduler must be in Debug mode. It is possible to place breaks on particular Simsat native events, in much the same manner that a software debugger allows the user to place breaks on particular lines of code. When the event is about to be executed, the execution of the simulation is halted until commanded to continue. It is also able to step forward one event at a time. It is possible to perform several operations in the scheduler debugger, such as setting a breakpoint on a specified event, cancelling a break point, list the defined breakpoints, step to the next event to be executed on the schedule (not necessarily one with a breakpoint) and continues the execution of the simulation, until an event is encountered on which a break has been set.

In addition to the scheduling debugging it is also possible to configure the simulation so that a runtime analysis of the scheduled events is produced. The output of such analysis can then be used to determine which, if any, are the

events that consume most of the simulation time, possibly leading to simulation execution slippage within relation to real time.

Please note that SMP2 as observed in Figure 4 - Simulation Environment State Diagram with State Transition Methods, does not mandate the debug mode. However it is possible to provide this simulation mode natively without impacting the SMP2 standard.

- Model binary object debug

Being the executed source code C++ it is possible to attach any C++ debugger to the simulation kernel and set breakpoints on the model code as in any other C++ program.

III. Methodologies and technologies

Simsat 4.0 makes use of a disparate set of methodologies and technologies to leverage its development and delegates, whenever possible, functionality to already existing open source third party tools.

Before presenting the actual methodologies and technologies used throughout the project development it is important to distinguish between two distinct aspects of Simsat 4.0.

- The Graphical User Interfaces: Simsat 4.0 operates using a client server approach where the client is composed of the modelling tools (corresponding to the Design, Implement, Integrate and Schedule phase present in Figure 1) that provide a Graphical User Interfaces (GUI) to allow the creation and manipulation of catalogues, schedules, assemblies and packages. In addition a GUI is developed to provide a view of the simulation during runtime execution.
- The Kernel Runtime: This is a backend server application without any direct user interaction. Nonetheless it exposes a set of CORBA interfaces that allow it to be controlled by a remote GUI.

G. The modelling tooling

Simsat 4.0 provides the tools needed to support the entire life cycle development of SMP2 compliant models. For the modelling tool it was decided to make use of several projects developed under the Eclipse Foundation umbrella that would leverage the development of the required editors, *validators* and code generators.

As the metamodel language of the SMP2 standard is defined using the Extensible Markup Language (XML) the Eclipse Modelling Framework was found suitable to represent all XML documents in memory. By using the Eclipse Modelling Framework (EMF) the source code implementation of the SMP2 Metamodel can be machine generated through the EMF code generator, although not without some customisation as EMF natively generates Java code and SMP2 is only mapped to C++. The code generation is based on the schemas provided by the SMP2 Metamodel Specification and conforms to the C++ Mapping of the SMP2 standard[2].

Moreover, after realising the benefits obtained from the Eclipse Modelling Framework it was also acknowledged that the more general Eclipse Rich Client Platform (RCP) could be used to build the entire GUI set of components that then would be combined, using the Eclipse Mechanisms to build an application.

The entire Simsat 4.0 MMI is built around the Eclipse Rich Client Platform and where the MMI functionality is encapsulated in the form of plugins. Combinations of plugins can be used to form and create applications. With this approach it is possible to create an application that provides only a subset of the functionalities and another application that provides another subset of the required functionalities, eventually sharing plugins between the two.

3. The usage of the Eclipse Rich Client Platform

The RCP represents the minimal set of plug-ins needed to build a platform application with a UI. This minimal set encompasses only the two plug-ins org.eclipse.ui and org.eclipse.core.runtime together with their prerequisites.

A RCP program requires an application writer to provide the following support:

- Core application plug-in to initialise the RCP application
- Plug-in entry point for the core application plug-in
- Application Entry point to initialise the application layout and any configuration requirements
- Default Workbench layout to setup the main application window, such as menus and toolbars
- Perspective class that defines the default perspective

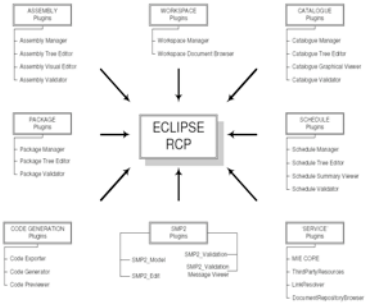

Figure 5 - The usage of the Eclipse RCP in Simsat 4.0

layout relative to the topmost window and how child windows are positioned within the topmost window.

Simsat 4.0 makes use of the above mentioned modularity characteristics to build the set of plugins needed to create the Simsat 4.0 Editors, validators and product generators (source code and build files) as can be observed in Figure 6.

Using the Eclipse Rich Client Platform it is possible to extend applications by simply adding new plugins conforming to the eclipse plugin manifest. It is also possible, due to the eclipse concept of product, to have several products that are just the result of grouping a given set of plugins to form an application.

The plug-in oriented concept allows it to run individual parts of the Simsat 4.0 modeling tools in isolation. For example, the constellation of a stand-alone catalogue editor can be achieved by bundling only the relevant plug-ins to the catalogue editor together with the Eclipse RCP. Contrariwise, the Simsat 4.0 GUI plug-ins can also be bundled together with plug-ins belonging to other applications, e..g. a C++ Debugger, to realize particular configurations.

Simsat 4.0 will provide two main distinct configurations:

- The Model Integration Environment (MIE) is the combination of the different SMP2 modeling plugins that will provide support to the Design, Implementation, Integration, Scheduling phases and partially to the Debugging phases. This configuration and its mechanisms are detailed in this section.
- The Simsat Runtime MMI configuration will be the configuration to use when a simulation is being executed in the runtime kernel and the user needs to connect to it. The Simsat Runtime MMI is detailed in section H - The Simsat 4.0 Runtime MMI.

4. The usage of the Eclipse Modelling Framework

Eclipse provides a set of editors and views that leverage the required effort by jumpstarting the development with a full set of already existing mechanisms for xml based model transformation and editing.

The Eclipse Modelling Framework can be briefly described using two concepts:

- The EMF core framework includes a meta model (Ecore) for describing models and runtime support for the models including change notification, persistence support with default XMI serialization, and a very efficient reflective API for manipulating EMF objects generically.
- The EMF edit mandates that The EMF.Edit framework includes generic reusable classes for building editors for EMF models. It provides content and label provider classes, property source support, and other convenience classes that allow EMF models to be displayed using standard desktop viewers and property sheets. A command framework, including a set of generic command implementation classes for building editors that support fully automatic undo and redo.

Further, the Eclipse Modelling Framework also provides a code generator that is not used by Simsat as it only provides the source code generator for the Java platform whereas the needs of Simsat are for the generation of C++ source code.

The SMP2 standard shares a common approach to the EMF core framework as they both have a core element from which all remaining elements are derived. In the case of EMF this is the EMF EObject class and in SMP2 is the IComponent interface[1]. As a consequence, in Simsat 4.0, the IComponent modeling element extends EMF EObject therefore making all SMP2 elements readable and understandable by the EMF framework. It is then possible to create new SMP2 catalogue, assembly, schedule or package elements by means of the EFactory mechanism provided by the EMF Framework.

As the EMF core framework is used to express the SMP2 modeling elements, the EMF editor is used to provide an editing tool for the graphical creation and manipulation of elements. Graphical Table and property viewers access the SMP2 model using an adapter object called a content provider. This allows to ‘out of the box’ display elements using standard desktop viewers and property sheets also provided by eclipse.

The EMF command domain editing set of interfaces are supported by all SMP2 Elements (Content Providers). This enables any element to be edited or displayed in a tree, property grid and have listeners.

5. The meta model validators

The process of validation *is the* application of a set of rules to a set of components.

In Simsat 4.0, it is necessary to provide validation for different phases of the simulation modelling life cycle, e.g. for the Design, Integration and Scheduling phases it is necessary to apply different validation rules. The validation process can therefore be separated into three high-level tasks: the definition of the rules, the iterative process of applying those rules to each component and the recording of the validation results. Where the first task is clearly different to each of the simulation phases as the different SMP2 Metamodel elements have distinct rules applied to

them the latter two tasks as common to all phases. Simsat 4.0 provides a validation engine implementing the latter two of these three tasks - functionality that is common to any validation process.

The engine plug-in acts as a service (i.e. there are no commands that invoke the engine directly) and simply provides a class library to the high-level *validators*.

A high-level *validator* (such as the catalogue, assembly, schedule or package validator) will use or extend the classes provided by the engine plug-in to perform their specific validation processes.

6. The source code generation

The SMP2 standard currently (version 1.2)[2] only provides a language mapping from the standard to the C++ language and the Simsat runtime kernel is only also able to load C/C++ models. In consequence the native code generator provided by the EMF was not suitable for Simsat as it was only able to produce Java Code. However due to the component based design approach of the eclipse modelling framework it was possible to replace the native Java code generator with a template based C++ generator. The template based tool Freemarker was used for this purpose.

The code generation is implemented in the Code Generator plugin that provides a facility to generate source code from a given element - a catalogue, namespace, model, etc - in a catalogue model.

To process an element in a catalogue the element is first converted to a 'code generation provider', by means of a look-up table that maps each type of catalogue element to the code generation provider for that type (e.g. the map associates the Integer catalogue class to the *IntegerCodeGenerationProvider* class).

Each code generation provider implements a method to generate source code for the type of element it specialises in. It does this by building a table of attributes (e.g. for an integer type: name, min value, max value, etc) and then processing that table using the relevant FreeMarker template file (e.g. "integerdeclaration. ftl"). The template file has access to the values in the attribute table, so allowing variable values to be output (access to the attribute table is provided by the FreeMarker software).

If the catalogue element has child elements (e.g. a namespace has nested namespaces), then the code generation provider will iterate through the child elements and process them in a recursive fashion.

Note: it is not the job of the code generator plug-in to output source code to a specific destination. The output from the code generator is directed to a generic writer interface that other plug-ins (namely the code exporter and code previewer) will be expected to implement with their own output writer that sends the generated source to whichever destination they want. This mechanism will allow for instance, that upon user request the generated source code is either previewed in a display window or saved in the file system.

7. The Graphical viewers

The modelling of catalogues, assemblies, packages and schedules will be done mainly recurring to a tree element where it will be possible to add, remove, edit and link elements. In addition to this, Simsat 4.0 will also provide graphical viewers on which it will be possible to observe graphically the relations between the different elements present in the modelling tree, Figure 1.

To produce the different relationship diagrams the Eclipse Draw2D library is used. This library contains the majority of the required graphics functionality.

The Catalogue Graphical Viewer plug-in provides the user with diagrammatic views of the relationships between catalogue components. Depending of the type of elements being edited, it is possible to observe different types of relationships. The types of relationship diagrams supported are Inheritance, Composition and Aggregation, which are only applicable to components that exhibit such relationships. Due to the overwhelming similarity of all catalogue graphical viewers, the majority of the functionality is implemented in a common base class. The only detail specific to each type of diagram is the creation of the diagram generator and the provision of the actual data to be displayed.

Component Type	Applicable diagram		
	Inheritance	Composition	Aggregation
Model	Yes	Yes	Yes
Class	Yes	No	No
Other	No	No	No

The Schedule Graphical Viewer plug-in provides the user with a graphical timeline view of selected schedule documents where each single event is displayed with accompanying labels and cycle and or count information.

H. The Simsat 4.0 Runtime MMI

Simsat 4.0 will provide an MMI that will allow controlling the execution of a Simsat runtime Kernel, i.e. to start, to monitor and control a simulation and to stop a simulation.

As for the MIE, The SIMSAT MMI is created as a set of Eclipse plugins which can be conbined to create different applications. The Simsat MMI is composed of three main layers:

- The RCP code to setup the application. This includes the menu and toolbars, desktop areas. Any RCP application must include such an application plugin.
- The core classes that wrap the simulation entities. These also contain the CORBA communication between the MMI and the kernel components.
- Services for other MMI plugins.

On top of the basic mechanisms there are several other plugins that contribute views and UI elements to the application, building on the classes exposed by the application plugin. Here's a brief description of the MMI plugins and what they provide:

- *Commander*: Processes user commands and command scripts. This component contains also the views that are specific to the kernel script engine.
- *DataViewer*: Allows display of simulation data. Alphanumeric displays (AND) and graph (trend chart) displays (GRD) will be supported.
- *LogViewer*: Shows the content of the chronological simulation log, either live or 'posthumously'.
- *MessageViewer*: Used to show MMI internally generated messages.
- *ScheduleViewer*: Allows viewing (and loading) of the SIMSAT schedule.
- *SimulationTree*: Used to browse the SIMSAT object hierarchy, showing callable functions and data of the objects in the simulation. It also contains a search view to help the user navigate.
- *StatusViewer*: Displays the status items applicable to the simulator as a whole.
- *Smp2EventsViewer*: Displays all events in the system, with a log of fired events and the registered publishers/listeners.
- *ScheduleAnalyzer*: Displays the results of an instrumented simulation run in a graphical way.
- *RecorderViewer*: Configures recorder instances that save data to file during a simulation.
- *SMIWizard*: Creates empty project template for SMI component.
- *RegistryBrowser*: Displays the list of available simulations and components in the Registry

One thing that is to be noted and is of paramount importance is that users of Simsat 4.0 can freely create their own plugins and simply add them to the Simsat MMI in order to execute them.

Simsat 4.0 foresees that different simulation have different viewing needs and provides a mechanism to load a user defined MMI containing a set of plugins configured in a perspective when loading a given simulation kernel.

I. The Simsat 4.0 implementation of the SMP2 Runtime Specification

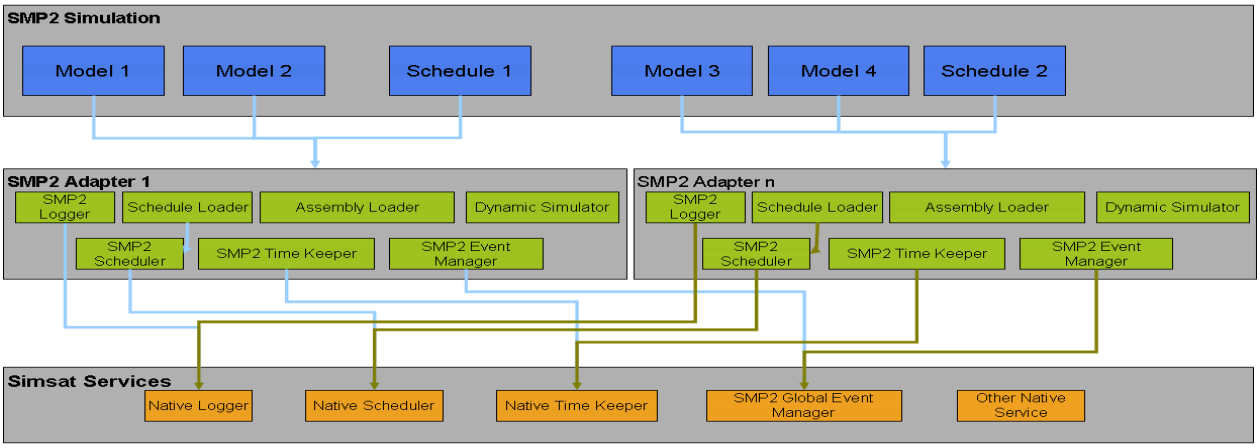

Figure 6 - The Simsat 4.0 Implementation of the SMP2 Runtime Specification

The Simsat Kernel was designed since its initial version on the Windows platform with the concepts of simulation services and components in mind by recurring to the Microsoft COM components paradigm. During the migration to the Linux platform in version 3.0 it was decided to keep the same componentised approach but

replacing the COM components with equivalent CORBA components as the first is not natively available in the Linux platform. For the Simsat 4.0 development the discussion regarding whether the Simsat Kernel should or not natively support the SMP2 interfaces and services arose.

It was decided that, since Simsat must also support the Simulation Model Portability (SMP) standard (predecessor of SMP2, but component unaware), the native Simsat services and component approach would not be changed and that instead SMP2 would be supported by recurring to the Adapter Pattern[5] a this would also protect the Simsat simulation kernel from being dependent on one Simulation Standard only therefore leaving the possibility of hosting other simulation standards.

The Adapter pattern basically converts the interface of one component into another interface clients expect, therefore letting components supporting different interfaces work together. This is the case for the SMP2 services that implement the interfaces mandated by the standard but delegate the actual work to the corresponding native Simsat service as can be depicted in Figure 6, where the SMP2 Logger, SMP2 Scheduler and SMP2 Time Keeper just act as connecting routers between the requests performed by the SMP2 Models and the Simsat native Service. Although not depicted in Figure 6, the same native Simsat services that provide the functionality needed by the SMP2 Services are also used by the old SMP standard adapter.

There is however one SMP2 service that is actually implemented outside the SMP2 Adapter and is shown in Figure 6, as a native Simsat service as there is the need to allow the cooperation, via events sinks and source, between models residing in different adapters. For that to be possible it is necessary to have the service residing outside a specific adapter, that is, having a event manager that is knowledgeable about the SMP2 event mechanism and that knows about all the SMP2 Adapters residing in the Simulation kernel.

IV. Future work

Concerning the evolution of the Simulation Model Portability Standard 2, it is foreseen its promotion to an ECSS Standard (ECSS E-40 Part 7).

As for the ESOC Simulation infrastructure, and regarding SMP2, it is foreseen to:

- Upgrade it to support future SMP2 standard updates (ECSS)
- Support of distributed and dynamic simulation model execution.
- Define a reference spacecraft simulator architecture using SMP-2
- Contribute to the development of generic/re-usable SMP-2 Model libraries

References

[1]SMP 2.0 Component Model, ESA, v1.2, 2005-10-28

[2]SMP 2.0 C++ Mapping, ESA, v1.2, 2005-10-28

[3]SMP 2.0 Handbook, ESA, v1.2, 2005-10-28

[4]SMP 2.0 Metamodel Specification, ESA, v1.2, 2005-10-28

[5]Gamma, Helm, Johnson and Vlissides, Design Patterns: Elements of Reusable Object-Orientated Software, Addison-Wesley 1995, ISBN: 0-201-63361-2