Simubox's SMP2 Code Generator under *Enterprise Architect* 9

Prepared by Leandro Toss Hoffmann

2014

Contents

1. Introduction

This document describes the Simubox's code generation tool for SMP2 compliant components as part of the thesis "THE ROLE OF COMPUTATIONAL STEERING IN SPACE ENGINEERING ACTIVITIES ASSISTED BY SYSTEMS MODELLING AND SIMULATION" developed by Leandro Toss Hoffmann in 2014, under supervision of Prof. Leonel Fernando Perondi, Ph.D. at Brazilian Institute for Space Research (INPE).

The tool is implemented as an Enterprise Architect 9 add-in and currently supports the following features:

- C++ header file generation for Models derived from `Smp::Management::IManagedModel`;
- Corresponding C++ implementation file;
- Construction of projects for building the model libraries (.dll); and
- SMP2 Assembly artefact production.

These features can support a complete chain of simulation models development, based on a component modelling process in UML. At the end, the auto-generated code (without behavioural code) and the assemblies are ready to be run in a SMP2 compliant simulation infrastructure. Still, in the future, the extension of the Add-in will support also SMP2 catalogue generation and a broader number of SMP interfaces.

2. Installation

The *Simubox's* SMP2 code generator runs under the Enterprise Architect (EA) environment. A SMP2 code template is provided to the Code Template Framework, used during forward engineering of definition files (i.e. header files). A *Simubox* add-in must be installed in order to extend the base templates and to allow the production of the SMP2 artefacts and model libraries.

The Add-in items to be installed comprise on:

- **`SMP2CodeGen.dll`**: implementation of add-in to be loaded by EA;

- **`SMP2_Template.xml`**: contains the SMP2 Code Template and the predefined reference data (i.e. stereotypes and tagged values) to be used with the UML diagrams.

2.1. Add-in installation

The add-in is compiled as a .NET DLL (see SMP2CodeGen.csproj compatible with Microsoft Visual Studio 2010) and its file must be registered as a COM assembly. The register is done using the *RegAsm* command:

```
C:\WINDOWS\Microsoft.NET\Framework\v2.0.50727\RegAsm.exe
"path/to/SMP2CodeGen.dll " /codebase
```

The second step is the registration of the add-in in the Windows Register. Insert the entry `"SMP2CodeGen.SMP2CodeGenClass"` under the `[HKEY_CURRENT_USER\Software\Sparx Systems\EAAddins\SMP2CodeGen]` folder.

Optionally, use the deployed `RegistryEntry.reg` file to automatically edit the Windows Register.

Note that both steps must be executed with administrator privileges.

After the EA initialisation, the new menu **Simubox** will appear under Add-in menu.

2.2. Reference Data

For each Enterprise Architect project that is going to make use of the SMP2 code generation tool, the reference data must be imported.

Open the **Import Reference Data** Dialog from the menu:

Project / Model import/export / Import reference Data.

Select `SMP2_Template.xml` file and import all Datasets: SMP2_Code_Template, Stereotypes, and Property types (for Tagged values).

To make the SMP2 template language available, create a new entry in the Programming Languages Datatypes: **Settings** / **Code datatype** / **Add Product**. Use the name SMP2 and save any new datatype (e.g. Smp::Int32).

3. Components Modelling

UML components would be a good metaphor to represent simulation models, but since the Enterprise Architect engine facilitates the code generation for elements of class type, the class diagrams was adopted.

For each simulation model, create a new class. Excepted by the *interfaces*, always define classes stereotypes as *SMP2Model*.

Make sure to set the default language of the element as *SMP2*.

3.1. Fields

There are two different ways for representing Model's fields. Based upon the fields type, a proper modelling must be taken. If the field represents a regular model parameter, then it is modelled as a class *attribute*. However, if the field is an input/output field, then a *port* element should be used, as illustrated in the next Figure.

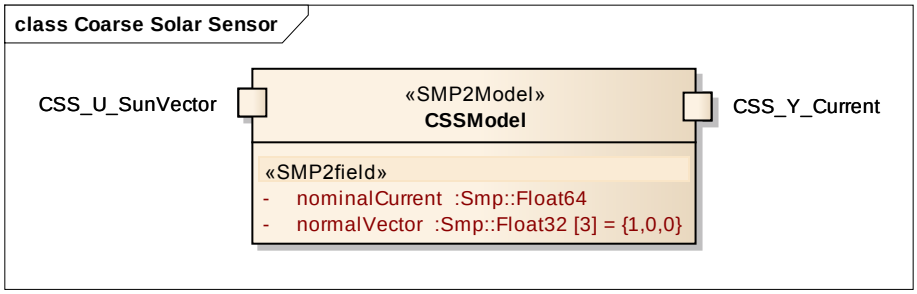

Figure 1. Simulation model representation and its fields.

The different aspects of fields modelling are detailed next.

Simple fields

Simple fields are defined as a regular class attribute (see *CSSModel*::*nominalCurrent*). Additionally, set the following:

- **Stereotype**: SMP2field
- **Type**: any SMP2 type (e.g. Smp::Int32)
- **Initial Value** (optional): value, accordingly to Type.
- Tagged values:
    - **state** (optional) = {True, False}, default = True;
    - **viewKind** (optional) = {none,debug, expert, all}, default = none;

For fields, all the tagged values are optional. By default, the fields are persistent and all operations are published with View Kind option equals to None (i.e. not be made visible to the user). If a different configuration is desired, define the corresponding tagged value.

Array fields

Array fields are also defined as a regular class attribute (see *CSSModel*::*normalVector*).

The same configuration of *Simple Fields* applies to *Array Fields*, but in addition the dimension information must also be filled:

- **Upper bound** [Detail/Multiplicity]: array dimension (e.g. 3)
- **Attributed is a collection** [Detail/Collection]: checked.

Input/Output fields

Input/Output fields are special attributes of classes, since they can be linked with other fields. In this way, they are modelled as *Port* elements.

To add a *Port* to the simulation model, right click the *class* element and select **Embbeded Elements** / **Add Port**.

This element doesn't support the same types of properties as the class attributes, so the *Type* and *Value* (optional) should be inserted as tagged values. Additionally, set a tag ***output=true*** or ***input=true***, accordingly to field's direction.

3.2. Exposed interfaces

When a simulation model implements a certain interface it can directly derive the base class. However, to facilitate the dependence definition among models, the implemented interfaces shall be described as exposed interfaces of classes. For each exposed interface, add an embedded port and then an embedded provided interface.

The port must have the *SMP2interface* stereotype.

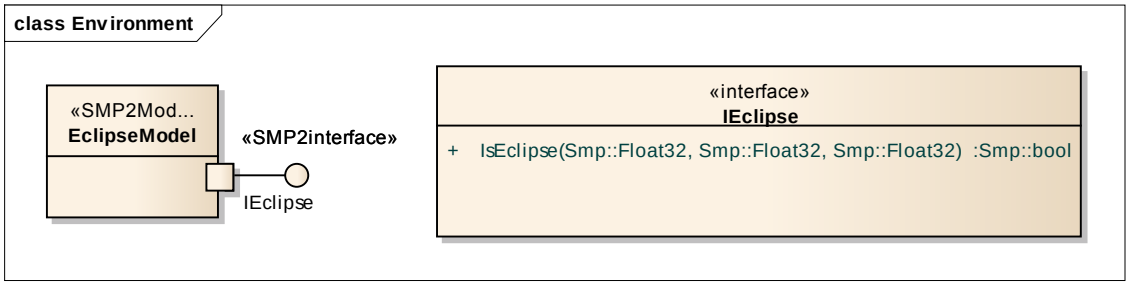

Figure 2. *IEclipse* is a provided interface by *EclipseModel*.

3.3. References

References are represented as requested interfaces and, analogy to the exposed interfaces, they are modelled as embedded interface of a port belonging to the class.

The port's stereotype must be *SMP2interface* and its name will be used as the container of the model (*Smp::Reference*). The name of the interface is the name of the referenced interface.

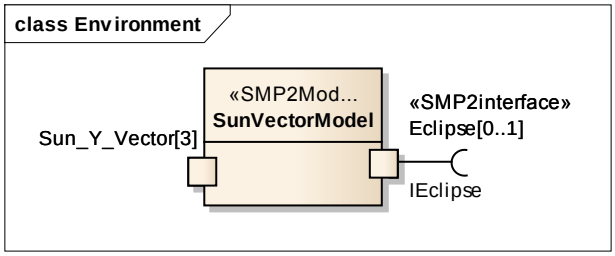

Figure 3. *SunVectorModel* requires a reference to the *IEclipseModel* interface.

3.4. Containers

Containers are defined as composite relations. The link's properties carry the name of the container (i.e. Source Role), the multiplicity and must be *SMP2container* stereotyped.

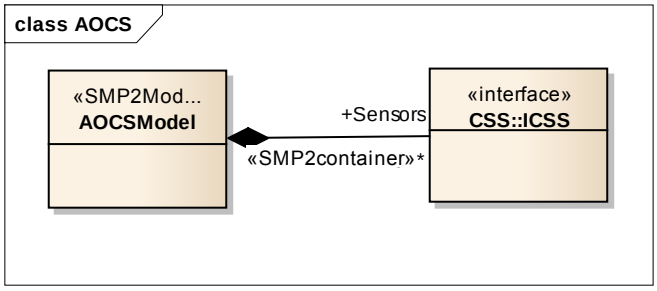

Figure 4. The *AOCSModel* is a composition of *Coarse Solar Sensors*, whose are held in the *Sensors* container.

3.5. Operations

Operations are same as regular class methods.

By default, all operations are published with *View Kind* option equals to *None*, which means that it will not be not made visible to the user. If a different configuration is desired, define a tagged value: viewKind = {debug, expert, all}.

The *direction* attribute of operation's parameters should be specified to achieve its correct publication (e.g. In, InOut, etc..).

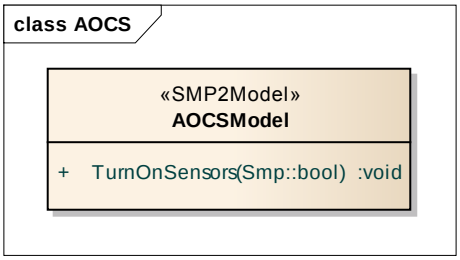

Figure 5. The *AOCSModel* has an operation to Turn On/Off all the sensors.

3.6. Entry Points

Entry Points are the same as a regular class method, but a void/void function with *SMP2entryPoint* stereotype. They can have tagged values to associate input and output fields.

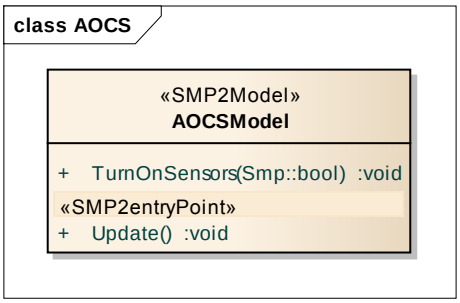

Figure 6. The *AOCSModel* has an Entry Point *Update*().

3.7. Events

TBD.

4. Assembly Definition

The assembly defines the SMP2 link among the model instances. These connections are defined in an object diagram, where the model instances are placed as object of classes.

When an object is created its embedded elements are not shown by default. In order to show then, right click the element and choose **Embedded elements…** / **embedded elements…** / **Show Inherited**.

4.1. Field values for the instance

The default field values for a given model's instance can be defined as a tagged value. The name of the value must be the same as the field. This feature doesn't apply to output and input fields.

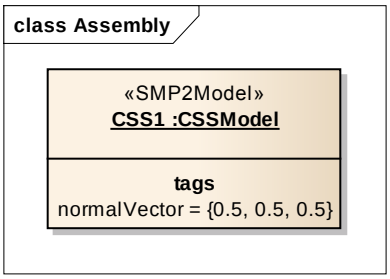

Figure 7. Definition of *normalVector* for the instance CSS1 as a tagged value of object.

4.2. Composition

The model hierarchy is defined by the composition connection with *SMP2composite* stereotype and the name of the parent's container.

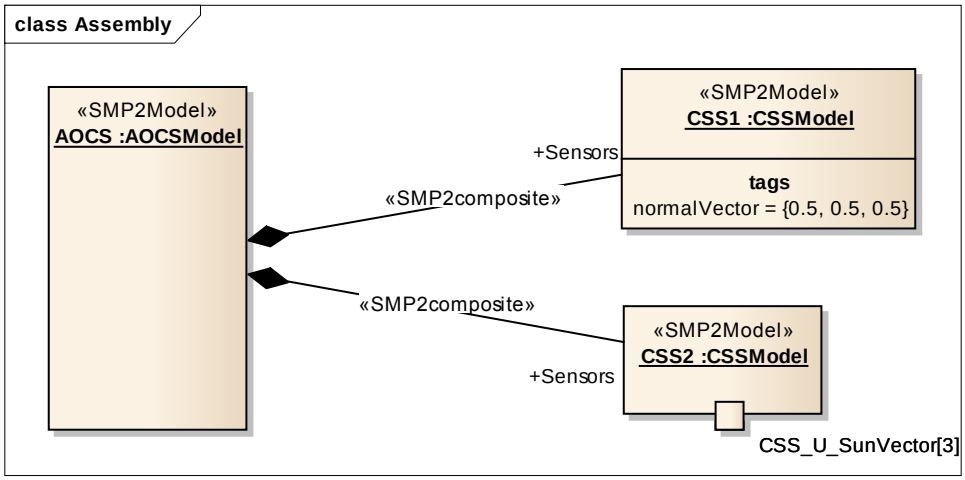

Figure 8. The composition of *AOCS* instance and its two sensors CSS1 and CSS2.

4.3. Field links

The field links are defined as direct dependencies between the output fields and input fields. Strictly speaking, the input field depends on output field, but in order to better visualise the data flow, this association is meant to be inverted.

The type of the fields being linked is not checked. A type mismatch will result in error during the simulation building.

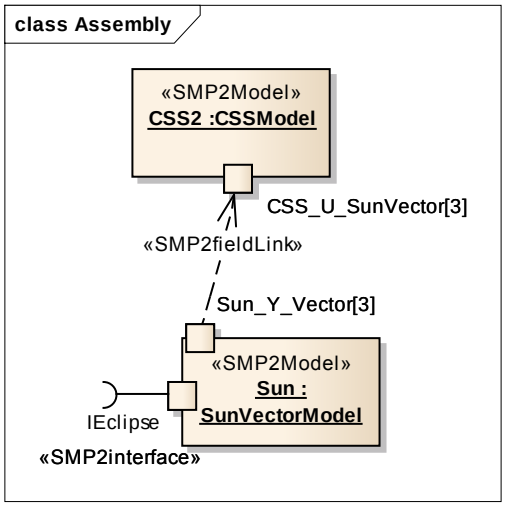

Figure 9. The output field *Sun_Y_Vector::SunVectorModel* is linked to the input field *CSS_U_SunVector::CSS2*.

4.4. Interface Links

References as linked using the assembly connector with *SMP2interfaceLink* stereotype.

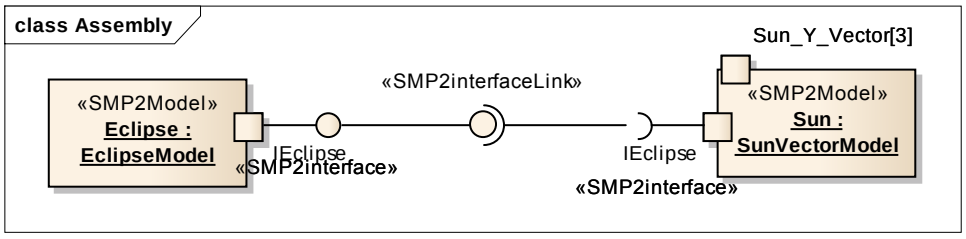

Figure 10. A interface link among *Sun* model (consumer) and *Eclipse* (provider).

5. Code Generation

This section describes the step to generate the source code of simulation models. All the commands are accessed from the Add-Ins context menu.

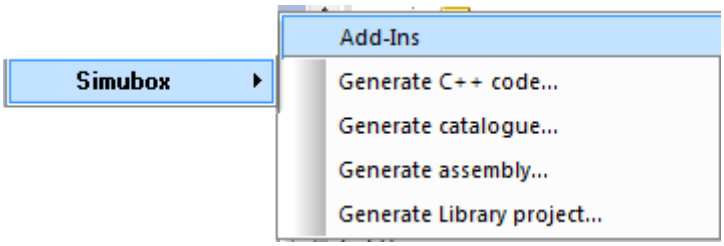

Figure 11. *Simbux* Add-in commands.

5.1. C++ Class definition file (.h)

Using the same code generation command from Enterprise Architect and choosing the SMP2 language.

5.2. C++ Class implementation file (.cpp)

Right click the package or the class in the tree view and choose **Add-in** / **Simubox** / **Generate C++ code...**

5.3. SMP2 catalogue file (.cat)

Right click the package or the class in the tree view and choose **Add-in** / **Simubox** / **Generate SMP2 Catalogue...**

TBD

5.4. Project for implementing the model library and SMP2 Package

Right click a package in the tree view and choose **Add-in** / **Simubox** / **Generate Lib Code...**

Input the target path for the project and the *Simubox* root path (i.e. the location of Simubox project).

The following files will be generated for a given *Package*:

- ***Package*.pro**: *Qt* project definition;
- ***Package*_Global.h**: Defines for DLL export or import directives;
- ***Package*Lib.h**: Library's function definitions;
- ***Package*Lib.cpp**: Library's function implementations; and
- ***Package*Lib.pck**: SMP2 package for the library.

6. Next version improvements

6.1. Fix current restrictions

1. Events are not modelled.
2. Array fields must be one-dimensional.
3. Structures, properties are not generated.
4. Class doesn't derive from *Smp*::*IDynamicInvocation*.
5. The Model package generation doesn't include the *type registration* capability.
6. The generated code for implemented interfaces only works for interfaces in the same package.

7. The model must be in a packaged path with level greater than 1 in order to properly generate de namespace.
8. Assembly always has *rootModel* as Root model and its reference to catalogue is undefined.
9. The input/output types of fields being linked in Assembly are not checked. The verification could avoid error during the simulation building.
10. There is not limit checking for the length of field names. Some field links IDs are created with more than 32 characters, which is not allowed by the SMP2 standard.
11. It is not possible to define the generation order of the instances in the assembly.

6.2. Additional features

1. Assembly validation against SMP2 catalogue.
2. Schedule definition.

Appendix A. Description of Code Generator classes

SMP2CodeGenClass.cs: entry point of the add-in. It defines the menu entries and the functions invoke by the Code Template Framework's macros, used for definition code (.h) generation.

CppGenClass.cs: Produces the implementation source code (.cpp) of SMP2 C++ models.

LibGenClass.cs: Produces the files for building a component library (DLL) and a SMP2 Package .

CodeGenDialog.cs: generic dialog for file system browsing.

UtilClass.cs: common functions.

The component also includes an Enterprise Environment project with the examples of this manual and additional models used in the verification of the add-in.