SMP2 Developments in EuroSim

Robert de Vries[(1)], Jeroen Moelands[(2)]

[(1)]Dutch Space B.V.
Mendelweg 30
2333 CS Leiden
The Netherlands
r.de.vries@dutchspace.nl

[(2)] National Aerospace Laboratory NLR
Anthony Fokkerweg 2
1059 CM Amsterdam
The Netherlands
moelands@nlr.nl

INTRODUCTION

The Simulation Model Portability (SMP) standard is ESA's standard for the interface between simulation models and simulator infrastructures. The purpose of the standard is to promote portability of models among different simulation environments and operating systems, and to promote the reuse of simulation models. SMP1 [1], the first version of the standard, is still supported by EuroSim.

SMP2 [2] is the successor of SMP1. SMP2 is a complete revision of the standard, adopting state-of-the-art techniques like component-based design and model driven architecture, and has a much wider impact on model and simulator development than its predecessor. The way of working with this standard and its complexity demand tools for specification, development, and integration of SMP2 models. EuroSim incorporates a set of tools to accomplish these tasks.

A further iteration of the SMP standard is currently in the final draft stage as European Cooperation for Space Standardization (ECSS) standard E-40-07. The EuroSim consortium has been involved in the process of specification of the draft ECSS standard, like it was in SMP1 and SMP2 specification.

The simulation modelling platform (SMP) is a new standard invented from scratch. It needs to be tested in the field on a real test bench. The EuroSim simulation infrastructure needs to be updated to comply with this standard. It can then be used to verify that the new standard is suitable for the most demanding applications.

EUROSIM

EuroSim is a proven platform that contains a set of integrated tools to support all phases of a real-time simulation project [3]. Model code is imported into the EuroSim environment using the Model Editor. Model code is scheduled with the Schedule Editor. Simulators are executed using the Simulation Controller. These are just some key examples of the dozens of tools that are available. Commonly used functionality of EuroSim is made available through user-friendly graphical user interfaces. Most of EuroSim's customers start modeling in MATLAB/Simulink. NLR's tool MOSAIC (Model Oriented Software Automatic Interface Converter) [4] allows users to automatically transfer MATLAB/Simulink models to the EuroSim environment.

EUROSIM SMP2 IMPLEMENTATION

Generally, an SMP2 simulator is specified using the following types of XML files defined in the *SMP2 Metamodel*. The *Catalogue* offers a variety of mechanisms to specify all kinds of data types in an object-oriented fashion, including simulation models. A *Package* file represents, at an abstract level, a set of specific implementations of simulation models that are specified in Catalogues. It is possible to develop multiple implementations of the same simulation model using different Packages. The *Assembly* is a file type that specifies a simulator in the form of a hierarchy of model instances, referring to their specification (in a Catalogue) and implementation (in a Package). It also contains links to define their interaction with the other instances. Finally, the *Schedule* file offers a way to specify scheduling of model instances in an assembly. SMP2's *C++ Mapping* prescribes a mapping from type specifications in a Catalogue to equivalent specifications in their language of implementation (C++). Finally, the interface between the SMP2 models and the SMP2-compliant simulation environment is defined by the *SMP2 Component Model*.

The standard defines an API and a toolbox of file types and ways to use these, offering different possible ways to obtain an SMP2 simulator, some suitable for trivial projects only. EuroSim applies the standard in a way that is suitable for complex simulation projects. Figure 1 shows an overview of the steps that need to be taken to obtain an executable EuroSim simulator from the SMP2 files. This is further explained below.

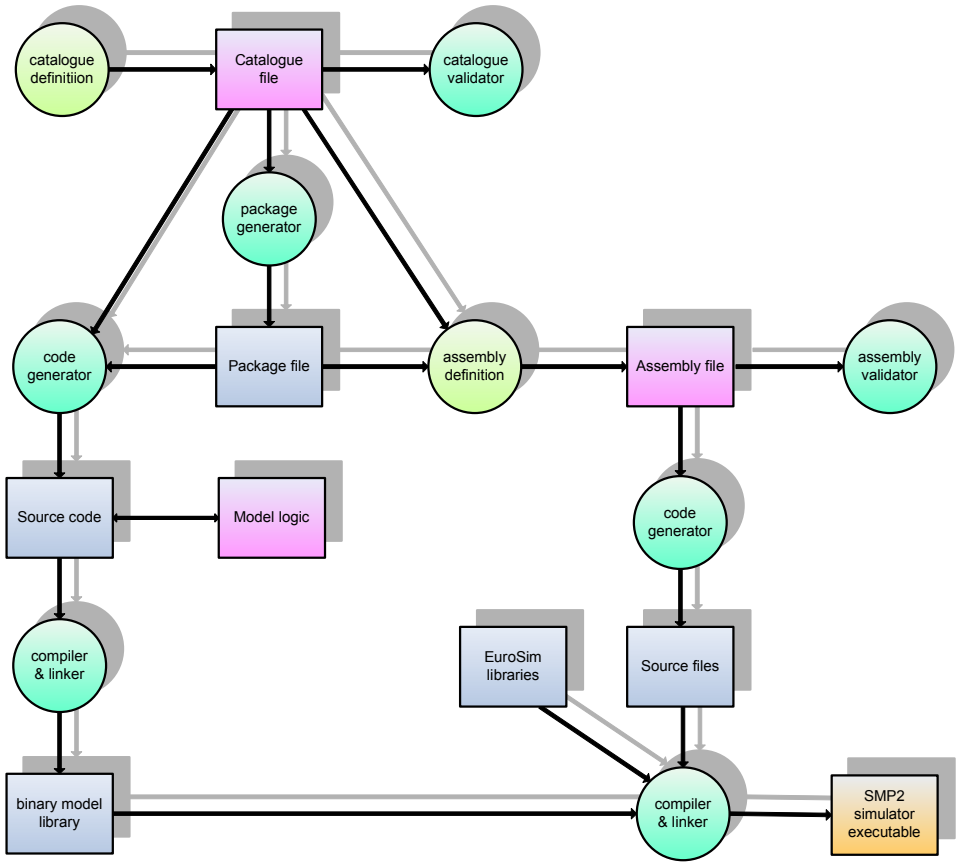

Figure 1: Overview of the steps to obtain an SMP2 simulator in EuroSim

SMP2 MODEL DEVELOPMENT

SMP2 simulator development starts with the definition of one or more Catalogues (see Figure 1). Catalogues can be obtained from multiple possible sources:

- Derived from models developed in MATLAB/Simulink, using the MOSAIC tool.
- Specified using an XML editor. EuroSim offers an SMP2 validation tool to verify correctness of a catalogue specified using an XML editor.
- Specified using a third-party Catalogue editor. As part of third party simulation environments, tools are available to specify Catalogues. New ones are under development.

EuroSim currently supports most Catalogue features and aims at implementing a user-friendly graphical Catalogue Editor. A limitation of some SMP2 tools is that they graphically represent a Catalogue as something that is only a small step from an XML file. Operating them requires intimate knowledge of the SMP2 specification.

Subsequently, a Package file must be generated for every Catalogue. In EuroSim, Packages are required for code generation and for working with Assemblies. The normal use case would be to provide a single implementation for all classes and models in a Catalogue, and to specify these in a single Package. This use case is covered by EuroSim's package generator (see Figure 1).

SMP2 CODE GENERATION

The next step in SMP2 simulator development is code generation (see Figure 1). At this point, integration with EuroSim's simulator build process starts. Using the Model Editor, the user can import the Catalogues and Package files in EuroSim's model tree and easily generate all required source code.

From the specified catalogues, C++ code must be produced according to the C++ Mapping. As this is a very complex and tedious task, the user should be supported as much as possible when using code generation tools. SMP2 only defines generation of pieces of C++ specifications (snippets of so-called header files). EuroSim offers a code generation tool that not only generates these specification code snippets, but generates as much logic as possible. This leaves to the user only the task of implementing the actual model logic, which is not specified in the Catalogue and therefore by definition cannot be generated. A possible alternative solution for this is provided in [3]. All generated code can be compiled out-of-the-box using the generated makefiles. As shown in Figure 2, the following types of file are generated:

- A complete header file for each type specified in the Catalogue.
- A so-called boilerplate file which contains a predefined implementation of all SMP2-related functions required for a simulation model or other SMP2 type in a Catalogue, like initialisation and model publication code. Boilerplate files are generated from Catalogue files. The predefined implementation of these functions is complete and requires no further work from the model developer.
- An implementation file is generated for each type in a Package that needs addition model logic specified by the user. Templates for the methods requiring additional model development are generated in this file. Note, that implementation files are not generated from Catalogues directly, but from Packages. In the case of different implementations of the same type, there will be multiple Packages specifying the different implementations, allowing the generation of different implementation files.
- A makefile is generated that aids the compilation of all boilerplate code generated from a Catalogue into a binary model boilerplate library containing the implementation-independent functionality that is fully generated by EuroSim.
- A makefile is generated that aids the compilation of all implementation code generated from a Package into a binary library containing the implementation-dependent functionality that is added by the user, and references to the required binary model boilerplate libraries.
- Files containing library initialisation and finalisation code are generated from a Package, supporting both static and dynamic libraries.

The functionality EuroSim offers for code generation from Catalogues and Packages automates as much as possible of the development of source code. Moreover, we consider such a feature a requirement for any adequate tool offering an SMP2 model development environment.

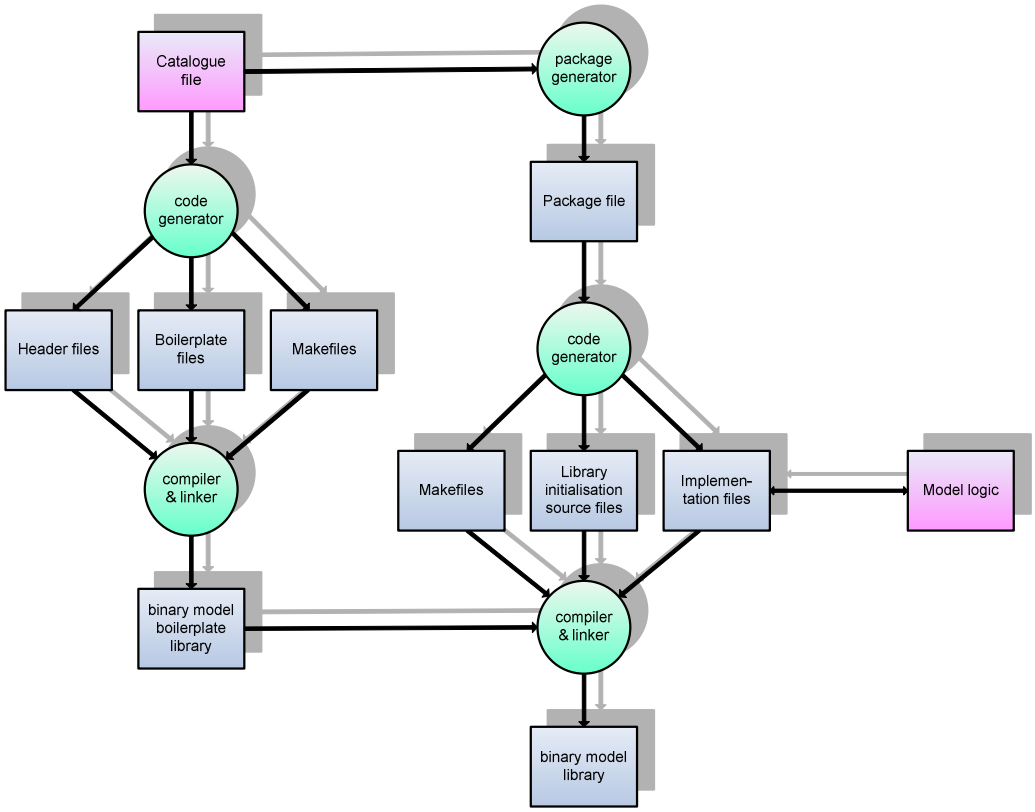

Figure 2: Code generation steps from Catalogue and model logic to binary model library

SMP2 SIMULATOR DEVELOPMENT

After implementing the simulation models, an Assembly must be constructed. This again is an XML file that can be created with XML editors, third party Assembly editors, etc. EuroSim offers an Assembly validation tool to be able to

validate the file before importing it. The assembly file refers to existing Catalogue and Package files and must be imported in EuroSim using the Model Editor. From the Assembly, the Model Editor will automatically generate “glue” code. This code constructs at run-time the specified SMP2 simulator from instantiated models and integrates it with the EuroSim run-time environment. No additional user logic needs to be specified here. EuroSim builds all generated and user-defined code into an executable simulator that, together with a real-time schedule, can be run by the simulation controller tool in the usual way.

EuroSim’s native scheduling mechanism provides more features than the SMP2 Schedule and offers hard real-time multiprocessor performance that SMP’s Schedule doesn’t. SMP2 model instances can be scheduled just like native EuroSim models, using EuroSim’s Schedule Editor. EuroSim features a tool that converts an SMP2 schedule to the native EuroSim schedule format. Using EuroSim schedules, SMP2 users can take advantage of EuroSim hard real-time capabilities. To make SMP2 more suitable for hard real-time performance, more features should be added to the SMP2 Schedule in a future version.

CODE MERGING

Incremental and iterative software development has since long replaced the waterfall model as a software development method of preference. This fact is reflected in a crucial feature of the SMP2 development environment, code merging. Code merging is the integration of user-defined logic that is located in an existing generated source file, with a newly generated version of that source file. It is a common use case in incremental and iterative software design: a first version of the Catalogue is created, code is generated and model logic added. Subsequently, a change is made in the Catalogue and a new version of the source code must be generated, transferring the existing implementation to it. See Figure 3. EuroSim’s code generator automatically analyzes existing versions of generated source files and preserves the existing model logic by moving snippets of code at designated locations in the source file to equivalent locations in the newly generated file.

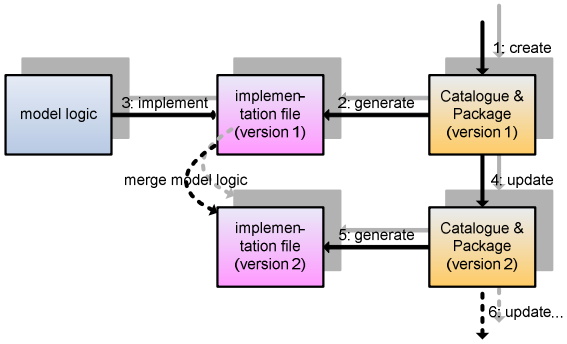

Figure 3: The important role of code merging in incremental and iterative development

MODEL PORTABILITY AND REUSE IN PRACTICE

The purpose of SMP is to promote portability of models among different simulation environments and operating systems, and to promote reuse of simulation models.

Regarding the portability of SMP2 files, the use of third-party editing tools to specify Catalogues and Assemblies is an example of portability between different simulation environments. EuroSim supports almost all Catalogue features. In the rare case when unsupported features are used in a Catalogue created in a different simulation environment, some effort will be involved in porting them to EuroSim.

The SMP2 specification leaves open some important aspects of code generation and subsequent building of model libraries, like the mapping of snippets of code to complete source files, the naming and hierarchical organisation of the source files, and the contents and naming of the resulting model libraries. In the area of source code generation, EuroSim’s code generator offers functionality that reflects good design choices made in some third-party SMP2 code generators. This improves the chances of easy portability of model code between simulator infrastructures. EuroSim generates platform-independent and operating system independent source code from Catalogues and Packages. Glue code generated from Assemblies is EuroSim-specific. This code will need to be generated for Assemblies that are ported to EuroSim.

A straightforward implementation of code merging is by supplying generated code with unique markers that indicate the beginning and end of locations where user-defined model logic is expected to be added. SMP2 does not prescribe any code merging markers in its C++ mapping, so these will not be present in code generated by a different SMP2

development environment. This somewhat limits easy development on models that are ported from different simulation platforms.

EuroSim is available on multiple operating systems. SMP2 files and model source code developed with it can be used on any of these operating systems.

APPLICABILITY OF SMP2 STANDARD

The goal of the SMP2 standard is not to cover all use cases of every simulator developer. Its main goal is to encourage model reuse by defining a common model interface standard. This leaves room for special purpose applications such as hard real-time simulators.
Figure 4 illustrates the application areas where the SMP2 standard is sufficient and where it lacks certain features.

Simulator integration		**Additional EuroSim scheduling features**
Model development		
Simulation infrastructure interface		**Additional EuroSim infrastructure interfaces**
	Non-real-time (soft real-time, as fast as possible, etc.)	Real-time (hard real-time)

Figure 4 SMP2 application area matrix

The simulator integration covers assembly and schedule definition. The following assembly methods are supported:

- Interface based
- Event based
- Data flow based

The interface based integration method uses direct calls between models. This means that model A may call directly one or more methods of model B. There are two problems with this integration method in the case of multi-processor real-time simulators. First there is the issue of accountability of execution time. When the scheduler calls an entry point of model A that calls one or more methods of model B, there is no easy way (except profiling) to tell where the execution time is spent. If model A is provided by contractor A and model B is provided by contractor B it is now hard to find the guilty party. Secondly if model A is executing concurrently with model B it is possible that incorrect or inconsistent data is used. Of course it is possible that the developer adds synchronization primitives (such as mutexes) in the code to prevent this. However this introduces the problem of non-deterministic execution times in case of a blocking mutex. It is therefore recommended not to use this kind of integration for real-time simulations unless the simulation runs single threaded.
The event based integration method suffers from similar problems. It uses the concept of event sources and event sinks. Whenever an event source is triggered, it will call the methods of all connected event sinks. As event sinks are most often in other models, this means that entry points of other models are triggered while executing the entry point with the event source. This gives rise to the same problems as with interface based integration.
The data flow based integration does not suffer these problems as the data flows are triggered after executing the entry point producing the data and before the entry point consuming the data.

The SMP2 schedule definition file does not have any facilities that allow you to specify execution over multiple CPU's or to schedule tasks triggered from hardware interrupts. There are virtually no facilities that allow you to specify a schedule that can be used for a hardware in the loop simulator. The richness of features found in the EuroSim schedule editor is not available in SMP2.
The following features in EuroSim are not present in SMP2:

- events triggered from interrupts, signals, semaphores etc.
- low-latency asynchronous event handling
- graph based scheduling

- task priorities
- mutexes
- frequency dividers/multipliers
- non-real-time tasks in real-time context

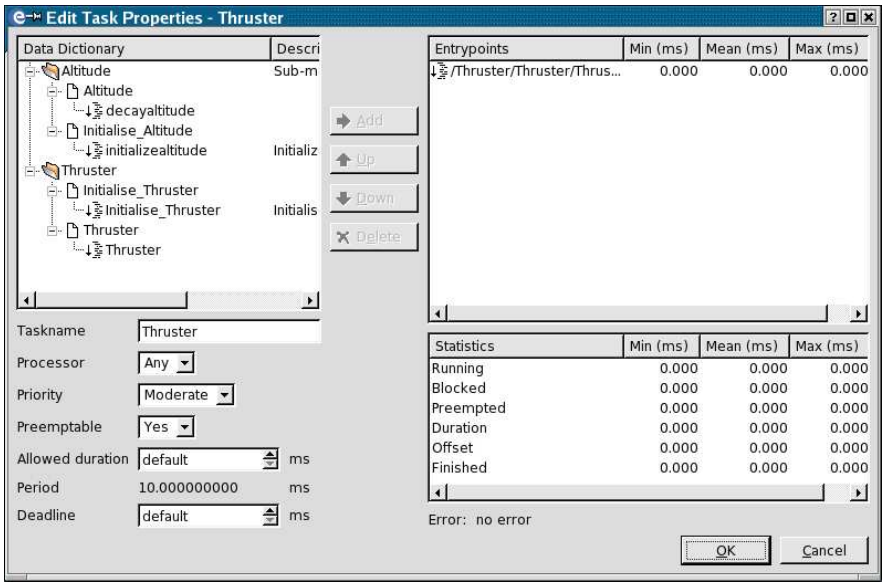

Figure 5 EuroSim task properties dialog box

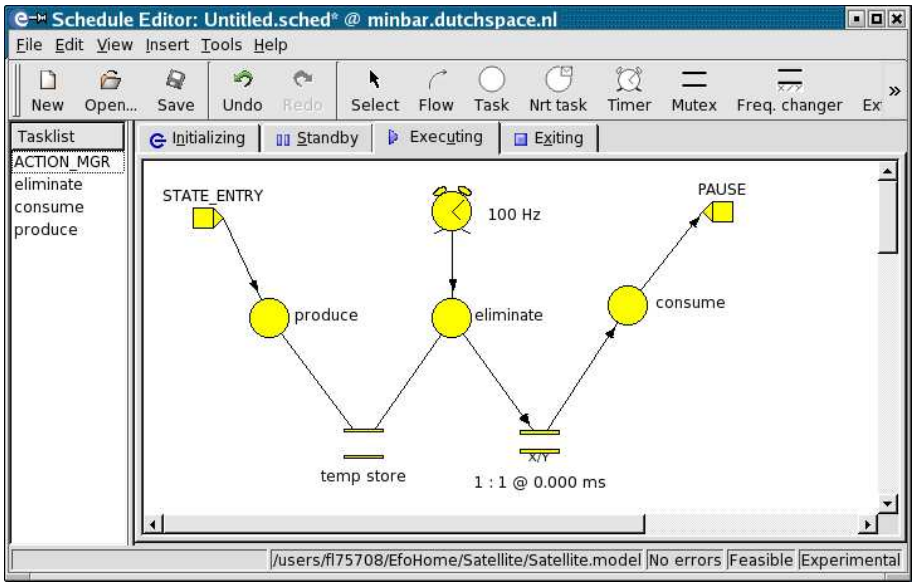

Figure 6 EuroSim schedule editor

Figure 5 and Figure 6 show screenshots of the EuroSim schedule editor. This tool makes it very easy to define and tune real-time schedules.

The SMP2 standard also defines a number of interfaces with the simulation infrastructure. The interfaces are necessarily limited in number. It comes therefore as no surprise that a number of interfaces needed for real-time simulators are not included. The following interfaces are missing:

- dynamic memory allocation (delete, new, malloc, etc.)
- file I/O
- network I/O

These interfaces provide allow real-time models to perform these functions from the real-time domain without taking an nondeterministic amount of time. If these functions are called directly, the operating system calls are performed from the real-time domain. Blocking or other sources of latencies introduced by the operating system kernel are directly influencing the execution time of the real-time task. This will lead to real-time errors and therefore a failed test.

Another important aspect is the deterministic execution times of the infrastructure itself and the models. For SMP2 the deterministic execution time of the generated code is also important. The user has no influence on the generated code, so the code must be of the highest quality and suitable for all applications. This is true for most of the generated code, but there is one item where this is not the case. The code generated for the dynamic invocation interface performs sequential checks for the name of the operation. This generated code does not scale well when there are many operations, properties and entry points. There is no direct method to get and set the values of properties from the simulator infrastructure other than to do a dynamic invocation of the getter and setter method. That this is not very efficient is another matter. It is worse that the execution time of a property depends linearly on the number of properties times 2 plus the number of operations plus the execution time of the operation function (which may be relatively short, especially for a property setter or getter).

```
/// Dynamic invocation of operation.
/// Dynamically invokes an operation using a request
/// that has been created by CreateRequest() and filled
/// with parameter values by the caller.
/// @param   request Request object.
/// @remarks On successful invocation, the return value of the
///          operation can be retrieved via GetReturnValue().
void Examples::Sample::Invoke(Smp::IRequest* request) throw (
    Smp::IDynamicInvocation::InvalidOperationName,
    Smp::IDynamicInvocation::InvalidParameterCount,
    Smp::IDynamicInvocation::InvalidParameterType)
{
    if (request)
    {
        // MyEntryPoint
        if (strcmp(request->GetOperationName(), EntryPointName) == 0)
        {
            if (request->GetParameterCount() == 0)
            {
                _MyEntryPoint();
            }
            else
            {
                throw Smp::IDynamicInvocation::InvalidParameterCount(
                    request->GetOperationName(), 0, request->GetParameterCount());
            }
        }
        // Counter Property getter
        else if (strcmp(request->GetOperationName(), "get_Counter") == 0)
        {
            if (request->GetParameterCount() == 0)
            {
                Smp::Mdk::AnySimple returnValue;
                returnValue.Set(get_Counter());
                request->SetReturnValue(returnValue);
            }
            else
            {
                throw Smp::IDynamicInvocation::InvalidParameterCount(
                    request->GetOperationName(), 0, request->GetParameterCount());
            }
        }
        // Counter Property setter
        else if (strcmp(request->GetOperationName(), "set_Counter") == 0)
        {
            if (request->GetParameterCount() == 1)
            {
                set_Counter(request->GetParameterValue(0).value.int32Value);
            }
            else
            {
                throw Smp::IDynamicInvocation::InvalidParameterCount(
                    request->GetOperationName(), 1, request->GetParameterCount());
            }
        }
        else
        {
            throw Smp::IDynamicInvocation::InvalidOperationName(
                request->GetOperationName());
        }
    }
}
```

Figure 7 Example of generated code of dynamic invocation routine

The example show in Figure 7 shows the code generated for one property and one entry point. Already there are three string compares needed to execute the setter function of the Counter property. This clearly illustrates the fact that the current SMP2 interface for getting and setting properties and calling operations does not scale well with the number of operations and properties of a model. Moreover, for very complex models, the number of generated "else if"s, which start in the Invoke() method runs into the limits of certain compilers, as every else starts a new, nested if statement.

CONCLUSIONS AND FUTURE ENHANCEMENTS

The next version of EuroSim will support almost all SMP2 features. EuroSim features an SMP2 development environment that offers an excellent way of working with the SMP2 standard and is fully integrated in the EuroSim toolset. This makes the tools easy to learn and easy to use for the EuroSim user. Compared to the current version of EuroSim, improvements are in improved support of SMP2 version 1.2 Catalogue features, Package, Assembly, and Schedule support, generation of source code that is more in line with other SMP2 tools, and code merging. This makes EuroSim a suitable candidate for use in SMP2 based simulation projects. Using EuroSim's native scheduling mechanisms, we make available EuroSim's hard real-time simulation capabilities to the SMP2 community. EuroSim's code generation capabilities offer optimal support for the SMP2 simulation model developer.

Regarding model portability and reuse, the next version of the SMP standard should be stricter in its C++ mapping specification and should specify handles for code merging.

The EuroSim consortium has been involved in the specification of the set of SMP standards, and will be in the future. EuroSim aims at implementing user-friendly graphical Catalogue, Assembly, Package and Schedule Editors, improved support of various ways of working with the standard, and improved support of working with source code. Code merging is only the beginning of the latter. A model developer should be able to develop C++ source code and generate a new Catalogue from it, or synchronise an existing Catalogue with source code that was originally generated from it and that was modified afterwards. This is a very practical and user-friendly way of working with SMP.

REFERENCES

[1] V. Reggestad, L.Argüello, A. Walsh, M-E. Bégin, "Simulation Model Portability (SMP) Past/Present/Future," *SESP 2002*, November 2002.

[2] A. Walsh, P. Ellsiepen, P. Fritzen, "Application of the Simulation Model Portability 2 Specification," *SESP 2006*, November 2006.

[3] Dutch Space, "EuroSim Mk4.1 Software User's Manual," *available on-line at www.eurosim.nl*.

[4] J.M. Moelands, W.F. Lammen, M. Jansen, M. Arcioni, Q. Wijnands, "Automatic Model Transfer from MATLAB/Simulink to Simulation Model Portability 2," *SESP 2006*, November 2006.