Methodology and Toolset for Model Verification, Hardware/Software co-simulation, Performance Optimisation and Customisable Source-code generation.

M. BERGER, J. SOLER, L. BREWKA, H. YU, M. TSAGKAROPOULOS, Y.LECLERC, C.OLMA
DTUx4, TELETEL SA, M3SYSTEMS,SEPROTRONIC GmbH
DTU: Tech. University of Denmark
Oersteds Plads 343, DK-2800 Kgs. Lyngby
DENMARK
msbe@fotonik.dtu.dk http://www.modus-fp7.eu

Abstract: - The MODUS project aims to provide a pragmatic and viable solution that will allow SMEs to substantially improve their positioning in the embedded-systems development market. The MODUS tool will provide a model verification and Hardware/Software co-simulation tool (TRIAL) and a performance optimisation and customisable source-code generation tool (TUNE). The concept is depicted in automated modelling and optimisation of embedded-systems development. The tool will enable model verification by guiding the selection of existing open-source model verification engines, based on the automated analysis of system properties, and producing inputs to be fed into these engines, interfacing with standard (SystemC) simulation platforms for HW/SW co-simulation, customisable source-code generation towards respecting coding standards and conventions and software performance-tuning optimisation through automated design transformations.

Key-Words: -Model Verification, HW/SW co-simulation, customizable code generation, SW optimization.

1 Introduction

Software quality is of primary importance in the development of embedded systems that are often used in safety-critical applications [1]. Moreover, as the life cycle of embedded products becomes increasingly tighter, productivity and quality are simultaneously required and closely interrelated towards delivering competitive products [2]. In this context, the MODUS (Methodology and supporting toolset advancing embedded systems quality) project aims to provide a pragmatic and viable solution that will allow SMEs to substantially improve their positioning in the embedded-systems development market. The project will develop and validate a set of technical methodologies, as well an open and customisable toolset, advancing embedded systems quality when using Formal Description Techniques (FDTs), by enabling:

- Model verification by guiding the selection of existing open-source model verification engines,based on the automated analysis of system properties, and producing inputs to be fed into these engines.
- Interfacing with standard simulation platforms for HW/SW co-simulation.
- Software performance-tuning optimisation through automated design transformations.
- Customisable source-code generation towards respecting coding standards and conventions.

In addition, the project will provide methodologies and open interfaces for customising and extending the MODUS toolset for use with different (domain-specific) FDTs, modelling practises, programming languages, target platforms, etc.

MODUS does not aim to be competitive with the vendors of CASE tools that are presently used in embedded software engineering. On the contrary, the project aims to allow the adoption of quality strategies by complementing these tools and preserving existing investments in technical-know.

2 Background

2.1 Formal verification and tools

Formal model verification (also known as formal model checking) concerns a wide range of techniques that are used for testing automatically whether a model of a system meets a given specification (set of properties). Most of the times,

such techniques apply for models of hardware/software systems that need to meet safety requirements such as the absence of deadlocks or critical states that can cause the system to crash, or may even be used to verify the timing properties of real-time systems.
Formal model verification relies on mathematically-based techniques for describing system properties, and they can be used to mathematically prove and ensure whether a system model meets a specified property or not. This is why safety-critical applications were the first to adopt such techniques.
In the framework of formal model verification, the system under verification is specified as a finite state machine (FSM). The system properties to be verified are expressed in temporal logic that allows reasoning over the possible execution paths. So, model verification tools accept two inputs: (i) The system model that may be represented in a wide-range of modelling languages (UML, SDL, Promela, IF, Verilog, etc.) and (ii) the models of the properties that may be represented in a properties modelling language (e.g. PCTL, PLTL, CTL, LTL, etc). The main inherent drawback of model checking techniques is the incapability to deal with infinite state spaces. Therefore, special assumptions are made and appropriate techniques are often applied for the cases of large or complex systems.

2.2 HW/SW co-simulation and tools

The proper operation of embedded software with the hardware is of particular importance. Currently, most of the tools used for the development of embedded systems support model simulation features that can be used for HW/SW co-simulation. Simulation is often used in conjunction with model verification; if a system design is found not to meet a given specification (set of properties) then simulation techniques may be applied to identify the design defects.
The industry has already realised the importance of standards for embedded systems HW/SW co-simulation. Towards this direction, SystemC[3] is promoted by OSCI, the Open SystemC Initiative, and has been approved by the IEEE Standards Association as IEEE 1666™-2005. SystemC is a C++ library/language used for the description of Systems on Chip (SoCs) at different levels of abstraction, from cycle-accurate to purely functional models. It is becoming a de facto standard for HW/SW co-simulation. SystemC is being increasingly used for writing the Transaction Level Models (TLM) [3] that allow embedded software development on a virtual prototype of the final chip. OSCI also provides an open-source proof-of-concept simulator, on which SystemC implementations can be validated in advance before integrated into the target platform.
However, existing CASE tools for embedded software development do not support interfaces to SystemC-based simulation platforms. Various techniques for generating SystemC from FDT models in languages as UML and SDL have been recently proposed [4], [5] but existing implementations are mainly proof-of-concept prototypes that concern a limited set of input modelling languages.

2.3 Methodologies and tools for software performance optimization

Code optimization is a critical component in achieving high performance for embedded systems. Computational specialists have adopted programming strategies affecting the utilisation of machine resources and have parameterized their algorithm implementations to accommodate the architectural variety of modern computing platforms [6], [7], [8]. While this approach has been quite successful, it is extremely error prone and time consuming for developers to manually program the management of hardware resources [9].
The programmer has to, first of all, make a potentially beneficial program modification, then compile it, before finally executing the new program and recording its execution time. This modify-compile-execute cycle must be repeatedly performed until a sufficient performance gain is achieved (or the programmer has run out of time).
Towards overcoming the aforementioned problems, there has been much work in the area of iterative optimisation aimed at automating this process [10], [11], [12], [13]. Such approaches focus on choosing good program modifications or transformations so that the number of modify-compile-execute cycles is reduced. Although it is possible to find good performance improvement automatically, iterative optimisation still requires many executions of different versions of the program. As execution time is frequently the limiting factor in the number of versions or transformed programs that can be considered, mechanisms that can automatically predict the performance of a modified program without actually having to run it have been proposed [9], [14].
The main shortcomings of available methods and tools for automated code optimisation can be identified as follows:

- Code optimizers typically only deal with a part of a program at a time, often the code contained within a single module; the result is that they are

unable to consider contextual information that can only be obtained by processing larger system parts. As a consequence, they focus on relatively shallow "constant-factor" performance improvements and most often do not improve the algorithmic complexity of a solution.

- Existing techniques are confined in the use of specific programming languages, target platforms and certain classes of program modifications.
- There exist no integrated developments environments (IDEs) that effectively combine code optimisation with HW/SW co-simulation. Therefore, performance tuning is very time consuming, as in order to test the effect of the performed modifications, developers have to iteratively compile the program, execute it and record its execution time on the given platform.

2.4 Present limitations relating to compliance to coding conventions/standards

Currently, various coding standards are available in different industrial sectors. The avionics industry for example requires that safety critical-software be assessed according to strict certification authority guidelines before it may be used on any commercial airliner. ARP 4754 and DO-178B are guidelines used both by the companies developing airborne equipment and by certification authorities. In this context, the main relevant shortcoming of present CASE tools is that their automatic code generation strategies are not sufficiently customisable. In fact, most often software developers have to manually reorganise the generated source code in order to comply with customer-/project-specific coding standards. Of course, there exist several open-source model-driven code generation tools (e.g. Open ArchitectureWare, Eclipse Motion Modelling, AndroMDA, Mia-Generation, etc) that can be extensively customised, but this sort of customisation is very time-consuming as it is based on the use of complex and not standard programming interfaces.

3 The MODUS project contributions

MODUS will apply and advance state-of-the-art technologies towards developing a set of methodologies and tools advancing embedded systems quality by enabling effective model verification, easy interfacing with standards for HW/SW co-simulation, model-level performance optimisations, and customisable source-code generation. In relation to the shortcomings presented in Section 2, MODUS advances the technological progress through the following innovative activities:

3.1 A harmonised methodological and tooling framework.

The project will provide a harmonised methodological and tooling framework for model verification, HW/SW co-simulation, performance optimisation, and customisable source-code generation, without placing restrictions on the use of FDTs and modelling languages.The MODUS framework will be centred on the definition of a Language Neutral Representation (LNR) for event-based systems that will be used as the intermediate format for interfacing with external tools and platforms. Advanced features as system-logic and architecture transformations for interfacing with different tools and platforms and optimising system performance, as well as system model analysis for selecting/customising model verification strategies, will be performed at the LNR model level, before generating the outputs to external tools and target platforms. It should be stressed though that the LNR format will be hidden from the users of the tools. In fact, they will be provided with the means to develop high-quality implementations by using their favourite Formal Description Techniques (UML, SDL, Simulink, LUSTRE, etc.).

3.2 Effective exploitation of existing model verification techniques

MODUS will allow the effective exploitation of existing model verification techniques that are currently dispersed across different modelling frameworks and tools. As explained in Section 2, formal model verification features are incorporated in a limited set of CASE tools for use with specific modelling languages. On the other hand, there exist many stand-alone formal model verification tools but these are highly specialised and do not adopt harmonised modelling approaches. In this context, MODUS will not merely provide a front-end to existing tools for formal model verification. It will further develop a tool that will guide the selection of the underlying model verification techniques to be used, through the automated analysis of the input system models and properties to be verified.

3.3 Formal representation of coding standards/conventions andautomatic generation of code.

MODUS will provide a methodology and tool for the formal representation of coding standards/conventions and the automatic generation of code that complies with them. As already mentioned, the code generation strategies of popular CASE tools are not sufficiently customisable. On the other hand, the customisation of open-source model-driven code generation tools is very time-consuming as it is based on the use of complex and not standard programming interfaces. As a consequence, in practise software developers have to manually reorganise the source code generated by CASE tools in order to comply with customer-/project-specific coding standards.MODUS will define a formalism for the effective modelling of coding standards/conventions and will develop a tool for the customisable generation of code that respects these formal representations. This will enablesoftware developers to automatically acquire high-quality source code without needing to repeatedly and manually apply the coding rules on the source code generated by CASE tools.

3.4 Performance optimisation applicable across different target platforms.

The project isgoing to develop a methodology and tool for performance optimisation that will facilitate the improvement of the algorithmic complexity of software designs and will be applicable across different target platforms.As explained in Section 2 present code optimisers are for use with specific programming languages and platforms. Moreover, they are confined to relatively “shallow” performance improvements that are applicable to small system parts.The MODUS will provide a code optimisation approach that will be based on transformations performed at the Language-Neutral-Representation (LNR) model level. It will make use of patterns for identifying and applying applicable transformations relating to the control-flow complexity (state-transition logic) and data-flow complexity (algorithms and processing taking place within states) of a software system, rather than just rewriting small code blocks in a source-code implementation. This will be achieved through the exploration/analysis of the LNR models by using formally represented rules for identifying design patterns for which optimisation-related transformations apply. Up to now the use of formally represented design patterns has been confined in the context of formal model checking/verification techniques but has not been applied for the purposes of performance-tuning transformation.

The MODUS approach will provide significant benefits by facilitating the improvement of the algorithmic complexity of software designs. In addition, through the use of the MODUS code generation tools, it will allow the easy derivation of optimised source code for different target platforms.

4 The MODUS Approach to enhance embedded system quality.

To meet its overall goal, and the objectives identified in Section 3, MODUS shall design and develop a set of methodologies and tools for model verification, HW/SW co-simulation, and customisable code generation.

4.1 Infrastructures for interfacing with existing tools

The project will first develop the tooling infrastructure that will allow the effective interfacing with existing tools and platforms. The development of the MODUS framework for model transformation and code generation will involve the accomplishment of the following objectives:

- Definition of a language neutral representation (LNR) for the generic modelling of event-oriented systems.
  LNR will be used as the intermediate format for generating different types of system representations, i.e. inputs to model verification engines, inputs to HW/SW co-simulation engines, and optimised source-code system implementation that respects coding standards/conventions. By using a standard intermediate format, any model transformation or code generation strategy will rely on inputs in the same formalism (LNR). The definition of the LNR will be based on a thorough analysis of the behavioural concepts and date types of a wide range of potential input FDTs / modelling languages (UML, SysML, SDL, LUSTRE, etc), as well of the potential output modelling/programming languages (Promela, IF, SMV, C, Ada, etc.).
- Definition of the strategies for the generation of LNR system representations from FDT-based system models.
  The concepts and structure of the language, in which the system/model is implemented, have to be known by the LNR generator. The input system/model will be parsed (based on the

appropriate model parsing profile as shown in Figure 1) and the characteristics of the input language will be appropriately categorised.

- Definition of the strategies for model transformation.
  Both the input and output models of the LNR model transformation engine will be in the LNR formalism. In the context of the MODUS approach, model transformation will serve two main purposes: Model transformations applied at the LNR-model level (e.g. data-flow optimisations, loop optimisations, etc.) towards optimising the system runtime performance. Formal refinement of system models for adding information (e.g. timing properties, platform information) willbe used for the representation of system properties for different types of model verification and simulation engines.
- Definition of the strategies for code generation.
  A technique will be defined for the generation of code in different modelling/programming languages using as inputs the LNR-based system models. These mechanisms will provide the front-ends to different tools and platforms. They will also be responsible for enforcing non-functional coding rules as naming conventions, addition of comments, file organisation, etc. Code generation will rely on the use of code generation patterns (templates) that will be based on state-of-the-art languages used for such purposes (e.g. ATL, TXL, Stratego/XT, CIL, etc).

4.2 Tool for model verification and HW/SW co-simulation

The MODUS framework for model verification and HW/SW co-simulation will remove the need for software engineers to deal with the implementation details of the modelling languages used by existing model verification engines (e.g. Promela, IF, SMV, LTL) and HW/SW co-simulation platforms (SystemC). The implementation of the relevant tools will exploit the LNR-based model transformation and code generation tool (Section 4.1). The model verification and HW/SW co-simulation tools will be used in conjunction with existing modelling environments / CASE tools.
The MODUS tool for model verification and HW/SW co-simulation will support the following key features:

- Automated generation of system representations in the formalisms used by existing model verification engines and HW/SW co-simulation platforms from system models in different FDTs / modelling languages.
- User friendly modelling of system properties (temporal logic expressions) will be verified by means of model checking, without needing to deal with the multitude of relevant property languages (e.g. PCTL, PLTL, CTL, etc) used by existing model checking tools.
- Interactive control of model verification and HW/SW co-simulation strategies by the user through the analysis of the characteristics of the automatically generated LNR system models in relation to the model verification / simulation requirements. For instance, this feature will allow the selection of a model checking tool (and relevant target formalism) that better deals with the large size of a system as compared to others, or it is more appropriate for the verification of specific aspects (e.g. timed properties).

The development of the MODUS framework for model verification and HW/SW co-simulation will involve the accomplishment of the following objectives:

- Analysis of the features of existing model checking tools (SPIN, NuSMV, Uppaal, EmbeddedValidator, etc) and development of a relevant knowledge base.
- Definition of a methodology for the GUI-based modelling of system properties to be verified by means of model checking.
- Definition of the strategy for the automated analysis of generated LNR system models in relation to model verification and HW/SW co-simulation requirements, and the interactive control of the relevant strategies based on the contents of the knowledge base.
- Prototype development of the TRIAL tool including the development of the libraries of Model Transformation Patterns and Code Generation Patterns for automatically generatingthe inputs to existing model verification and HW/SW co-simulation platforms.
- Design and development of the module allowing the analysis of generated LNR system models in relation to model verification and HW/SW co-simulation requirements, and the interactive control of the relevant strategies.
- Design and development of the GUI for the user-friendly modelling of system properties under verification.

4.3 Tool for performance optimisation and customizable source-code generation

The MODUS approach to source-code generation will combine the automated organisation of source code according to coding conventions/standards, with the optimisation of system performance.
In detail, performance-tuning optimisationsare related to the provisions made for optimising system performance and/or handling trade-offs between memory utilisation, speed, power consumption, etc, by applying transformations as data-flow optimisations, loop optimisations, etc.
Coding conventions/standards are related to a wide-range of source-code implementation aspects as complexity, security, readability, maintainability, etc.
The MODUS tool for performance-tuning optimisation and customizable source-code generation will support the following key features:

- Assisted design optimisation for meeting performance requirements, through the iterative, automated analysis of the generated LNR models and the identification of applicable optimisations.
- Formal modelling of coding conventions/standards using a formalism that will defined within the project. Moreover, checking and identification of conflicts between the specified coding convention/standard rules towards allowing the user to provide consistent specifications.

The development of the MODUS toolfor performance-tuning optimisation and customizable source-code generation will involve the accomplishment of the following objectives:

- Definition of the strategies for applying different types of design optimisations. This will involve the mapping of the identified types of optimisations to “low-level” model transformation patterns represented in the selected “low-level” code transformation language (e.g. ATL, SmartQVT, Kermeta, etc).
- Definition of the strategies for the automated analysis of generated LNR models and discovery of applicable optimizations. These strategies will involve the processing of the generated LNR models for finding the system blocks that each candidate optimisation may apply for. This will be achieved through the exploration/analysis of the LNR models by using formally represented rules for identifying design patterns for which optimisation-related transformations apply.
- Analysis of existing coding standards and widely-adopted coding conventions (e.g. ARP 4754, DO-178B, etc).

5 MODUS toolset technical specifications

In the following sections, an overall description of the MODUS toolset technical specification is provided by giving a detail description of the MODUS tool components in terms of functional specifications, architecture, interfaces and deployment features.

5.1 MODUS Infrastructure for interfacing with existing modelling languages

One of the main goals of the MODUS project is to incorporate different Formal Description Techniques (FDTs) that are widely-used into the software development process of embedded systems (e.g. UML, SysML, etc.) towards the definition of a language neutral representations (LNR) of the input models. The LNR will represent a standard intermediate format which will facilitate the model transformation for generating different types of system/software representations to modelling/programming languages(e.g. Promela, C, SystemC, etc.) for model verification, source code generation or HW/SW co-simulation.

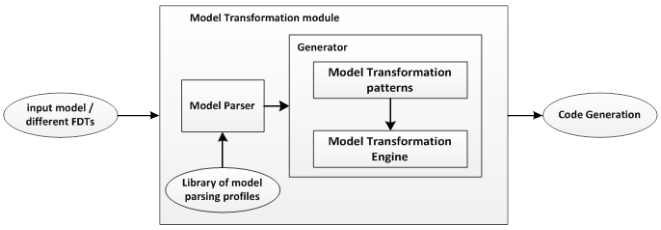

Figure 1: MODUS infrastructure for interfacing with existing modelling languages

Figure 1illustrates the proposed infrastructure for interfacing with existing modelling languages based on a model transformation module. The model transformation module will include two main components: the Model Parser and the Generator.
The Model Parser will parse the different FDTs models based on a library of model parsing profiles. The library of model parsing profiles will be extendable and will include the description of behavioural concepts and data types according to FDTs input models. The Generator will generate the LNR representation of the input model according to specified transformation patterns that correspond to

the mapping of input models concepts to the defined LNR language

Based on the initial analysis of the state-of-the-art review of existing technological requirements and SMEs' development strategies from different application domains [1], UML has been identified as the main modelling language for a wide range of applications/systems in different sectors.

The Unified Modelling Language (UML) is a standardized general-purpose modelling language in the field of object-oriented software engineering created by the Object Management Group. UML has become the industry standard for modelling software-intensive systems and includes a set of graphic notation techniques to create visual models of object-oriented software-intensive systems. UML is used to specify, visualize, modify, construct and document the artifacts of an object-oriented software-intensive system under development.

Towards the definition of a common formalism for the transformation of the input modelling languages, UML language and its extensions (i.e. modelling languages that reuse UML's concepts in terms of stereotypes, definitions, constraints, diagrams etc.) like SysML and UML MARTE are considered as the starting point of LNRs specification. Thus, in the context of this deliverable, UML-based input models will be considered for the initial specification of the separate functionalities of the MODUS toolset below.

5.2 FORMAL MODEL VERIFICATION

The Formal Model verification functionality of the MODUS toolset will provide to the user the means for performing the formal validation of the system designed.

The interface that users interact with should be integrated with Eclipse IDE providing an environment in which a lot of developers are familiar with.

A wizard-like menu system will be implemented in the MODUS toolset where information required for formal verification and validation can be provided by the user. This information includes:

- description of the system,
- preferences for the formal modelling language,
- properties to be verified (described in a syntax coherent with the modelling language used for the formal verification).

The block diagram of the Formal Verification and Validation module of the MODUS toolset is presented in Figure 2.

Major part of the interfacing with the user is realised within the Eclipse modelling framework [4](depicted by the Eclipse block). The input i.e., the UML diagram of the system to be verified can be created or modified, and provided from the Papyrus model editor.

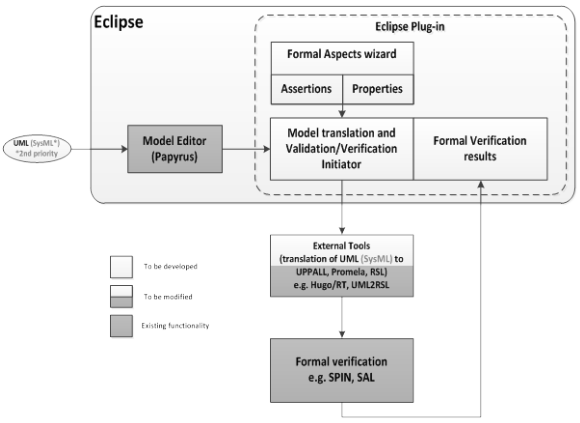

Figure 2: Integration of tools for Formal Model Verification within the MODUS tool-set

5.3 HW/SW CO-SIMULATION

The architectural block diagram of the HW/SW co-simulation module is depicted in the following figure:

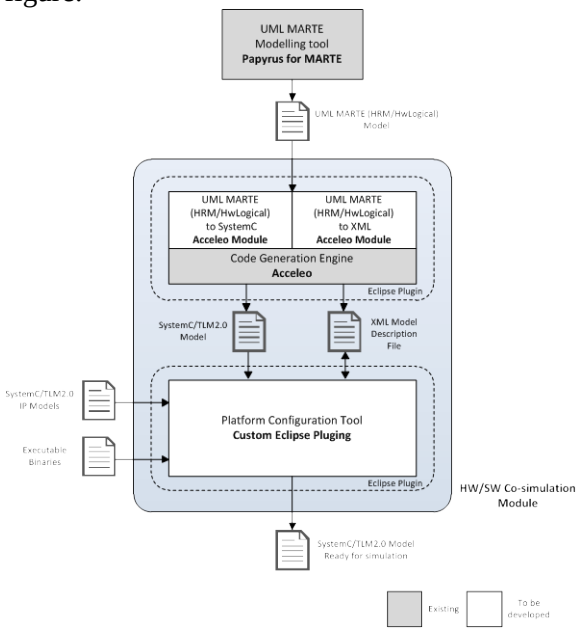

Figure 3: HW/SW Co-simulation Module - Architectural Block Diagram

The HW/SW co-simulation module is made of two elements:

- The SystemC/TLM2.0 model and XML model description file Engine.
  This element generates the SystemC/TLM2.0 architecture of a virtual platform, taking as input a UML MARTE HRM/HwLogical model. To

be more precise, the subset of the model used by the element is the Component Diagram (see §3.3.1). Along with the SystemC model, a XML model description is generated (useful for the second element of the module).
In order to implement this functionality, a generic code generation engine is used: the Acceleo Code Generation Engine[13]to generate SystemC/TLM2.0 and XML.Two specific Acceleo modules have to be designed: one for SystemC/TLM2.0 generation, one for XML generation. This element will mainly be based on specific generation scripts and templates, and will be provided as an Eclipse plugin.

- The Platform Configuration Tool.
  This toolgenerates the SystemC/TLM2.0 virtual platform, ready for simulation, taking as inputs the SystemC/TLM2.0 architecture and the XML description file generated by the previous element, a set of SystemC/TLM2.0 IP models, and a set of executable binaries.
  In addition, it provides a front-end interface forMODUS' users. This graphical interface helps the users: in managing the SystemC modules' skeletons in which to manually include their IPs, in managing the SystemC modules corresponding to the memory components in which to manually link their executable binaries, and in setting the simulation-relevant parameters. The XML description file summarizes the information related to the configuration, and is extensively used by this module. This module will be provided as an Eclipse plugin.

5.4 Customizable source code generation.

The modelling concepts of the input models that will be implemented by the code generation mechanism will be identified. For each of these modelling concepts, the implementation strategy and the properties of the code generation that can be customized by the end-users will be defined. Specific parameters (customisable properties) can be defined by the end user in order to select or control the target language implementation for every semantic. The code generation module architecture for the MODUS toolset is illustrated in Figure 4.
The code generation module will include two main components: a graphical interface for the configuration of the generator, the code generator GUI (Code Generator GUI Plugin) and the main code generator (Code Generator Module Plugin).
Code Generator GUI
The code generator GUI will be based on the following blocks:

- Front-End
- Properties
- Generation Execution

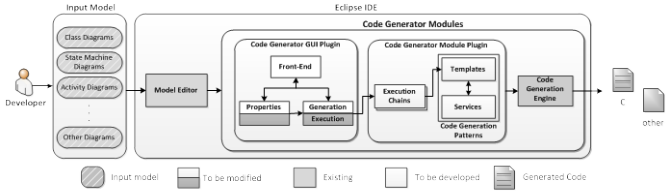

Figure 4: Code Generation Architecture Diagram

The front-end will be the generator interface relative to the user, where the user interacts with it directly. Through the front-end the user can access the code generator properties and can start the code generation procedure. An interface can be a popup menu, or a menu in the toolbar of Eclipse. The properties element is a menu called by the front-end, where the user can set specific actions to the source code generator. Such actions can be the generation output folder, to generate header files, to generate the C makefile, to generate the author name in all the source files, etc. The generation execution will be the main entry point of the code generation module. According to the properties set by the user, it will call the code generation module plugin for starting the actual code generation procedure.
The code generator module will be developed with different levels of abstractions based on templates, services and execution chains elements.
Templates will describe the information required to generate source code from a meta-model such as UML. Within each template, several scripts will enable the developer to customize the generator accurately. The developer will write the templates repository in Model to Text Language (MTL) [9], an implementation of the Object Management Group (OMG) MOF Model to Text Language (MTL) standard [9].
Services will be used to extend templates in order to implement complex operations that would be very complicated to implement within the template files. Services will be public methods which are executed in the same context, from one call to another.
Execution chains will be used to execute a generation for a target application. They allow the chaining of several generations and operations on models. Execution chains will be used to simplify execution and customization operations for the generation. The actions part, also called action set, will contain all the tasks that must be carried out by

the execution chains according the configuration of the code generation module.
Finally, the Code Generation Engine will be parse the input model and the specified template/services described above for the selected target source code language in order enable the transformation of the input model to the selected source.

5.5 Performance Optimisation

The architecture of the performance optimisation module for the MODUS toolset is presented in Figure 5.

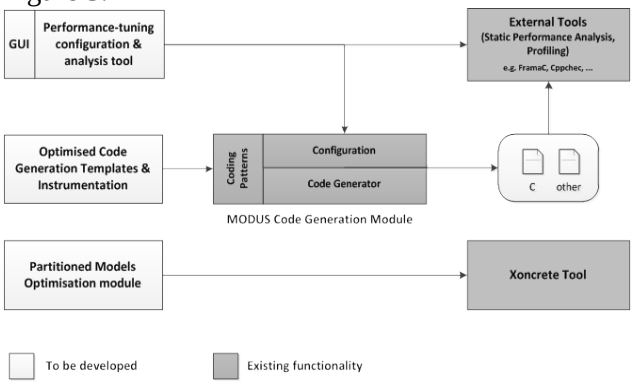

Figure 5: Performance Optimization Module Architecture Diagram

The performance optimization module will include three main components: a graphical interface for the configuration of the module, the optimised code generation templates& instrumentation and the partitioned models optimisation module.
The performance optimisation GUI will provide the user configuration interface for the performance-tuning functionalities. Through the GUI the user will access the separate functions executed by the performance optimisation module and will be able to configure the relevant parameters for each functionality. A menu with the available configuration options will be integrated within Eclipse, where the user shall include the desirable configurations and specific actions for the execution of the performance optimisation module.
The optimised code generation templates& instrumentation component will provide additional code generation strategies to achieve code optimisation according to developer's needs and constraints. Different programming strategies affecting the utilisation of machine resources will be defined and evaluated for the UML modelling concepts implemented by the code generation mechanism. In order to evaluate the performance impact of each source code generation strategy, specific performance indicators will be identified including memory consumption and code execution metrics. The developed code generation strategies will include templates, services and execution chains for each performance indicator, as described in section 5.4:
The partitioned models optimisation componentwill include the parsing mechanism of UML-specific model representations of a virtualised embedded system. Additionally, it will provide the interworking interface with the Xoncrete external tool [15]making the proper model transformations according to the Xoncrete tool API specification. Finally, it will generate the optimized scheduling configuration file according to the user input that will enable the deployment of the XtratuM hypervisor [16]on the target platform based on the input model specifications.

5.6 Toolset Integration

In the frame of the MODUS project, the Eclipse Modelling Framework (EMF)[4] will be used which is a modelling framework for building tools and other applications based on a structured data model. EMF provides an open source integrated and user-consumable environment for editing any kind of EMF model and particularly supporting UML and related modelling languages such as SysML and MARTE. In addition, the integrated Papyrus tool provides diagram editors for EMF-based modelling languages amongst them UML 2 and SysML and the glue required for integrating these editors (GMF-based or not) with other CASE tools. MODUS tool will be composed by extension modules that implement the various supported functionalities in an independent and configurable way.

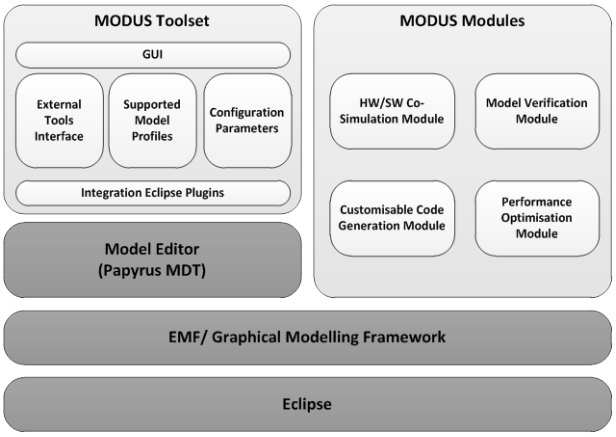

Figure 6: MODUS toolset integration architecture

Figure 6 presents the integration architecture of the MODUS toolset. By building on the infrastructure

provided by the EMF, the MODUS tool will be highly customizable and can be easily augmented with additional features in an incremental and non-invasive manner.

6 Conclusion

MODUS is targeting the market of tools for embedded software engineering. The project will develop a toolset advancing embedded systems quality that will target the growing group of SMEs (and bigger companies as well) specialising in the development of embedded systems in different industrial sectors (e.g. avionics, automotive systems, consumer electronics, telecommunications systems, etc).

It should be stressed that MODUS does not aim to be competitive with the big vendors of CASE tools presently used in embedded software engineering. On the contrary, the project aims to allow the adoption of quality strategies by preserving existing investments in technical-know and tools. The MODUS approach is aligned with present market needs; the familiarity with tools, ease of use, and compatibility/interoperability remain among the most important criteria when selecting the development environment for a project.

7 References:

[1] XIN BEN LI, DE CHAO SUN, ”Formal Analysis and Verification of a Communication Protocol”, 5th WSEAS Int. Conference on Information Security and Privacy, Venice, Italy, November 20-22, 2006

[2] Ďurfina, L., Křoustek, J., Zemek, P., Kolář, D., Hruška, T., Masařík, K., Meduna, A.: Design of an Automatically Generated RetargetableDecompiler, In: 2nd European Conference of COMPUTER SCIENCE (ECCS'11), Puerto De La Cruz, Tenerife, ES, NAUN, 2011, p. 199-204, ISBN 978-1-61804-056-5

[3] Hyperlink at: http://www.systemc.org

[4] W.H. Tan, P.S. Thiagarajan, W.F. Wong, Y. Zhu and S.K. Pilakkat “SynthesizableSystemC Code from UML Models,” (2004) hyperlink at: http://www.scientificcommons.org/43529677

[5] Sergey Balandin, Michel Gillet, Alexey Rabin, ValentinOlenev, Alexander Stepanov, Irina Lavrovskaya, “SystemC and SDL Co-ModellingImplementation”, hyperlink at: http://fruct.org/conf7/Rabin_SystemC_and_SDL_Co-Modelling_Implementation.pdf

[6] R. C. Whaley, A. Petitet, and J. Dongarra. Automated empirical optimizations of software and the ATLAS project. Parallel Computing,27(1):3–25, 2001.

[7] R. Vuduc, J. Demmel, and K. Yelick. OSKI: An interface for a self-optimizing library of sparse matrix kernels, 2005.

[8] M. Frigo and S. Johnson. FFTW: An Adaptive Software Architecture for the FFT. In Proceedings of the International Conference on Acoustics, Speech, and Signal Processing (ICASSP), volume 3, page 1381, 1998.

[9] Qing Yi, “Automated Programmable Code Transformation For Portable Performance Tuning,” Hyperlink at: http://www.cs.utsa.edu/~qingyi/papers/ROSE2POET.pdf

[10] M. Stephenson and S.P. Amarasinghe. Predicting unroll factors using supervised classication. In Proceedings of International Symposium on Code Generation and Optimization (CGO), pages 123-34, 2005.

[11] B. Franke, M.F.P. O'Boyle, J. Thomson, and G. Fursin.Probabilistic source-level optimization of embedded programs. In Proceedings of the Conference on Languages, Compilers, and Tools for Embedded Systems (LCTES), pages 78-96, 2005.

[12] K.D. Cooper, A. Grosul, T.J. Harvey, S. Reeves, D. Subramanian, L. Torczon, and T. Waterman. Acme: adaptive compilation made e cient. In Proceedings of the Conference on Languages, Compilers, and Tools for Embedded Systems (LCTES), pages 69-77, 2005.

[13] P. Kulkarni, W. Zhao, H. Moon, K. Cho, D. Whalley,J. Davidson, M. Bailey, Y. Paek, and K. Gallivan. Finding e ective optimization phase sequences. In Proceedings of the Conference on Languages, Compilers, and Tools for Embedded Systems (LCTES), pages 12-23, 2003.

[14] Christophe Dubach, John Cavazos, BjörnFranke, GrigoriFursin, Michael F.P. O'Boyle, Olivier Temam, “Fast compiler optimisation evaluation using code-feature based performance prediction,” CF '07 Proceedings of the 4th international conference on Computing frontiers.

[15] Xoncrete, Integrated editor and analysis tool/planning tool for XtratuMhypervisor, http://www.fentiss.com/en/products/xoncrete.html

[16] XtratuMhypervisorfor real-time embedded systems, http://www.xtratum.org/