ITEA

INFORMATION TECHNOLOGY

FOR EUROPEAN ADVANCEMENT

MARTES

Model-Based Approach for Real-Time Embedded Systems development

Title:

Tool Enhancements and Gateways

Deliverable ID:	2.10
Version:	1.0
Date:	19/11/07
Editor:	Michel Barreteau
Status:	Final
Confidentiality:	Public

ITEA

INFORMATION TECHNOLOGY

FOR EUROPEAN ADVANCEMENT

Authors:

Author	Partner	Email
Michel Barreteau	TRT	michel.barreteau@thalesgroup.com
Marko Hännikäinen	TUT	marko.hannikainen@tut.fi
Dennis Alders	NXP	dennis.alders@nxp.com
Stefan Van Baelen	K.U. Leuven	stefan.vanbaelen@cs.kuleuven.be
Aram Hovsepyan	K.U. Leuven	aram.hovsepyan@cs.kuleuven.be
Fernando López	TI+D	fla@tid.es
Julio Cano	UC3M	jcano@it.uc3m.es
Natividad Martínez	UC3M	nati@it.uc3m.es
Ralf Seepold	UC3M	ralf@it.uc3m.es
Per Andersson	Lund	per.andersson@cs.lth.se
Johan Devos	Barco	johanr.devos@barco.com
Vincent Perrier	CoFluent	vincent.perrier@cofluentdesign.com
Paul Feautrier	INRIA	paul.feautrier@ens-lyon.fr
Pablo Sanchez	Cantabria	sanchez@teisa.unican.es
Bruno Lheritier	Softeam	bruno.lheritier@softeam.fr

Document History:

Date	Version	Editor	Description
30/05/07	0.0	M. Barreteau	Initial template
30/05/07	0.1	M. Barreteau	Integration of TUT and NXP contributions
11/06/07	0.2	S. Van Baelen	Integration of K.U.Leuven contribution
04/07/07	0.22	F. López	Inputs of TI+D & UC3M
30/06/07	0.3	M. Barreteau	Inputs of Barco, Lund and TUT update
10/10/07	0.4	M. Barreteau	Inputs of INRIA, TRT and CoFluent Design
19/11/07	1.0	M. Barreteau	Final version

Filename: D2.10_v1_ITEA-MARTES.doc

TABLE OF CONTENTS

Executive Summary

This deliverable summarizes, for each partner, its tool enhancements and gateways that have been implemented within the MARTES framework.

1 Introduction

The goal of this document is to describe the different tool enhancements and gateways that have been developed by each partner thanks to MARTES. It enables to introduce the software package and a user manual-like. Each section matches a partner implementation.

2 Lund University - Telelogic

The university of Lund has a close cooperation with Telelogic developed a UML to SystemC code generator as an add-in to Telelogics Tau G2 UML modeling tool. During initial system modeling, a pure UML model is used. Though it is possible to define a set of mapping rules from a pure UML model directly into SystemC code, it would give the engineer little influence on the mapping and most likely a less satisfactory result. Instead we divide the mapping into three steps. All models are available and editable. This makes it possible for the engineer to have full control over the relevant details for the system under development and have the tool manage all remaining details.

Step 1, vertical refinement transformation: In this step an initial UML description is refined to a UML description, which follows a UML profile for SystemC. This step will, at least partly, be carried out manually. To minimize the design effort it should not be required to tag the whole model. This means that a set of default values for the SystemC specific attributes of the UML profile, must be defined. The default values will in most cases provide a satisfactory mapping in the following transformation steps.

Step 2, vertical refinement transformation: In this step the model is transformed into a new UML description that only includes UML constructs with direct representations in SystemC, i.e. classes, attributes, inheritance etc. Other constructs such as state machines are translated to the target language. During this step we transform each state machine to a class with methods that implement the behavior of the states and transitions. In the first version of the tool, the resulting model will be an un-timed functional model. A complete list of UML constructs which are removed during this transformation is beyond the scope of this paper. In addition to removing UML only concepts, we also make all relations in the model explicit. When a class is made active in UML it implies that the class will have its own thread of execution. In SystemC this is realised using SC_THREAD or SC_METHOD which implies that the class is an instance of the SystemC class sc_module. During this transformation all such implicit relations are made explicit. For example, we add a generalization relation to the SystemC class sc_module from all active UML classes. In the first version of the tool the resulting model is a untimed functional model.

Step 3, horizontal transformation: In this step the UML model resulting from step 2 is transformed into a corresponding SystemC code. This transformation is a one to one correspondence between the UML model to the resulting SystemC code, i.e. this is a "pretty print" of the UML model. This step is implemented using the existing C++ code generator from Telelogic and thus reuse its support for scope rules, header-file inclusion and make file generation without any modifications. If the generated code is to be read by humans it is desirable to use the common SystemC macros when applicable. This requires a slight customization of the syntax of the generated code. We do this using an agent, a mechanism which makes it possible for third party executables to interact with the C++ Code generator in Telelogic Tau G2. Our agent generates SystemC like module declarations, instead of a C++ class declarations, SC_MODULE(MyModule){...} instead of class MyModule:public sc_module{…}.

In addition to SystemC code generation the Telelogic Tau tool has been extended with an add in for exporting UML models to the MARTES eCore file format. The export have support for structure, simple behavior as well as timing. The timing information and also some other properties, such as queue policies, are added to the UML model using stereotypes.

3 Telefonica

Large and complex systems design is still being a challenge, which is even bigger when developing embedded, distributed or real-time systems. OSGi is a platform created to reduce some of the software designing problems, increasing reusability, modularity, etc. A methodology based in MDA, and aiming real-time embedded systems, can be extended to have OSGi as the target platform reducing applications development time and complexity [3].
Telefonica, along with the UC3M, have developed a toolset for the MARTES project, based in the idea presented above. This toolset provides service modelling facilities and all the transformations needed for facing a MDA development process for distributed system based on an OSGi platform.

3.1 Design steps

Service design steps using MARTES Telefonica+UC3M toolset, can be summarized as follows:

- Requirements modelling. A requirements analysis and a use case capture is carried out. Any relevant information is extracted using a service-oriented (CIM) modelling.
- Semi-automatic transformation of the requirement model into the application and the execution platform models
- More detailed modelling, including application, execution platform, and mapping between them
- Allocated to implementation automatic model transformation, including further modelling via model edition capabilities.
- Java code generation.

3.2 Meta-model and model definition

For modelling definition purposes Omondo Eclipse UML Free Edition [1] is used. Omondo is a visual modelling tool natively integrated with Eclipse [2], which allows the user to instantiate and edit a concrete model.

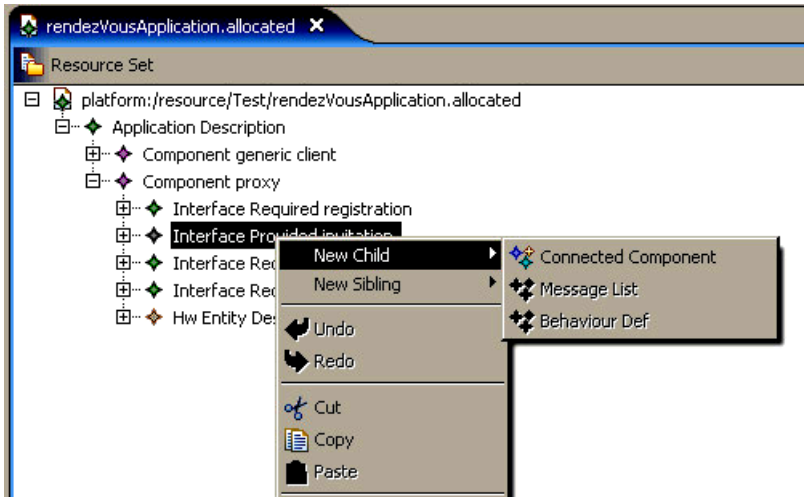


Figure 1: Model edition

Application, execution platform, allocated and implementation meta-models are defined following the Eclipse Modelling Framework [6], and then saving in Ecore format. A fully functional Eclipse editor is generated for every Ecore metamodel. This editor is split between two plug-ins:

- an "edit" plug-in includes adapters that provide a structured view and perform command-based editing of the model objects;
- an "editor" plug-in provides the UI for the editor and wizard. (see Figure 1:). This editor provides useful load, edition and validate methods for a given model, according to the corresponding meta-model.

It is worth mentioning that any XMI-based modelling tool could be used instead of the one packed with Omondo. In this regard, interoperability among different modeling tools is guaranteed. A complete interoperability test has been done using ArgoUML [7].

Service requirements and specifications are gathered via UML 2 diagrams. Handmade use case, sequence, activity and deployment diagrams are defined by the user according to the target service initial specifications. All the information gathered is automatically processed and then mapped into the application and execution platform model. Due to the possible complexity of the specified service, the resulting application and execution platform could need a manually editing and refinement process.

Every model can be saved in XMI format for portability purposes.

3.3 Model Transformations

Transformations in MARTES TID+UC3M tool are carried out thanks to JAXP (Java API for XML parsing) [4]. This API provides basic functionality for reading, manipulating and generating XMI [5] (XML pseudo-XMI) documents. JAXP Transform package includes all the functionality needed for XMI transformations.

Model transformation languages are defined in a meta-model level and establish the relationship between source meta-model elements and target meta-model elements [3]. TID+UC3M tool uses XSLT language to represent the transformation rules. An XMI model file is processed using an XSLT engine, generating the transformed target model in XMI format, which can be then edited and refined using the model editor introduced before.

A XSLT template is defined for every transformation needed during the modelling process. These templates are integrated into the tool, but are accessible for the user to modify and adjust them.
MARTES TID+UC3M tool transformation dialog is shown in Figure 2:

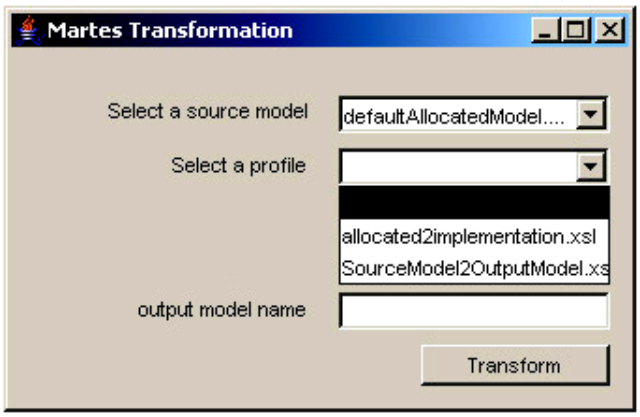


Figure 2: Martes Transformation Dialog.

3.4 Code Generation

Code generation is based on the Eclipse EMF (Eclipse Modelling Framework) ability to generate Java code from ecore models the same way it is used to generate plugins to edit models based on ecore metamodels.

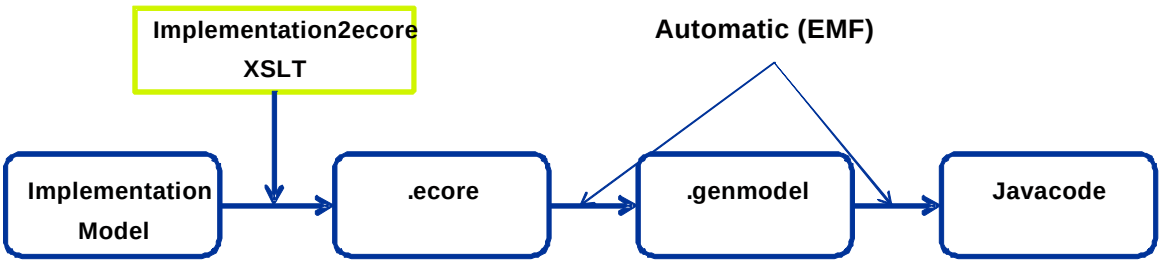


Figure 1 Java code generation using EMF

A XSLT is used to transform the implementation model in XMI format to an ecore file containing the model for the code. From the ecore file a generator model is created. This generator model automatically generates all the specified Java code. These last two steps are done using the EMF Eclipse plugin.

3.5 MARTES Eclipse plug-in

The toolset introduced before has been packed as an Eclipse plugin, which extends the capability of your development platform, allowing Eclipse users to follow the MARTES MDA-based modelling methodology easily. In addition, Omondo Eclipse UML services are accessible from the same development environment.

After installing the MARTES TID+UC3M plugin, a user will be able to access to the MARTES modelling facilities from a menu item located in the Eclipse toolbar, as is shown in Figure 3:.

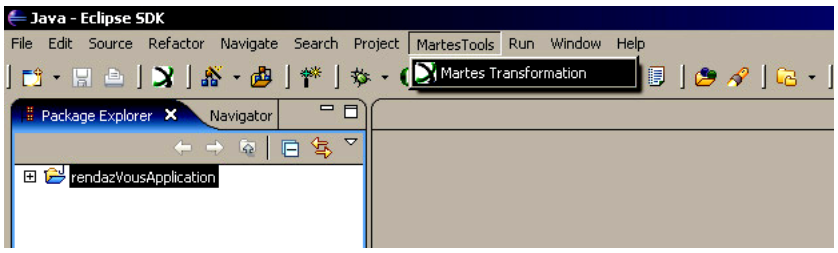


Figure 3: Martes menu item

3.6 References

[1] Omondo. http://www.omondo.com/

[2] Eclipse. http://www.eclipse.org/

[3] J. Cano, N. Martinez, R. Seepold, F. López Aguilar, "Model-driven development of embedded system on heterogeneous platforms", Forum on Specification & Design Languages (FDL'07), Barcelona, Spain, September 18-20, 2007.

[4] Java API for XML Processing (JAXP). http://java.sun.com/webservices/jaxp/

[5] http://www.omg.org/technology/documents/formal/xmi.htm

[6] http://www.eclipse.org/modeling/emf/

[7] ArgoUML. http://argouml.tigris.org/

.

4 Tampere University of Technology

TUT's Koski consists of several separate tools that form a complete design flow utilized for the MARTES project by TUT. Most of the tools are developed at TUT. In addition, few defined third-party tools are integrated into the Koski flow. An overall view of Koski is shown in Fig 1.

TUT has two major contributions: UML profile for embedded system design [8] and automated design flow [9][10]. Koski is a proprietary tool framework completely and independently developed by TUT. Koski has been used for UML2.0 (TUT-Profile oriented) prototype designs of several multiprocessor systems, including the Wireless Video Terminal implementations [4] presented, for example, in ITEA Symposium 2006 in Paris.

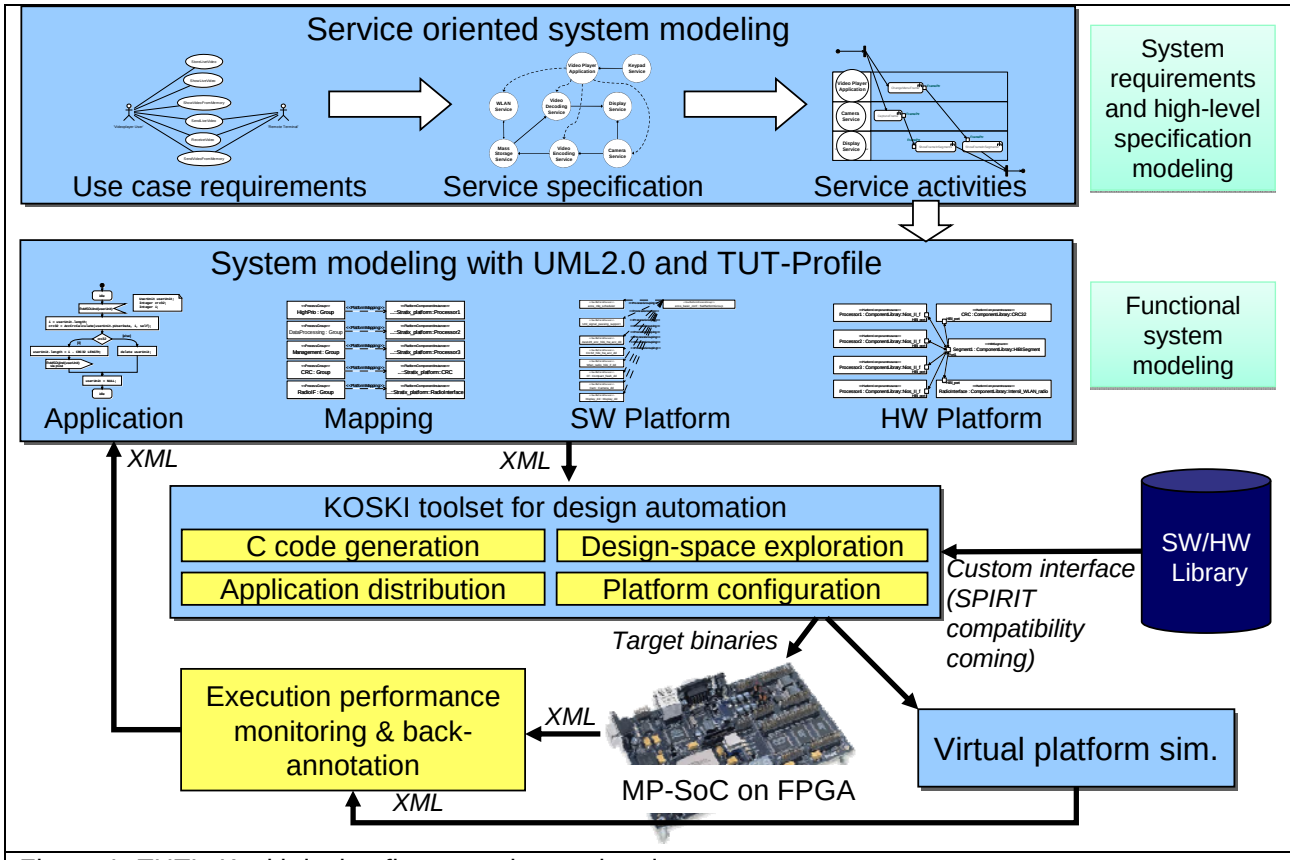


Figure 1. TUT's Koski design flow overview and main steps.

4.1 Design steps and Tools used in Koski

System design steps in Koski are shown in Fig 1. and briefly summarized as follows:

- Requirements analysis, use case capture, and service-oriented (high-level) modeling
- More detailed modeling, including application, execution platform, and mapping between them
- System optimization with automated design space exploration
- Code generation (SW, HW), compilation and synthesis
- Execution either on FPGA or virtual platform simulator. Performance is monitored and statistics are collected in both cases.
- Back-annotation of gathered performance data in order to simplify system analysis and further development.

4.2 Tools used in Koski

Koski [9][10] is rather large concept and consists of several tools which interact in seamless manner. The Koski development environment runs both on Linux and Windows.

Tools are summarized as follows:

Modeling

- Initial requirements and specification are captured in free (human-readable) form with text, figures and tables.
- The modeling is done with commercial UML tool called Telelogic Tau G2.
- The modeling uses the TUT-profile stereotypes and guidelines. TUT-Profile is not actually a tool but a set of modeling concepts and design rules. However, it is fundamental part of Koski flow. Detailed description of TUT-Profile can be found in [8] and MARTES Deliverable document D1.5.
- The functional verification is done by simulating the UML model (actually running the code generated from it). Currently, no automation for actual verification is used.

Optimization

- UML model is converted into Koski-specific format by in-house parsers [12]. The format is called XML System Model (XSM) and it is used between all TUT's tools.
- The system model is optimized with architecture exploration tools which are fully developed in TUT [13][14][15].
- Exploration includes both optimization heuristics and abstract system modeling. The system modeling is done Transaction Generator (TG) [16][17] that is mimics the system behavior with Kahn Process Network that is mapped and very abstract HW models. TG is implemented in SystemC.
- The Koski is controlled from graphical user interface (GUI). First version is implemented with Tcl Tk whereas the second is being implemented in Java.

Implementation

- SW code generation for from behavioral models in UML is performed by Telelogic Tau.

- SW generation phase includes also SW platform configuration and addition of TUT's SW platform components [18][19]. The SW platform allows distributed execution of applications in transparent manner: the application tasks need not to be aware which CPU executes the other tasks. The configuring task for the SW generation is implemented at TUT.
- HW generation mainly deals of interconnecting existing HW components, i.e. creating structural VHDL architectures. This is implemented at TUT. The VHDL codes are located in hw library.
- The soft Nios CPUs are configured using SOPC Builder by Altera Corp.
- HW synthesis is performed with Quartus by Altera Corp. For ASIC implementations, all implementation-oriented phases after RTL synthesis (placement, routing, layout, etc.) are beyond the scope of Koski.
- Downloading all parts (compiled SW, synthesized HW) into FPGA is done with Altera's tools. However, the procedure is automated with TUT's controlling scripts. The system may be executed also in virtual platform simulator instead of FPGA (this part is being developed currently)

Analysis

- Runtime monitoring of a MP-SoC is performed with TUT's tools.
- The performance results can be viewed at runtime with specific Execution monitor (by TUT), or after the execution via log files (for example in Matlab or Excel)
- Back-annotation is performed by TUT's tools [20].

Other

- Koski also uses the following sub-tools: Cygwin (only for Windows), Tcl (ActiveTcl is used for Windows), SystemC, Java, Nmake, Graphviz

4.3 Availability of Koski

Interfaces

Koski has interfaces for the Telelogic Tau G2 tool, SOPC Builder and Quartus FPGA synthesis tool by Altera Corp. The UML models are input and output with XML-based (XMI) format.

Availability

Koski is currently an internal university based toolset of TUT, and used in MARTES internally by TUT. Koski can be licensed for research purposes with separate agreements.

4.4 References

Koski flow and TUT-Profile

[8] Petri Kukkala, Jouni Riihimäki, Marko Hännikäinen, Timo Hämäläinen, and Klaus Kronlöf, "UML 2.0 Profile for Embedded System Design". In: Proceedings of 8th Design,

Automation and Test in Europe (DATE 2005), March 7-11, 2005, Munich, Germany, pp. 710-715.

[9] Tero Kangas, Petri Kukkala, Heikki Orsila, Erno Salminen, Marko Hännikäinen, Timo D. Hämäläinen, Jouni Riihimäki, Kimmo Kuusilinna, "UML-based Multi-Processor SoC Design Framework", Transactions on Embedded Computing Systems, May 1, 2006, Vol.5, Issue 2, pp. 281-320, ACM.

[10]Tero Kangas, "Methods and Implementations for Automated System on Chip Architecture Exploration", PhD Thesis, Tampere University of Technology, Publication 616, 2006, 181 pages.

Case study

[11]Petri Kukkala, Mikko Setälä, Tero Arpinen, Erno Salminen, Marko Hännikäinen, Timo D. Hämäläinen, "Implementing a WLAN Video Terminal Using UML and Fully-Automated Design Flow", EURASIP Journal on Embeded Systems, January 10, 2007, Issue Embedded Digital Signal Processing Systems" edited by Jarmo Henrik Takala, Shuvra Bhattacharyya, and Gang Qu., 15 pages.

Design steps and tools

[12]Jouni Riihimäki, Petri Kukkala, Tero Kangas, Marko Hännikäinen, and Timo D. Hämäläinen, "Interfacing UML 2.0 for Multiprocessor System-on-Chip Design Flow". In: Proceedings of 2005 International Symposium on System-on-Chip (SoC 2005), November 15-17, 2005, Tampere, Finland, pp.108-111.

[13]Heikki Orsila, Tero Kangas, Timo D. Hämäläinen, "Hybrid Algorithm for Mapping Static Task Graphs on Multiprocessor SoCs", International Symposium on System-on-Chip (SoC 2005), Tampere, Finland, November 15-17, 2005.

[14]Heikki Orsila, Tero Kangas, Erno Salminen, Timo D. Hämäläinen, "Parameterizing Simulated Annealing for Distributing Task Graphs on Multiprocessor SoCs", International Symposium on System-on-Chip 2006, Tampere, Finland, November 14-16, 2006, pp. 73-76.

[15]Heikki Orsila, Tero Kangas, Erno Salminen, Timo D. Hämäläinen, Marko Hännikäinen, "Automated Memory-Aware Application Distribution for Multi-Processor System-On-Chips", Journal of Systems Architecture, Elsevier, Accepted.

[16]Tero Kangas, Jouni Riihimäki, Erno Salminen, Kimmo Kuusilinna, Timo D. Hämäläinen, "Using a Communication Generator in SoC Architecture Exploration", International Symposium on System-on-Chip, Tampere, Finland, November 19-21, 2003, pp. 105-108.

[17]Kalle Holma, Mikko Setälä, Erno Salminen, Timo D. Hämäläinen, "Evaluating the Model Accuracy in Automated Design Space Exploration", 10th Euromicro Conference on Digital System Design, Lübeck, Germany, August 27-31, 2007, 7 pages, Accepted.

[18]Mikko Setälä, Petri Kukkala, Tero Arpinen, Marko Hännikäinen, Timo D. Hämäläinen, "Automated Distribution of UML 2.0 Designed Applications to a Configurable Multiprocessor Platform". In: Proceedings of 6th Embedded Computer Systems: Architectures, MOdeling, and Simulation (SAMOS VI), July 17-20, 2006, Samos, Greece.

[19]Tero Arpinen, Petri Kukkala, Erno Salminen, Marko Hännikäinen, Timo D. Hämäläinen, "Configurable Multiprocessor Platform with RTOS for Distributed Execution

of UML 2.0 Designed Applications", 9th Design, Automation and Test in Europe Conference (DATE 2006), Munich, Germany, March 6-10, 2006, pp. 1324-1329.

[20]Petri Kukkala, Marko Hännikäinen, and Timo D. Hämäläinen, "Performance Modeling and Reporting for the UML 2.0 Design of Embedded Systems". In: Proceedings of 2005 International Symposium on System-on-Chip (SoC 2005), November 15-17, 2005, Tampere, Finland, pp.50-53.

5 NXP

5.1 Objective

The objective of SCATE (Source Code Analysis and Transformation Environment) [1],[2],[3] is to provide a generic environment for fine grained C++/C parsing / analysis / transformation and hardware mapping tasks. The name environment is significant here. It means that the whole infra structure is a software platform which can help you to implement your own C++/C based analysis / transformation / mapping tasks. You are free to use it, but there where functionality is (still) lacking, you are invited to contribute to it as well. To allow new functionality to get in, SCATE is Open Source under the GPL License.

The functionality of SCATE that has been developed so far is a system-level design and programming environment for embedded multiprocessors. Here we consider the mapping of C++ based specifications of embedded systems functions onto heterogeneous multiprocessors containing hardware and software components. In practice this relates to a number of source code transformations on process networks, see Figure 4:. Transformations should be specified by a system designer. The enforcement of these design decisions in the source code is automated by the SCATE tool. The source code after transformation can - with or without additional manual modifications - be used as input for existing hardware and / or software compiler(s).

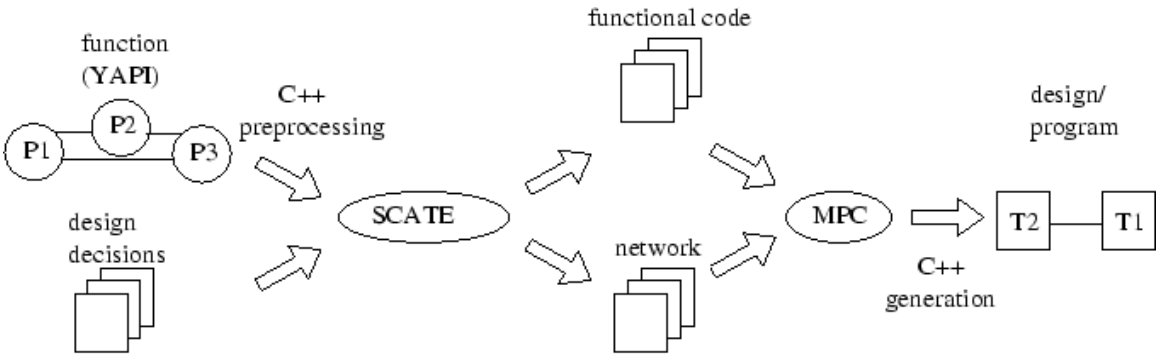


Figure 4: High level user view on the tools.

5.2 Features

- Provides an open source (ANTLR [4] generated) C++ parser.
- Implements a fine grained heterogeneous AST.
- Multi file compilation.

- Automatic recognition of YAPI [5] and TTL[1] [2] type process networks from C++ sources.
- Automatic file and directory restructuring. This provides one (consistent) software modeling style for all transformed application models.
- Automatic creation of Imakefiles for all transformed application models.
- Configurable API definition steers API recognition and transformations between API's. Moreover, it allows for the implementation of interface refinement. In order to recognize the network structure form sources the following restriction holds; the creation of the network should be implemented using a YAPI like syntax.
- There should be no perception difference between transformed source code, and manual (well written) source code; i.e. source code remains highly readable after transformation.
- (Optional) Graphical view on process network in dotty [6] format and / or SequoiaView [7] format.
- (Optional) Extraction of ecore meta-data of YAPI and TTL type process networks compliant to the MARTES [8] defined application meta-model (structure only, not the behavioral part; this still requires manual work).
- (Partial) Flattening of hierarchical networks.
- Conversion from C++ to C. The emphasis is not on a generic transformation for every language construct, rather we assume the main function of a process or task is rather close to the C language already. This requires an API definition in the C language.
- Integration with Qt [9] allows for the development of user interfaces.

5.3 Software infrastructure

The SCATE tool is created / cooperates with a number of other tools, see Figure 5:

- The C++ software infrastructure of SCATE has been generated using C3PO [10].
- For parser generation ANTLR is used.
- As a back-end tool for the process network generation MPC [11] is used.

5.4 Documentation

SCATE user manual (html [12] | pdf [13] | ps [14])
SCATE software infrastructure (html [15] | pdf [16] | ps [17])
SCATE demos of real examples (html [18])

[1] Task Transaction Level: a runtime environment will be made available as open source in the near future.

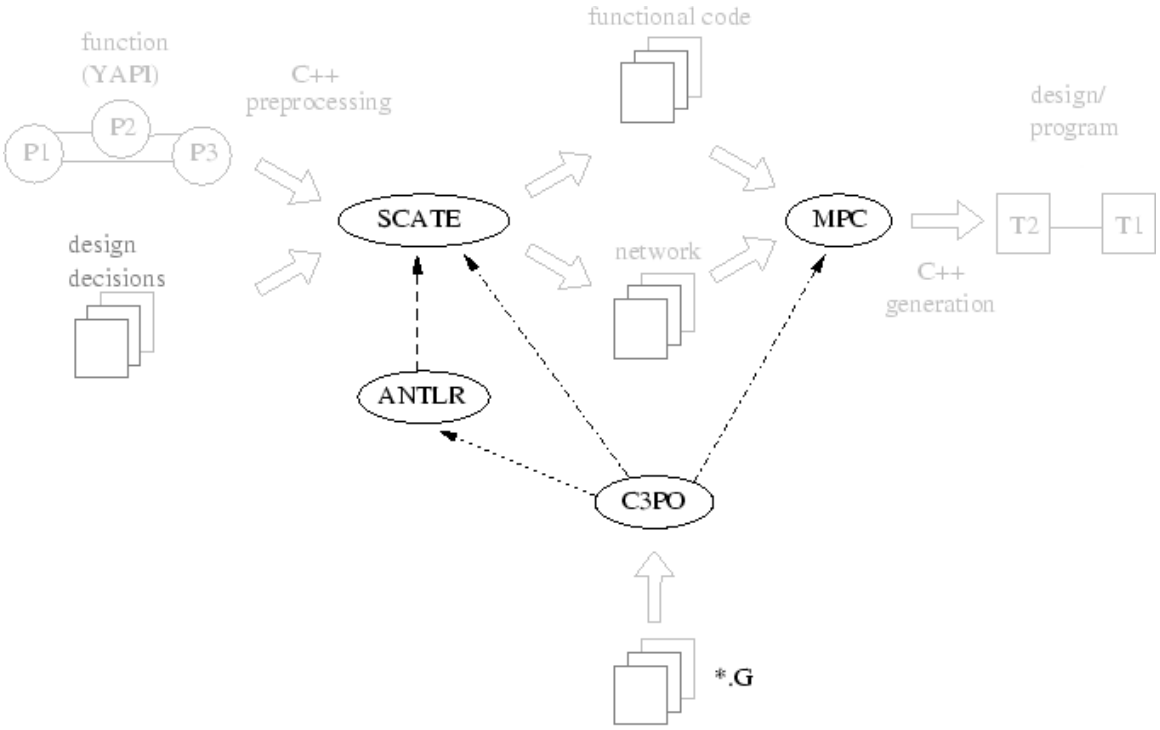


Figure 5: Component view from a SCATE tool designer perspective. The dashed dotted line represents data type generation. The dotted line represents the generation of input files for ANTLR. Finally, the dashed line represents parser generation.

5.5 References

[1] https://sourceforge.net/projects/scate
[2] Pieter van der Wolf, Erwin de Kock, Thomas Hendriksson, Wido Kruijtzer and Gerben Essink, Design and Programming of Embedded Multiprocessors: An Interface-Centric Approach, CODES ISSS, Stockholm, Sweden, September 8-10, 2004.
[3] Walter Tibboel, Victor Reyes, Martin Klompstra, Dennis Alders , System-Level Design Flow Based on a Functional Reference for HW and SW, *accepted at DAC*, In *Proc. DAC 06*, 2007.
[4] http://www.antlr.org/
[5] https://sourceforge.net/projects/y-api
[6] http://www.research.att.com/sw/tools/graphviz
[7] http://www.win.tue.nl/sequoiaview/
[8] MARTES is an official ITEA project (Jan 2005-Dec 2007): http://www.itea-office.org/
[9] http://www.trolltech.com/
[10] https://sourceforge.net/projects/c3po

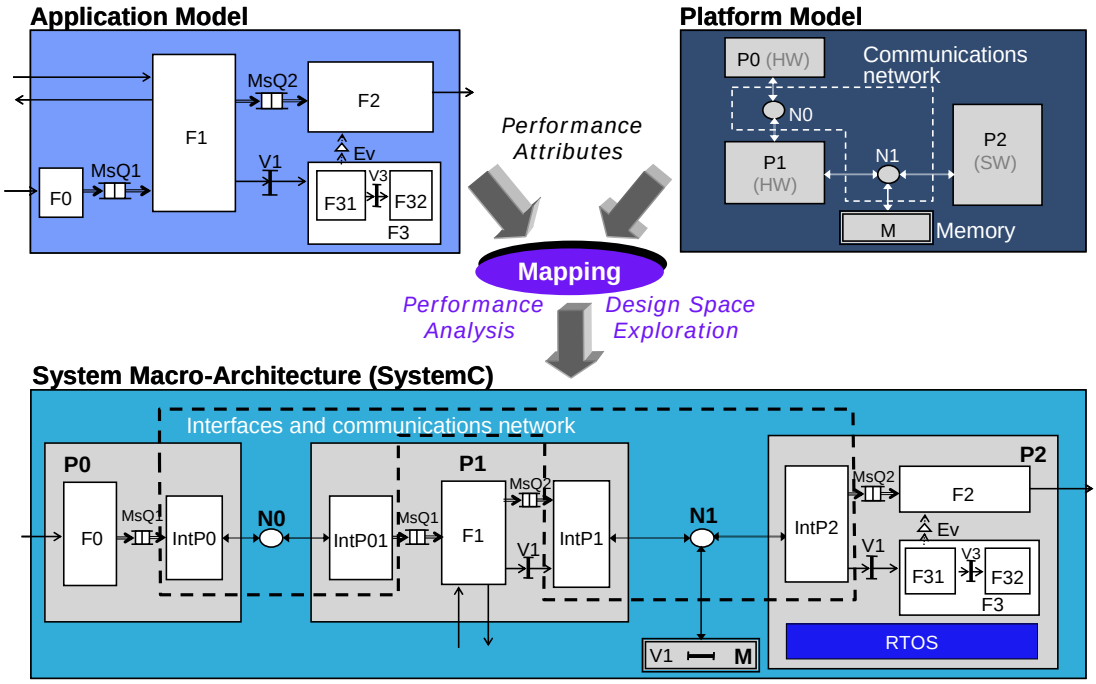
[11] https://sourceforge.net/projects/mpsc
[12] http://scate.sourceforge.net/manual/Scate_manual.html
[13] http://scate.sourceforge.net/manual.pdf
[14] http://scate.sourceforge.net/manual.ps
[15] http://scate.sourceforge.net/swinfra/Scate_software.html
[16] http://scate.sourceforge.net/triplem.pdf
[17] http://scate.sourceforge.net/triplem.ps
[18] http://scate.sourceforge.net/demos.html

6 CoFluent Design

6.1 CoFluent Studio Overview

CoFluent Studio is a visual **Electronic System-Level** (ESL) architectural development solution that enables performance analysis of complex hardware/software systems through a unique mapping technology. CoFluent Studio is used to develop the optimum system architecture at any stage in the development flow of an embedded system or System-on-Chip (SoC). Its high level of abstraction considerably simplifies and speeds architectural trade-off analysis by enabling fast modelling and high speed simulation.

CoFluent Studio is based on the *MCSE* methodology (French acronym for *Embedded Systems Co-Design Methodology*) and follows a "Y" design flow that separates the application from the platform modeling, to later merge them into an architecture model through a mapping operation. Mapping consists in allocating functions and functional communications of the application model to physical computation and communication resources of the platform model.



Behavioral application models are described using simple graphical notations with time parameters, and C/C++ code for data and algorithmic parts. Platform modeling is done independently by graphically assembling generic performance models of hardware components.

CoFluent Studio automatically generates SystemC code for simulation of the application and architecture models. It provides a rich set of observation tools for real-time and functional

validation, architecture exploration and performance analysis. Modeled performance indices include resource load, power consumption, memory footprint and cost.

From partial hardware and software descriptions only (no RTL code, no embedded software code, no firmware/OS, no ISS needed), CoFluent Studio allows:

- Early, agile and rapid system macro-architecture exploration and performance analysis by modeling the behavior of the application running on generic platform performance models instead of co-simulating hardware models with the real software code.
- Automatic generation of a SystemC testbench used for the system micro-architecture development & validation.
- Using CoFluent models as executable specifications by all project team members and to provide a blueprint for platform development and hardware/software implementation.
- Using CoFluent models to capture in libraries system-level functional and architectural IP components for capitalization and reuse.

CoFluent Studio available software packages are:

- **CoFluent Studio for Timed-Behavioral Modeling** (TBM) is targeted at specification/system engineer, application model designer, testbench developer and embedded real-time software developer. The TBM package covers application modeling and simulation only, including functional verification and real-time analysis. It provides automatic SystemC testbench and application code generation.
- **CoFluent Studio for System Architecting** (SA) is targeted at system macro-architects and embedded software architects. The SA package includes the TBM package and covers application, platform and architecture modeling and simulation. It provides architecture exploration and performance analysis capabilities; including analysis of software tasks scheduling and communication transactions.

More information on the CoFluent Studio software toolset can be found at http://www.cofluentdesign.com.

6.2 MARTES Goals & Objectives

MCSE and CoFluent Studio are based on a “Y” design flow that separates the system’s (functional) application viewpoint from the system’s (execution/physical) platform viewpoint. Both views are merged in a **mapping** operation to provide an abstract system architecture viewpoint.

The correspondence between CoFluent and MARTES terminologies is the following:

MARTES Terminology	CoFluent Terminology
Application Model	Application Model
Execution Platform Model	Platform Model
Allocation Model	*Internal: mapping rules (computations & communications)*
Allocated Model	Architecture Model
Implementation-Oriented Model	*Internal: C/C++ language syntax & semantics*

Already available transformations within CoFluent Studio are:

- Application model (to Implementation model) to SystemC, called "SystemC application generation"
- Application + Platform + Allocation models to Allocated model, called "architecture mapping"
- Allocated model (to Implementation model) to SystemC, called "SystemC architecture generation"

CoFluent Studio totally adopts and already supports most of the MARTES concepts and methodological aspects:

- **Model-based architecting**: CoFluent models help users explore architecture alternatives and obtain prospective performance results, including time, cost, power consumption and memory footprint.
- **Reuse models rather than implementations**: CoFluent Studio users define independent application and platform models, and can reuse them in libraries.
- **Utilize model transformations**: CoFluent Studio automatic transformations and code generation capabilities.
- **"Y" design flow**: CoFluent Studio already supports the "Y" design flow, and a direct one-to-one correspondence can be made between CoFluent and MARTES models.

Hence, CoFluent Design focused on the interoperability through the commonly-defined and shared meta-model for tool interoperability and model interoperability.

More information on the MCSE methodology can be found at http://www.cofluentdesign.com.

6.2.1 Model interoperability

Based on its extensive experience and proven commercial solution already implementing a co-modeling methodology very close to the MARTES approach, CoFluent Design actively participated in the definition of the MARTES meta-model. Thus, the MARTES meta-model and CoFluent meta-model are based on similar concepts.

Element of MARTES Application Meta-Model	Equivalent Concept in CoFluent
Application	Application
AppComponent	Component
AppAttribute	Attribute
AppModule	Function
AppConnector	*Implicit*
AppStructure	Structure
AppCommunicationNode	Functional Relation
AppSharedVariable	Shared Variable
AppMsgQueue	Message Queue
AppEvent	Event

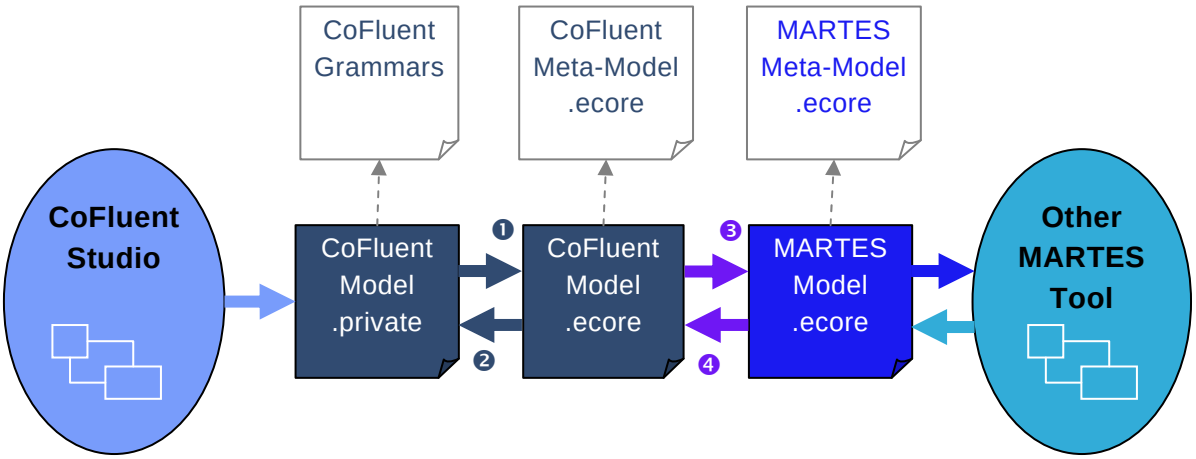
AppCompInterface	Interface
AppPort	Port
AppTemplateParameter	Generic Parameter
AppBehavior	Behavior
AppActivity	Activity
AppOperation	Operation
AppInputCondition	Input Condition
AppOutputAction	Output Action
AppConcurrentActivity	Concurrent Activity
AppDeclaration	Declarations
DataType	Data Type
AppData	Data

CoFluent Design developed internal (private) CoFluent Ecore meta-models by transcribing its internal grammar-based CoFluent-specific meta-models into an Ecore CoFluent-specific meta-model.

For each existing CoFluent Application grammar, CoFluent Design created the equivalent Ecore meta-model. Then it used LINA's ATL (Atlas Transformation Language, www.sciences.univ-nantes.fr/lina/atl/) from the University of Nantes to provide a bidirectional transformation path from grammar-based models to internal CoFluent Ecore models using the TCS (Textual Concrete Syntax) technology.

6.2.2 Tool interoperability

A bidirectional bridge is needed to link the CoFluent technical space to the MARTES technical space so MARTES users can use the advanced modeling and simulation techniques offered by CoFluent Studio.



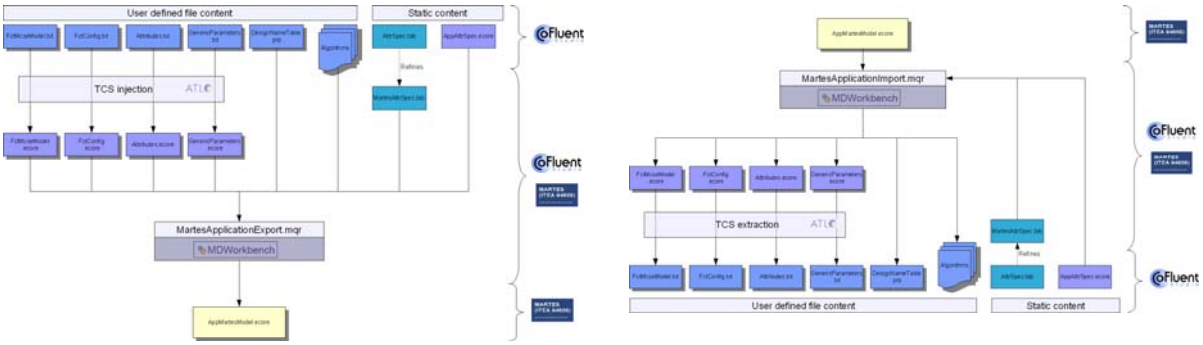
As CoFluent Studio already provides internal representations of all models and most of the transformations defined in MARTES using its own notations and tools, CoFluent Design focused on CoFluent-to-MARTES transformations in a two-step process:

- The first step required offering an Ecore input/output interface to CoFluent Studio: transforming from grammar-based CoFluent-specific models to Ecore CoFluent-specific models ❶ (and vice-versa ❷)
- The second step requires adapting MARTES Ecore meta-models to CoFluent Ecore meta-models: transforming from Ecore CoFluent-specific models to Ecore MARTES models ❸ (and vice-versa ❹)

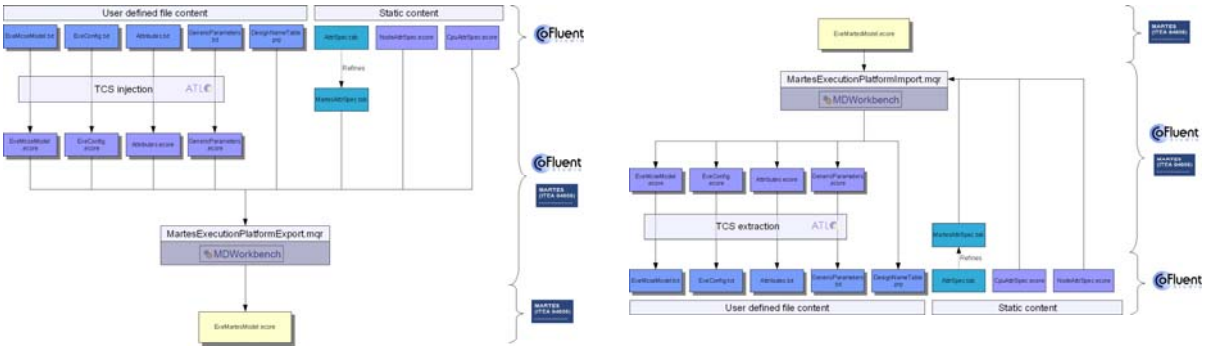
The four transformation paths *CoFluent.private* ↔ *CoFluent.ecore* ↔ *MARTES.ecore* for the Application, Execution Platform and Allocation MARTES meta-models was developed and integrated (as an option available to MARTES partners) into the CoFluent Studio v2.1.0 product release in September 2007.

Since the MARTES and CoFluent Application meta-models share similar concepts, CoFluent.ecore ↔ MARTES.ecore transformation rules (paths 3 and 4 in the above figure) were quite obvious to define. The export operation (path 3: from CoFluent model to MARTES) was achieved using the ATL Development Tools (ADT) 1.0. The import operation (path 4: from MARTES model to CoFluent) was achieved using Sodius MDWorkbench product (www.mdworkbench.com).

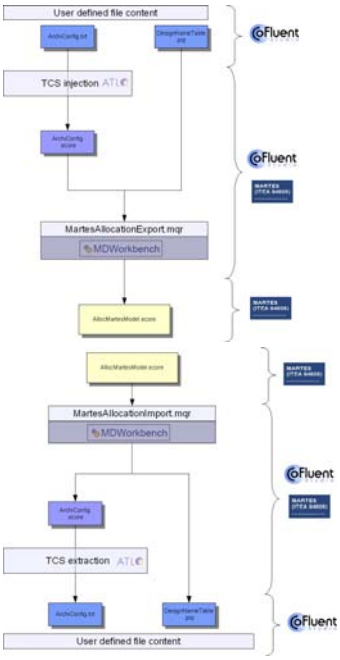
- **CoFluent-to-MARTES Export & Import: *Application***



- **CoFluent-to-MARTES Export & Import: *Execution Platform***



- **CoFluent-to-MARTES Export & Import: *Allocation***



6.3 CoFluent Studio MARTES Import/Export Software Package

Software offering MARTES import/export from CoFluent Studio was developed and integrated to the CoFluent Studio v2.1.0 product released in September 2007 as a dedicated **"Import/Export EMF" tool panel**.

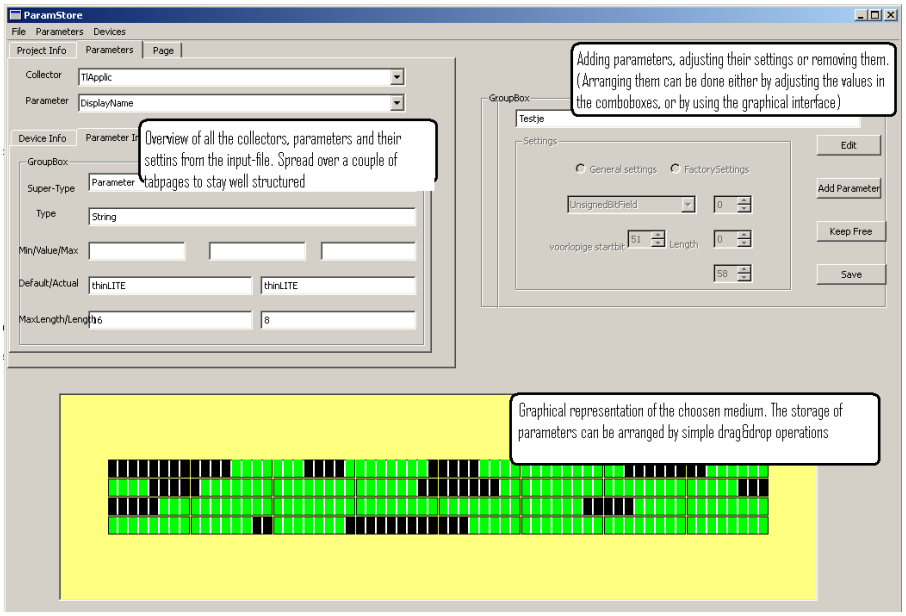
The CoFluent Studio "Import/Export EMF" tool panel provides a simple framework to link CoFluent projects to various modeling spaces. Currently, only two modeling spaces are considered: the CoFluent modeling space consisting of Ecore models created from CoFluent (grammar-based) project text files and the ITEA-MARTES modeling space containing models that conform to MARTES meta-models.
Both modeling spaces are related to meta-models that conform to the Ecore meta-meta-model from the Eclipse Modeling Framework (EMF).

The panel is divided into three areas:

- Project text files related to applicable transformations are presented on the left area.
- Ecore files available in the current workspace file structure are presented on the right area. A default "Project" tab-panel represents Ecore files in the current CoFluent project file structure. Custom tab-panels can be also added to define new import/export workspaces.
- Buttons to import or export files are available in the middle area.

7 Barco

One of the tools that have been developed within the scope of the Martes project is dealing with the domain of persistent storage and compatibility of this storage between software upgrade/downgrades. Having an offline tool to define the exact position of the storage items leads to more consistency during code generation/transformation and result is more automation during development.



The outcome of this tool is used during the model transformation & code generation to obtain a predefined and consistent storage layout. As this is kept under configuration control, we can monitor the changes throughout the design process and identify conflicts before they actually occur.

Other tools involve the definition of the hardware/electronics platform with respect to memory map, flash layout, peripheral, … and the definition of the (register) interface between software and (programmable) hardware.

8 K.U. Leuven

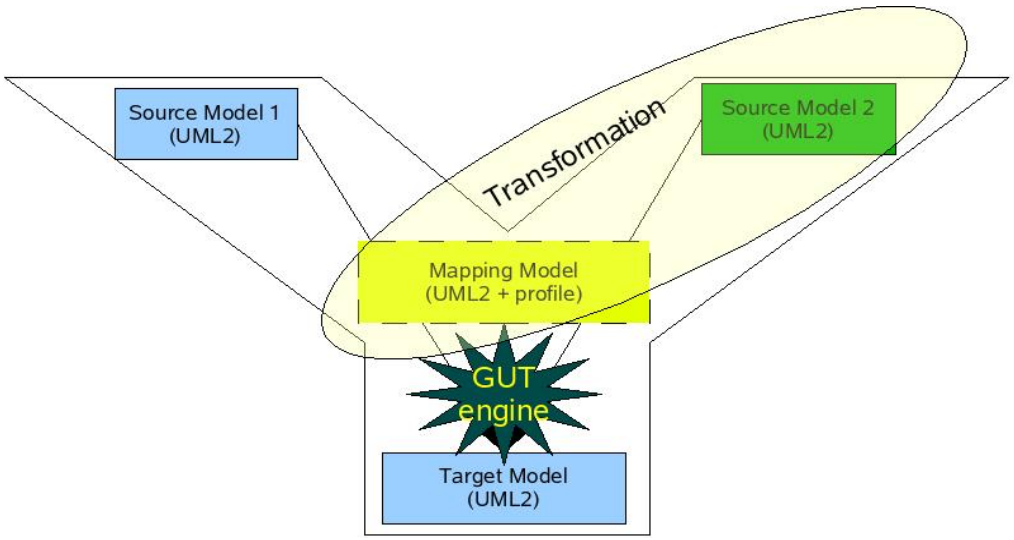
This chapter presents the Generic Upsilon Transformations (GUT) approach to model transformations that was developed by K.U.Leuven.

8.1 Overview

GUT represents a novel approach where each model transformation is realized by a merge of two models. GUT can be used to implement a certain class of model transformations: vertical transformations which introduce design patterns to the original source model. Currently GUT can only realize structural transformation. As a future work we will investigate to extend GUT to support also the behavioural aspect of design patterns.

Each transformation in GUT is represented as the Greek letter upsilon, hence the name of the approach. The left and right branches of the upsilon contain two source models. Along with the two source models we also provide a mapping model that instructs the model transformator how particular elements from the Source Model 1 map on elements from Source Model 2. Source Model 1 and 2 are standard UML2 class models. The Mapping Model is a UML class model along with a profile which realizes the linking of the corresponding elements from the source models. In order to realize different mapping strategies, the mapping links are stereotyped and may use tagged values for additional mapping parameterization. The Mapping Model is not self-contained as opposed to the source models, i.e. it is useful only if the related models exist as well.

Elements that are not referenced in the Mapping Model are copied to the Target Model. The rest of the elements are modified by the transformation engine according to the mapping they are involved in. From these three input models, using a generic transformation engine, we can obtain a generated output Target Model



8.2 GUT Mapping Strategies

Based on the GoF design patterns and a repository of security patterns we have devised a complete set of mapping strategies which can be used to apply these patterns on a source model.

8.2.1 Filter

Certain patterns might change the association structure between given entities. For instance when a proxy is placed between two communicating entities their direct association is replaced by an indirect association via the proxy. We introduce a filtering strategy which inserts a filter class or a set of classes between two source entities. To use this strategy, we need to place a <<*filter*>> dependency link from the association between the source entities to the desired filter class or set of classes. The transformation engine inserts the filtering class(es) between the source classes by deleting the source association and creating two new associations.

8.2.2 Merge

We have identified two different cases for which we need to merge two entities. a) two source models may contain elements which actually represent the same entity from different points of view. Often we need to merge these elements in order to obtain an entity that combines the properties of the original two ones. b) Patterns often contain classes which serve as a parameter and should be always bound. These parameter classes define certain properties which are necessary for the pattern to work (e.g. specific attributes or operations). We can bind the parameter classes with concrete ones by merging the two.

We introduce a merging mechanism that creates a new entity by combining the properties of two source entities. In order to realize the merging strategy we should place a <<*merge*>> marked association between two to-be-merged classes. The default name of the new entity is a concatenation of the names of the original classes (starts with the name of the entity on Source Model 1). We can however override it by naming the <<*merge*>> marked association.

8.2.3 Relate

We add to our approach the possibility to relate elements from the different source models. This can be realized by placing a standard UML relationship link (e.g. association, generalization, aggregation etc.) between the desired two elements in the Mapping Model.

8.2.4 Rename

It is useful to have a mechanism to rename elements during a pattern instantiation or model merge. Doing so manually on the Target Model has a number of obvious disadvantages: traceability is lost, subsequent iterations will always cancel the manual changes.

We introduce a simple renaming mechanism which can be applied to any entity (class, attribute, operation etc.) from the source models. In order to instruct the GUT engine to rename an element we need to place a dependency link in the Mapping Model from that element to itself and mark it with the <<*rename*>> stereotype. The name of that dependency link will be used as a new name for that element in the Target Model.

8.2.5 Remove

We provide also an additional construct to remove elements from the source models. In case we wish to remove a given element from one of the source models we should place a dependency link to from that element to itself and mark it with the <<*remove*>> stereotype. This dependency link should be placed in the Mapping Model.

8.2.6 Copy/Move

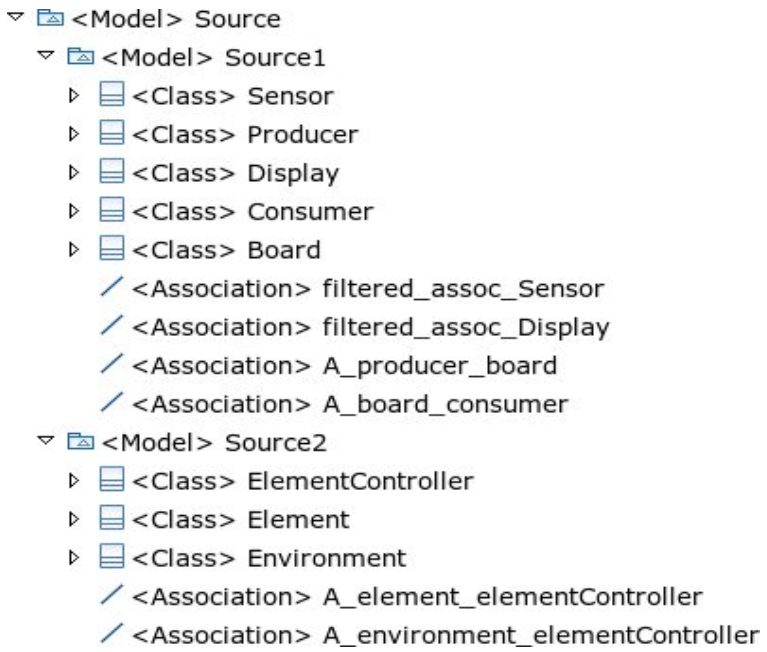
Some patterns may require moving operations or attributes from one class to another. Other patterns may contain template properties which should be instantiated with concrete ones from a source model. We introduce a property mapping mechanism which allows us to move and copy any attribute or operation from one class to another.
In order to instruct the GUT engine to copy a given property to another class we should place a dependency link from that property to the class. Moreover we should mark the dependency link by the <<*copy*>> stereotype. To move a property we should mark the dependency link by the <<*move*>> stereotype. Template property instantiation is realized using the same notation however in this case we should place a dependency link from the source property to the template property (instead of the class). By using the <<*move*>> stereotype the source property will be moved instead of copied. All this information should be placed in the Mapping Model.

8.3 GUT Usage

In this section we will discuss how to use the GUT engine. GUT is implemented on top of ATL hence the toolset used here is Eclipse along with the ATL and EMF plugins.

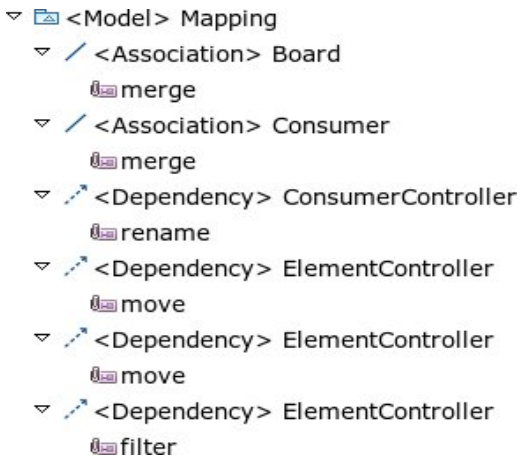
8.3.1 Source Models

For the sake of concreteness we will assume that Source Model 2 always contains the model of a pattern while Source Model 1 is the real source model.
First of all the two source models should be created and saved in an Ecore compliant format. Ideally given a source model (represented as Source Model 1), the transformation user will have a choice of a pattern library where he can select a pattern which should be applied on the source model. This pattern is then represented as Source Model 2. Both source models should be imported in Eclipse workspace where the GUT engine can access them.



8.3.2 Mapping Model Specification

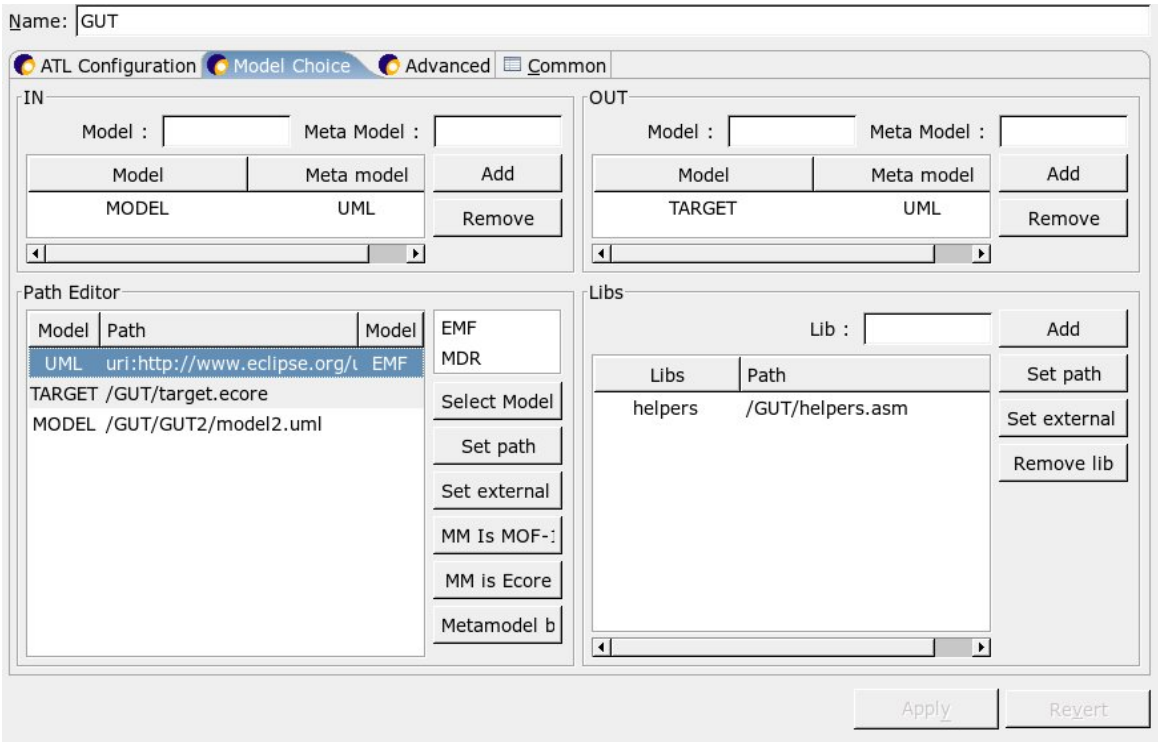
In the next step we have to add a mapping model which specifies how the pattern is applied on the source model.



8.3.3 Executing GUT

Once the two source models and the mapping between the two is specified we can run the generic transformation on these input models. We need to configure the GUT engine so it takes the correct input model and we also need to specify where the target model should be

produced. The following screenshot illustrates how this should be done.



When we hit the Run button the GUT engine generates the target model. The generated model is compliant with the Ecore format and can be used in further transformations.

8.4 Model and Transformation restrictions

The Generic Upsilon Transformations (GUT) approach enables to define model transformations for any kind of model views. Therefore, all models defined in the MARTES technical space can be used in the GUT approach. One must take care that the transformations defined in GUT respect the mapping rules between the MARTES models. However, this is considered to be the responsibility of the person(s) defining the transformations.

The Generic Upsilon Transformations (GUT) approach enables to define model transformations for any kind of model views. This allows following model transformations:

- Application model to application model transformations
- Execution platform model to execution platform model transformations
- Application-execution platform-allocation model to allocated model transformations
- Allocated model to allocated model transformations
- Allocated model to implementation model transformations
- Implementation model model to implementation model transformations

When the requirements model can be expressed using an UML class diagram, the following two model transformations are also possible using GUT:

- Requirements model to application model transformations
- Requirements model to execution platform model transformations

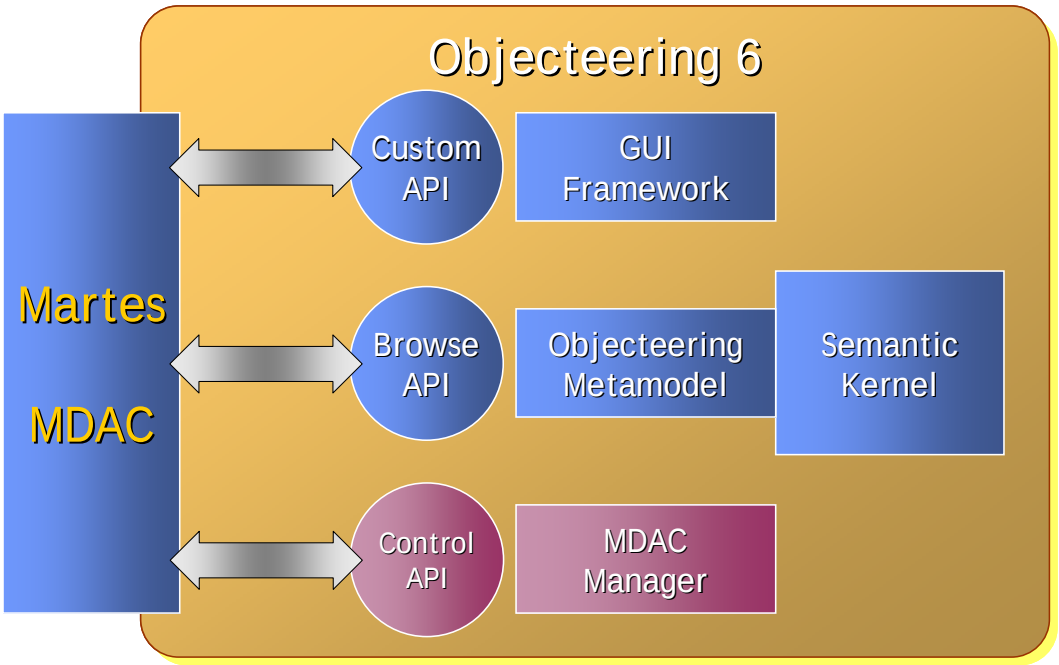
Since the Generic Upsilon Transformations (GUT) approach enables to define model transformations for any kind of model views, there is no need for a specific implemntation of the MARTES profile in order to enable the usage of GUT. GUT can be used to transform any class model expressed in Ecore, therefore GUT can be used for the transformation of a model using the MARTES profile

9 Softeam

9.1 Objecteering Overview

Objecteering is a complete and powerful framework for UML modelling activities and fully supports UML extensions (profiles) and the MDA approach. Extensibility inside Objecteering is realised by the MDAC technology and the Java Native Objecteering Interface – a complete and powerful API to browse and modify UML models inside Objecteering.

The following figure presents a very simple overview of the Objecteering-MDAC coupling.



On the Objecteering platform, MDACs may be considered as plug-ins bringing new features to the Objecteering UML modeler. From 2007, MDACs can be completely developed in Java language using MDA Modeler – MDA Modeler is the Objecteering MDAC development tool which is an MDAC itself.

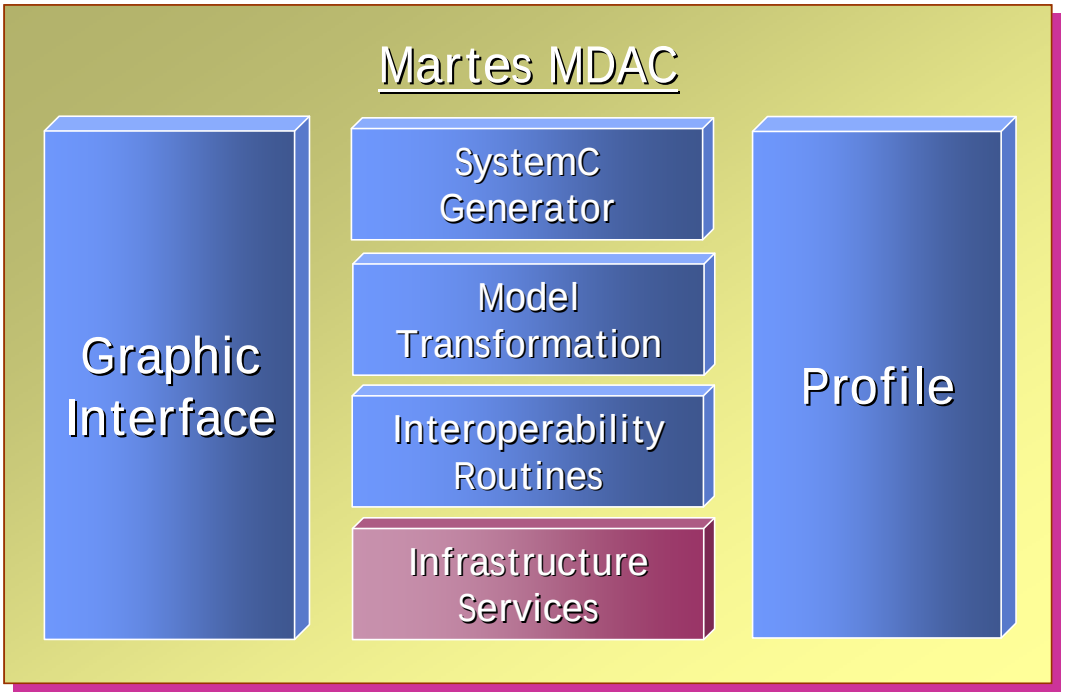
Today, the entire Objecteering MDAC collection provides all the software development team needs to design software applications from requirements management to source code generation for multi-language targets.

9.2 Objecteering in the MARTES context

The new features expected by the MARTES goals are naturally hosted by a brand new MDAC – the Objecteering/MARTES MDAC which was completely developed during the Martes project. The features supported by the MDAC are:

- modelling embedded systems respecting the “Y” design flow and MCSE methodology
- model interoperability using the EMF/Ecore technology
- multi-level SystemC source code generation

As described above, the Objecteering/MARTES is organised in many components as shown in the following figure.



The « **Graphic Interface** » component includes the graphic icons used in the dedicated toolbar and the UI commands linked to Martes functions.

The « **Profile** » component contains all UML extensions required to support the Martes terminology, such as stereotypes, tagged values.

The « **Interoperability Routines** » component provides the import/export services based on the EMF/Ecore technology.

The « **Model Transformation** » will further contain the implementation of model transformation rules used in the Martes workflow.

The « **SystemC Generator** » component includes all the functions required to generate the SystemC source code according to the three level defined by Thales Communication.

The « **Infrastructure Services** » contains the Objecteering technical platform services needed to plug the Martes MDAC to the Objecteering framework.

9.2.1 Co-Design approach with the MARTES MDAC

The Co-Design approach usually follows a “Y” Cycle where software application and hardware architecture are separately modelled in a preliminary step and then later joined to a unique allocation model.

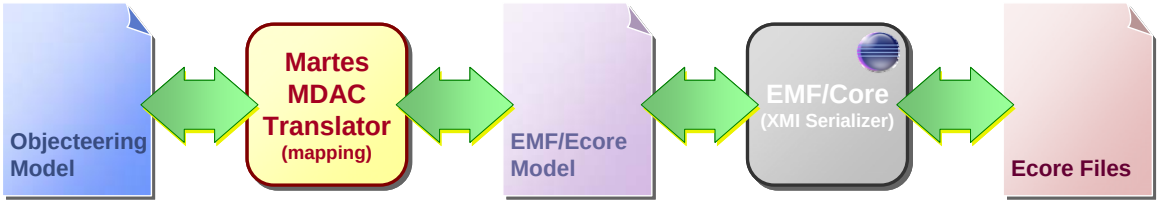
The Objecteering/MARTES MDAC obviously respects this principle by providing a model structure according to this separative approach. The application is first modelled in a specific package of the model and then the Execution Platform model is added into the appropriate part of the model. Further the allocation is realised by ensuring a simple mapping operation through an allocation diagram.

The allocation mapping simply consists in dragging dependency relations between Application elements and Execution Platform elements to define how the software is implemented on the hardware. The project model structure allows multiple definitions of allocation which is a very convenient feature to realise many allocation explorations.

9.2.2 Model interoperability

As explained previously the interoperability component provides the import/export functions to exchange models between tools through the Martes metamodel. The interoperability process implemented in the Objecteering/MARTES MDAC relies on the EMF framework to benefit from the Ecore format and the core services (XML serialisation).

As shown in the figure below, the translator operation ensures the bidirectional model transformations between the Objecteering internal metamodel and the Martes Ecore representation. XMI serialization is managed by the EMF/core services.

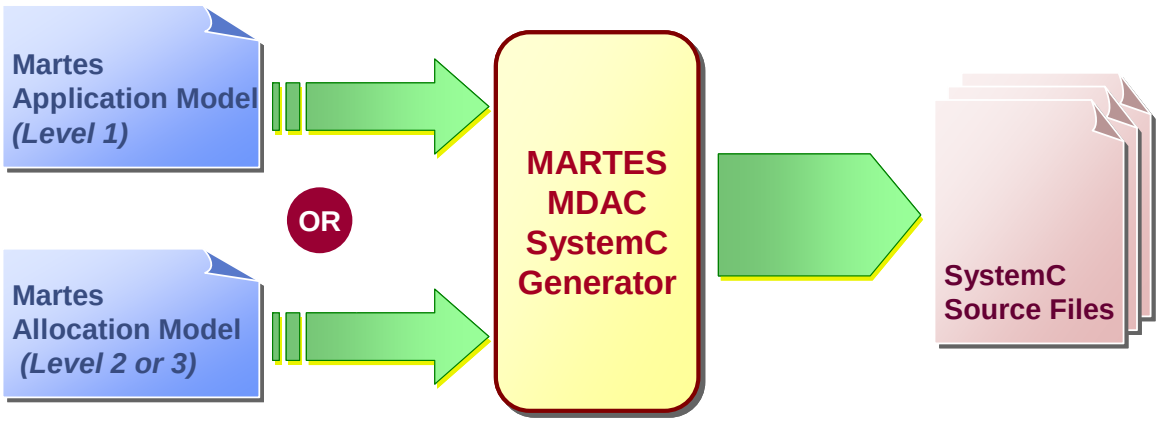


9.2.3 SystemC generation capabilities

The Objecteering/MARTES MDAC includes a multi-level SystemC generator which supports the Martes methodology. Depending on the selected element in the model and the place where the user launches the SystemC generation command, the appropriate level is used to drive the source code generation.

In a similar way for traditional language generator such as Java for example, the Martes generator browses into the user model to extract information and generates the source code accordingly. All source files are generated in the same directory which is specified by the user when he launches the command.

As UML notes are internally used to store some information such as attributes or string expressions content, obviously they are processed and analysed by the SystemC generator to produce the appropriate source code.



9.3 Objecteering/MARTES packaging

The MARTES MDAC packages the following software elements:

- ***Martes.mdac*** contains the definition of UML extensions (UML profile) used in the MARTES MDAC.
- ***MartesFramework.jmdac*** contains the java code for all new features provided by the MARTES MDAC (interoperability, SystemC generation, ...).

A short document provides some information regarding the technical requirements, the installation procedure and the user manual for the Objecteering/Martes MDAC.

The Objecteering/MARTES MDAC will be available in 2008 as a free Objecteering add-on and simply downloadable from the Objecteering web site.

9.4 Using Objecteering/MARTES MDAC

9.4.1 Graphical symbols

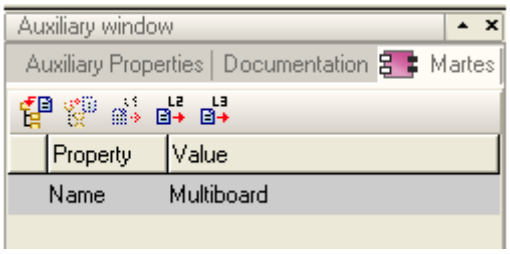
The table below describes the graphical symbols defined in the Martes profile and used to represent the Martes model elements in Objecteering.

	AppModule		AppSequenceActivity		XpRessource
	AppCommunicationNode		AppOperation		XpSpecialisedResource

	AppMessageQueue		AppLoopActivity		XpProgrammableResource
	AppSharedVariable		AppConditionalLoop		XpStorageResource
	AppSyncArray		AppMultiplicationLoop		XpCommManagerResource
	AppEvent		AppDivisionLoop		XpSupportingResource
	AppActivity		AppInfiniteLoop		XpBus
	AppConcurrentActivity		AppInputCondition		XpRouter
	AppConditionalActivity		AppOutputAction		XpPoint2Point

9.4.2 MDAC User Interface

As detailed in the Objecteering/MARTES MDAC user manual, the main commands are grouped in the toolbar of the Martes tab in the Auxiliary window.



Commands available in the toolbar are the following (from left to right):

- import a model from Ecore file
- export a model to Ecore file
- SystemC generation for level 1, 2 and 3

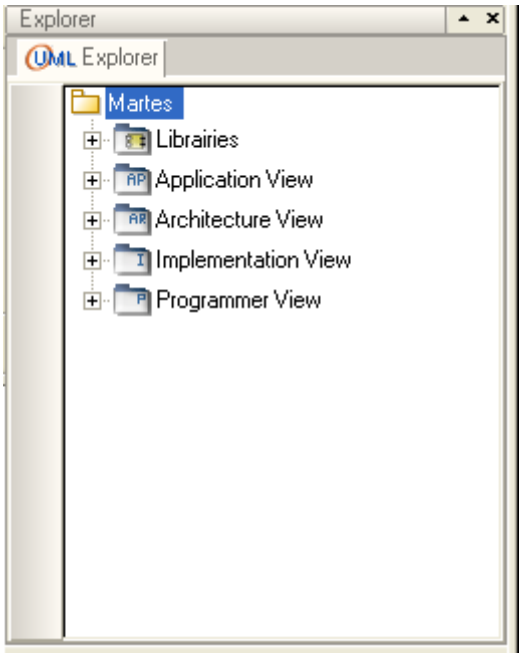
9.4.3 Deploying an Objecteering MDAC

Before using an Objecteering MDAC, the MDAC has to be deployed into an Objecteering base. To deploy an MDAC, just launch the "`Tools/Deploy an MDAC`" command menu and select the appropriate MDAC.

In the case of MARTES MDAC, first deploy the *Martes.mdac* file and then the second one, *MarteFramework.jmdac*.

9.4.4 Creating a new Martes project

When the Objecteering/MARTES MDAC is deployed, a new Martes project is automatically created with the following initial structure including the four main views defined in the Martes approach.



The ***Application View*** only contains the application model describing the organisation and the behaviour of the user application. This view supports the first level of SystemC generation.

The ***Architecture View*** contains both Application model and also Execution Platform models. For each architecture defined in this view, an allocation diagram describes the mapping between Application elements and Execution platform elements. Many allocation diagrams can be defined for one architecture allowing an explorative approach to find the best one. This view and more precisely the allocation diagram drives the SystemC generation process for levels 2 and 3.

The ***Implementation and Programmer Views*** also contain Application, Execution Platform model and allocation description. The main differences with the Architecture view concern the properties defining the model elements which are generally more detailed, and that will produce a different SystemC source code.
Unfortunately it was not possible to develop these two views inside the Martes project time frame. Specifications of these views still remain to define. Depending on the Softeam strategy regarding the Objecteering/Martes MDAC, they could be provided later in 2008.

10 INRIA

10.1 Syntol

INRIA is developing the Syntol tool, whose ultimate aim is to transform the specification of a process network into a compilable and/or synthesizable model. The basic paradigms and level of detail of the Syntol specification are comparable to those of the MARTES application model, with some differences which will be discussed later.

10.1.1 Syntol Input

Syntol has textual input. The initial specification is in the form of a process network, expressed in an *ad hoc* programming language, CRP (Communicating Regular Procsses). CRP is an extension of C with a few extra keywords for specifying processes, channels and ports. CRP is similar to NXP's YAPI. However, the communication abstraction of CRP is based on write once / read many arrays instead of the infinite FIFOs of YAPI and Kahn Process Networks. The resulting model is nearer to the expectations of embedded system designers and allows full compile time analysis.

10.1.2 Syntol Output

The main method of the Syntol tool is *scheduling.* A schedule is a function which assign a date and place to each operation of the input model. In other words, scheduling translates an application model into an allocated model, and this transformation is entirely automatic, provided the input specification is sufficiently regular.

In the present version, Syntol will be mainly useful for checking the coherence of the model. Syntol can check that the communication network is compatible with the usual design rules: each channel has exactly one writer and at least one reader. Deadlocks can be detected whether they are due to faulty topology or to insufficient buffer space. In a future extension, Syntol will also detect accesses to undefined variables or channel slots.

When a schedule exists, one may extract from it many useful information:

- minimum size of arrays and channel buffers
- degree of parallelism and parallel code
- communication code

Temporal information is also available in the form of an abstract schedule. Its interpretation in term of real time is a subject for future research.

10.2 Interoperability

INRIA has implemented a gateway which extract CRP code from a MARTES model. This gateway is written in Java under the Eclipse environment. While still experimental, it is powerful enough to translate simple models and to exhibit some of the most interesting features of Syntol, like the detection of deadlocks, the sizing of communication buffers and the memory / parallelism trade-off.

Since the position of low level code in the MARTES model is not rigidly fixed, this gateway exists in two versions, one of which follows the Cofluent conventions, and the other those of SPEAR DE. The last one is able to enrich the MARTES model with the results of Syntol analysis in a form suitable for use by SPEAR DE.

10.2.1 Martes to Syntol

Some of the MARTES concepts have direct correspondence in CRP:

MARTES	***CRP***
AppModule	Process
AppSyncArray	Channel
AppCompInterface	process prototype
AppPort	Port
AppDeclaration	Declaration
AppData	Data
DataType	CRP type
AppBehavior	process body
AppActivity	data processing code
AppLoopActivity	Loop

The AppStructure component of MARTES has no direct equivalent in CRP. The same effect can be obtained by constructing a “main” process in which channel declarations and process activation statements are collected.

The communication network structure is represented in MARTES by AppNodeConnectors, which relate on one side to AppPorts and on the other to AppCommunicationNodes. In MARTES, processes cannot be replicated (or, rather, have to be replicated by hand outside the model). Hence, there is only one activation statement per process. This statement is constructed by following the links from a process to its ports, then to its connectors and lastly to communication nodes.

The advantage of this approach is that very few TextualExpression attributes have to be inserted in the model:

- In AppActivity, for representing the computational body of each process;
- In DataType, for specifying the data structures which are associated to local data objects, ports and channels.

- In AppCommunicationNodes, for specifying the size of FIFOs and buffers (if this is a constraint of the model). Alternatively, Syntol has the ability to compute minimum sizes of channels, which can be returned as an attribute of the corresponding AppCommunicationNode.

10.2.2 Syntol to SPEAR DE

The theoretical work, which was started in 4Q 2006 has been finalized. The analysis, which was initially restricted to programs in the polytope model, has been extended to more general situations with the help of techniques from computer algebra. The class of programs which can be precisely analysed has thus been considerably extended. Since this work seems to be of interest to several teams (ARMINES, LIFL), a research report has been posted in the INRIA Open Archives under number 6193. The method has been implemented as an extension to Syntol and gives satisfactory results.

However, some of the programming rules at TRT (especially the use of array descriptors) are difficult to understand for Syntol and need tedious manual rewriting. This effort can be avoided at the price of a more sophisticated semantical analysis, which will be the subject of future work.

11 University of Cantabria

In MARTES several Domain Specific Models have been proposed. One of them has been defined by GMV as an extension to critical software applications. This model is an extension of the MARTES Domain Specific Model that was described on deliverable 1.5, “Specification of the Models (PIM, PSM) in the MARTES Methodology”.
UC has developed a toolset that provides simulation and performance analysis to the GMV Domain Specific Model. The toolset also includes tools that transform the GMV model into the e-core representation that is described on Deliverable 2.1, “Detailed Definition of the different models views and their mapping”. This allows integrating the GMV Domain Specific Model into the MARTES framework.

11.1 Design steps

The design flow with the GMV and UC toolsets integrate several tools. These tools define a complete framework from specification to software implementation. The toolset can be summarized as follows:

- Modelling: GMV has developed a UML tool that allows capturing the different model views: application, execution and allocation models. This tool is a plug-in of a commercial UML framework (Borland Together). The GMV extension allows generating XML code from the UML model. This code is used by the UC toolset.

- Interoperability: In order to allow that other tools can use the model, the UC toolset transform the GMV model into a MARTES model (described on deliverable 1.5). The developed tool is integrated into the open-source Eclipse framework, thus the transformation result is a MARTES e-core.

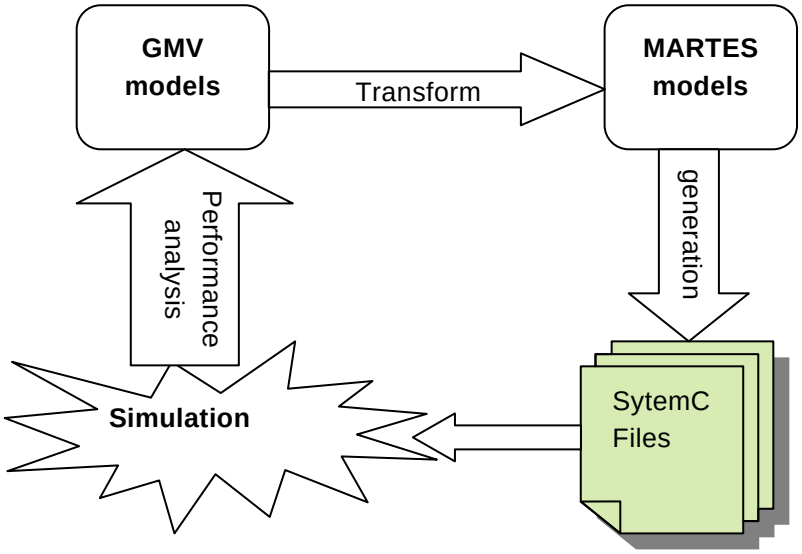


Figure 2. UC design-flow overview

- Simulation: UC has developed several generators that transform the MARTES model into a SystemC description. These generators allow simulating the GMV application model.

- Performance Analysis: The toolset includes specific tools that allow a SystemC-based performance analysis of the GMV model.

The figure 1 shows an overview of the design flow.

11.2 Model and Meta-model definition

The GMV model is defined with a visual UML modelling tool: Borland Together Architecture. The tool allows designing and implementing flexible and maintainable software architectures that can be easily modified as requirements change. The UML models that are generated with the tool are exported to specific XML files. These files include all the model information.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<Description xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
<model type="Application Model">
    <MetaModelObject name="PDS EV2" description="EV2 block">
        <ProcessingObject period="100" deadline="0" executionTime="10" worstCaseCompletionTime="15">
        </ProcessingObject>
    </MetaModelObject>
    <MetaModelObject name="ServerSocket2_1" description="Multiport socket 2">
        <interfaceObject codeAreas="VECTOR">
            <MultiPort listenPort="0">
                <Ports type="ports used" isSynchronous="False" isTransactional="False" direction="In" portUsed="0">
                </Ports>
            </MultiPort>
        </interfaceObject>
    </MetaModelObject>
```

Figure 3. GMV model example

There is not a common MARTES e-core source code. In the UC toolset, the MARTES Domain Specific Model has been defined on UML as a meta-model. This meta-model has been captured with the open-source Papyrus UML2 environment (www.papyrusuml.org) and it was exported to e-core format.

11.3 Model transformations

The UC toolset is able to transform GMV models into MARTES models. The GMV model is an extension of the MARTES model, thus some GMV model primitives have to be transformed into specific MARTES structures with some new parameters.

The toolset includes a program that is able to read the XML files that Borland Together generates. This program transforms the original XML file into a new XML file that can be imported into the Eclipse e-core, as next figure shows.

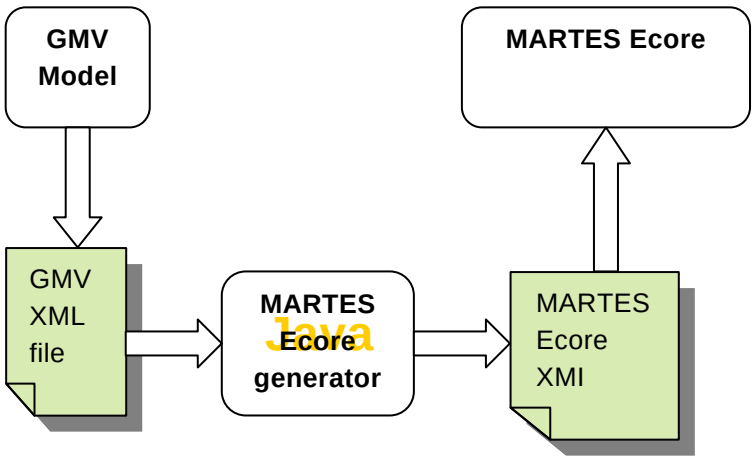


Figure 3: UC Toolset model transformations

This MARTES Ecore can be exported to other frameworks and tools. In the UC toolset, this model can be edited with the E-core Eclipse Editor. Figure 5 shows an example of a GMV model that has been transformed into MARTES e-core.

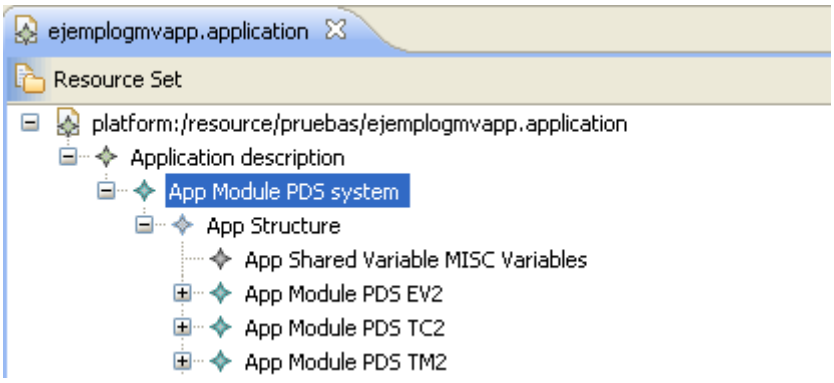


Figure 4: MARTES e-core of a GMV model

11.4 Code Generator

The UC toolset includes a SystemC generator that allows simulating the GMV models. This generator transforms XML files into a set of SystemC files. In order to obtain an executable model, these files have to be compiled with a specific library.

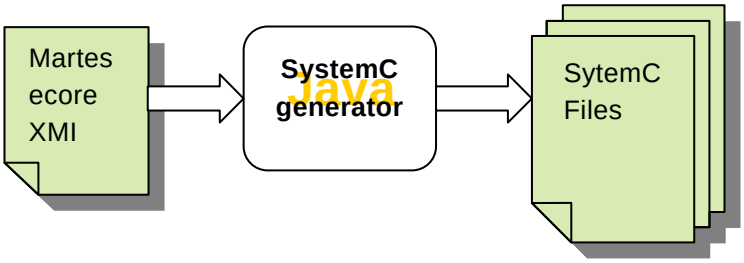


Figure 4: SystemC code generation

The UC toolset includes two libraries. The first one allows simulating the GMV application model. The second one allows evaluating the system performance (e.g. time execution, power consumption…). This library depend of the execution platform thus there is a close relation between this library and the GMV execution platform model.

12 THALES Research & Technology

Tool enhancements (performance simulation) and (EMF-based) gateways are related to the TRT's design tool called SPEAR.

12.1 Tool enhancements

First TRT enhanced its SPEAR Design Environment; its previous performance simulator written in Java upon Ptolemy II (Discrete Event simulation domain) has been replaced by a SystemC version. No change for the user in terms of GUI but it is obviously faster than before.

Performance simulation relies on three main submodels : Execution platform, Application and mechanisms description (see figure below).

The SPEAR Execution platform submodel details how the (quasi-physical) hardware is built (with which processors, memories and busses) and how HW components are connected together. Each element (processor, memory and bus) is tagged by a value that is associated with a "standardised" behavioural class. By standardised, we mean that only few methods have to be written for each kind of element, the remaining behaviour is inherited from the simulator core.

The SPEAR Application submodel describes the time and space parallelisation: the task scheduling, the amount of data that has to be computed and/or transferred. Data are allocated to one or several memories, computations are allocated to one or several processors and communications are allocated to busses and memories. Hence the SPEAR application (at the allocated model level) is strongly linked to the SPEAR execution platform.

Finally, each task is associated to a mechanism. There are two kinds of mechanisms:

- Elementary Transforms (ET), which are related to computations. They are characterised by an execution time per iteration (each ET is repetitive within the SPEAR context);
- Elementary Moves (EM), which are related to communications. They are characterised by a data path. A data path points out which memory is managed by which processor or communication controller, and which bus will be crossed to reach other memories.

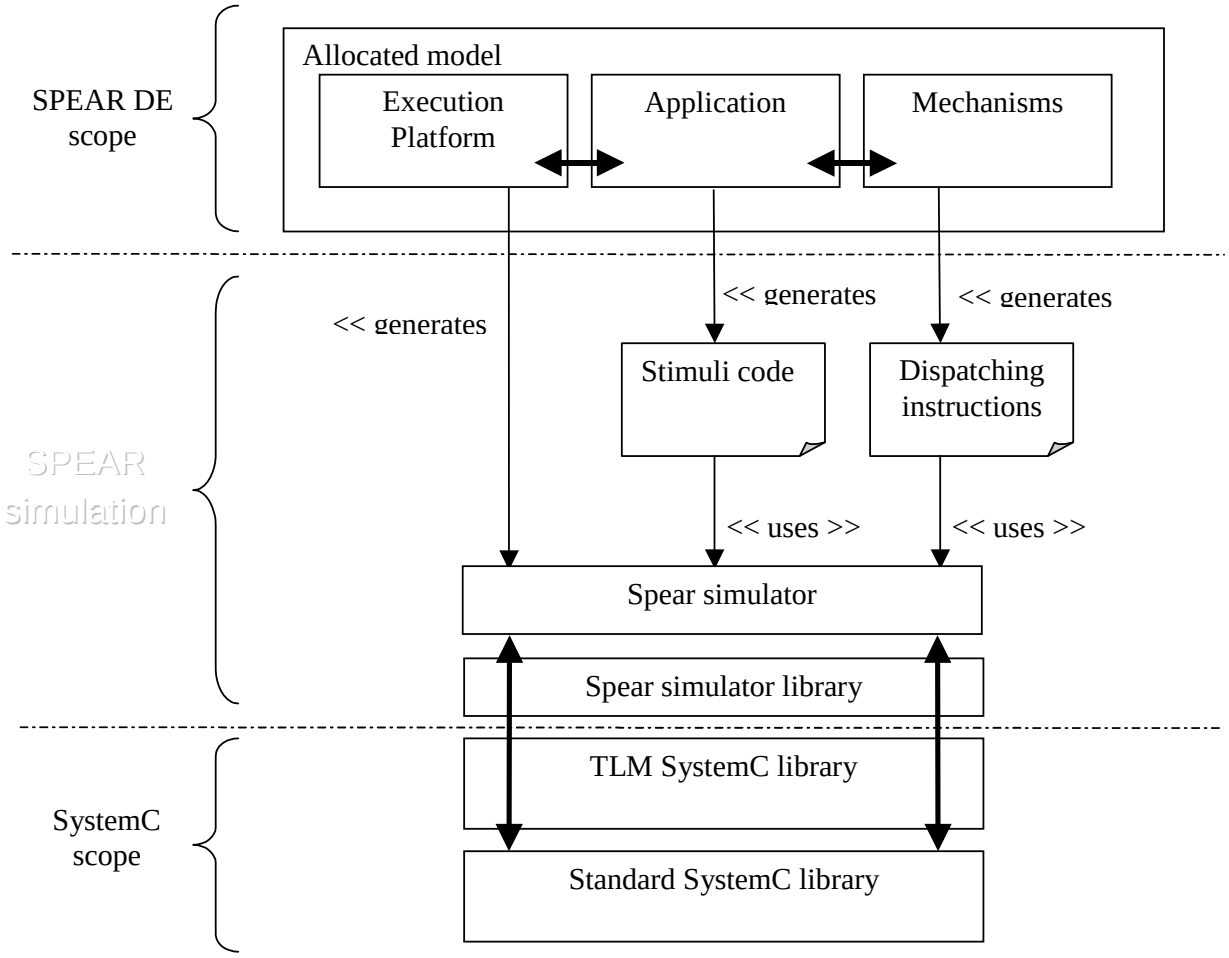
These mechanisms enable to generate the quasi-physical platform description from the execution platform submodel (still at the allocated model level). The underlying generated SystemC code is based on the SPEAR simulator library and can be compiled and linked to obtain an executable code. The application scheduling and the number of task iterations are generated from the application submodel into stimuli files. Finally, the execution time of each ET and the way whose communication instructions will be dispatched within this execution platform (EM) are generated from the mechanism submodel into dispatching files. There are a

stimuli and a dispatching file per initiator, where initiators are simulation elements that are in charge of analysing stimuli code and dispatching it to active elements of the executing platform (processors and/or communication controllers).

All elements in the simulator are based on standard SystemC and TLM libraries. Elements are “sc_module” and they are linked (through “sc_port”) with TLM channels. Instructions that crosses the whole execution platform are based on standard SystemC types (“sc_time”, “sc_signal”, …).

Regarding implementation, there are three layers:

1. the first one considers SPEAR application and execution platform models but also “mechanisms” that express data production and consumption (and at a coarse grain level the model of computation);
2. in comparison with the last layer, the second one is needed for our performance simulator because it enables to take into account memory contention; the execution platform model is translated into SystemC code, the parallelised application is converted into stimuli code that feeds the execution platform one and mechanisms consist in (sub)services that are dispatched upon the execution platform elements;
3. the last layer is similar to the usual SystemC distribution.



This package is fully integrated into SPEAR DE and hence is not available without.

12.2 Gateways

Second TRT implemented the SPEAR DE internal structure using Ecore. It also developed gateways that enable to import / export models with the common MARTES models.

12.2.1 Matching between SPEAR / MARTES

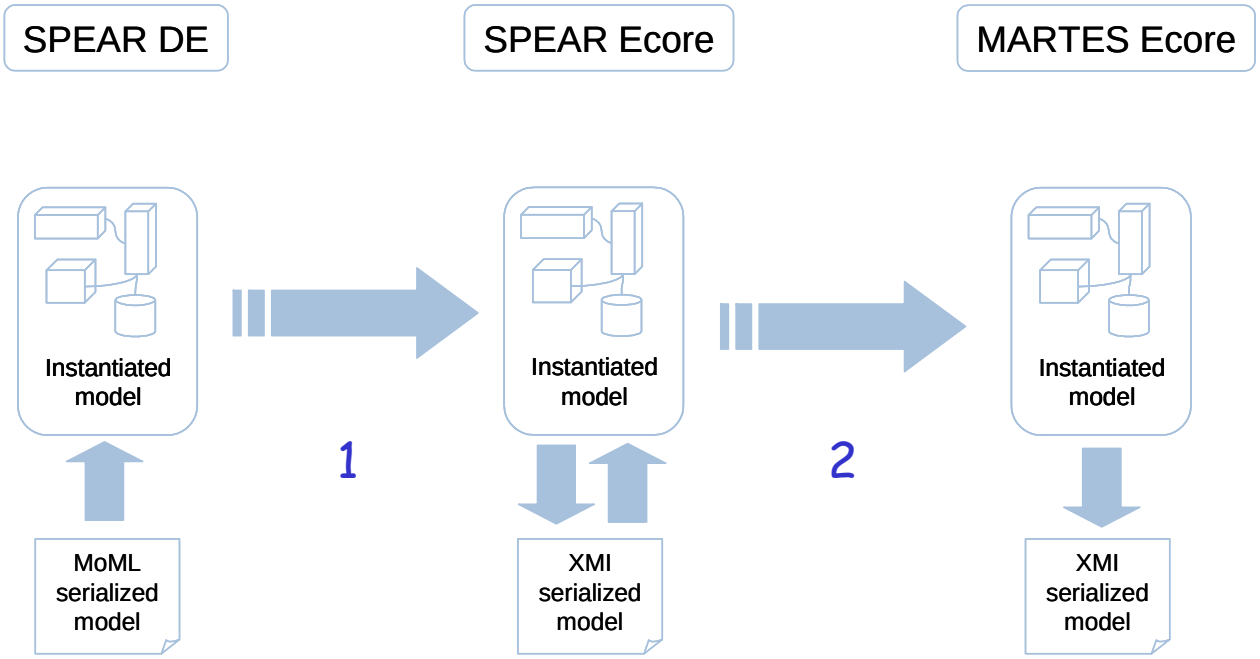
The following table shows the matching between the SPEAR gateway meta-model objects and the common MARTES ones:

SPEAR Meta Model	MARTES Meta Model
Application	
Root task	Application
Computation	AppModule + AppCompInterface + AppBehavior + AppAttributes + AppStructure
ArrayPort	AppPort
Variable	AppTemplateParameter
Array	AppSyncArray
AppRelation	AppSyncConnector
DataType	Datatype
ArrayView	AppAttribute
InOutAttribute	AppAttribute
View	AppAttribute
Motif_XD	StringExpresson
Elementary Transforms	AppAttribute
Paramter	AppAttribute
IO	AppAttribute
Timing	AppAttribute + StringExpression
Origin	AppAttribute + StringExpression
Fitting	AppAttribute + StringExpression
Paving	AppAttribute + StringExpression
Architecture	
Root ArchComposite	ExecutionPlatform
Processor	XpProgrammableResource
CommResource	XpCommManagerResource
Bus	XpBus
Link	XpPoint2Point
Memory	XpStrorageResource
ArchComposite	XpResource + XpStructure
Initiator	XpSpecializedResource
ArchPort	XpPort
ArchRelation	XpConnector
Profile	XpAttribute + Stringexpression

12.2.2 Export

The export phase (see figure below) consists in 2 steps:

1. the first one loads the (SPEAR DE) models in memory, converts instantiated SPEAR objects into SPEAR Ecore ones and serializes the (SPEAR Ecore) model into XMI
2. the second one loads the (SPEAR Ecore) models in memory, converts instantiated SPEAR Ecore objects into MARTES Ecore ones and serializes the (MARTES Ecore) model into XMI

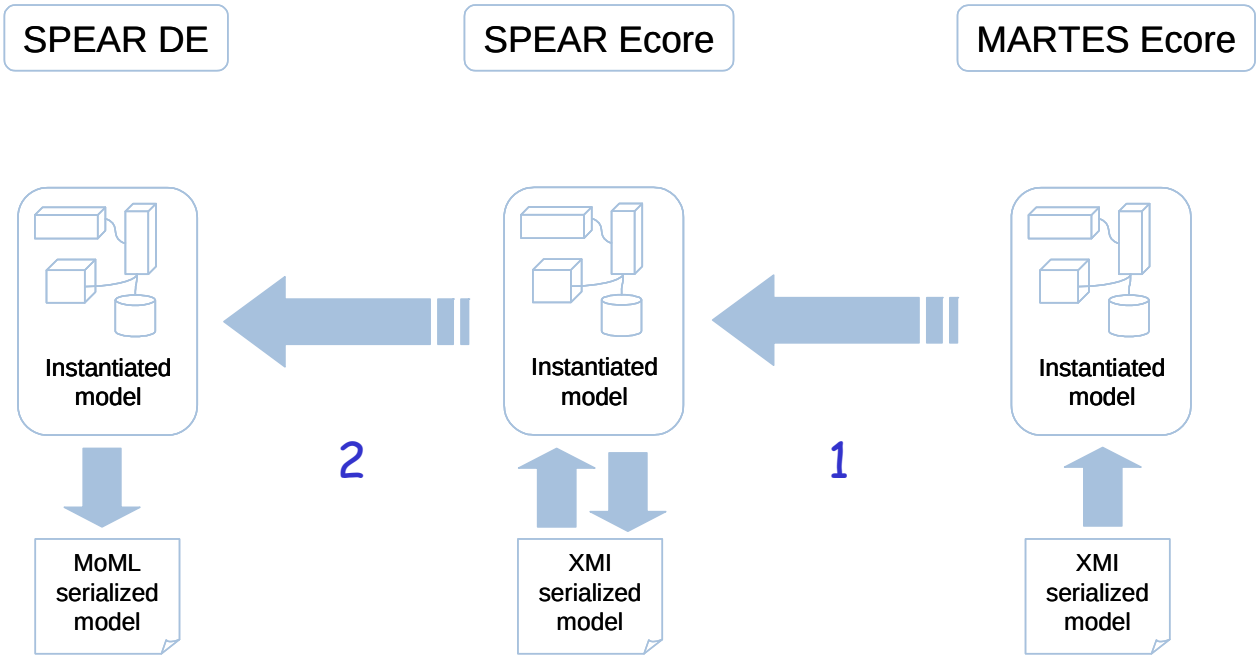


12.2.3 Import

The import phase (see figure below) also consists in 2 steps:

1. the first one loads the (MARTES Ecore) models in memory, converts instantiated MARTES Ecore objects into SPEAR Ecore ones and serializes the (SPEAR Ecore) model into XMI
2. the second one loads the (SPEAR Ecore) models in memory, converts instantiated SPEAR Ecore objects into SPEAR DE ones and serializes the (SPEAR DE) model into XMI

There is no need to make available these EMF implementations since close to the SPEAR DE internal structure and so useless for other developments.



13 Conclusion

This document described the different tool enhancements and gateways of each partner funded by the MARTES project.