Model-based development of embedded systems - the MADES approach

Neil C. Audsley, Ian Gray*, Leandro Soares Indrusiak,
Dimitris Kolovos, Nikos Matragkas*, Richard Paige
University of York, York, U.K.

***Abstract*—This paper discusses the goals of the MADES project, which aims to use techniques of model-driven development to assist the development of complex embedded systems. Three main areas are covered by MADES - the development of embedded software; the development of embedded hardware; and the verification and validation of the system. This paper presents an early example of the MADES model transformations being used to generate synthesisable VHDL descriptions from high-level UML MARTE models.**

***Index Terms*—Model Driven Engineering, Epsilon, MADES, Embedded Systems, Real-Time Systems.**

I. Introduction

The architectures of embedded systems are becoming increasingly non-standard and application-specific. They frequently contain multiple heterogenous processing cores, non-uniform memory, complex interconnect or custom hardware elements such as DSP and SIMD cores. However, programming languages have traditionally assumed a single processor architecture with a uniform logical address space and have abstracted away from hardware implementation details. As a result, developing software for these architectures can be challenging. Equally, such systems are frequently deployed in high-integrity or safety-critical systems which require the highest levels of predictability and reliability.

The MADES Project is an EU-funded project that aims to use model-driven techniques to enable the development of the next generation of highly complex embedded systems, whilst reducing development costs and increasing reliability and predictability. In this paper we provide an overview of the MADES approach to model-driven development of embedded systems.

II. MADES Project Goals

The MADES project aims to develop the elements of a fully model-driven approach for the design, validation, simulation, and code generation of complex embedded systems to improve the current practice in the field. MADES differentiates itself from similar projects in that way that it covers all the phases of the development process: from system specification and design down to code generation, validation and deployment. Design activities exploit a dedicated language developed on top of the OMG standard MARTE (Modeling and Analysis of Real-time and Embedded systems) [4], and foster the reuse of components by annotating them with properties and constraints to aid selection and enforce overall consistency.

Validation activities comprise the verification of key properties of designed artifacts and of the transformations used throughout the development process, and also the closed-loop simulation of the entire system. Code generation addresses both conventional programming languages (e.g., C) and hardware description languages (e.g., VHDL), and uses the novel technique of Compile-Time Virtualisation to smooth the impact of the diverse elements of modern hardware architectures and cope with their increasing complexity.

All these aspects will be fully supported by prototype tools integrated in a single framework, and will be thoroughly validated on real-life case studies in the surveillance and avionic domains. The project also aims to develop a handbook to provide detailed guidelines on how to use MADES tools in the development of embedded systems and promote their adoption.

III. The MADES Approach

A conceptual model of the inter-relationships between the various artifacts of the MADES approach is illustrated in figure 1.

One of the main characteristics of the MADES approach is the use of model-driven transformations throughout the entire design process. These transformations are used to convert one or more input specifications into one or more output specifications. Model transformation languages are defined in a metamodel level and establish the relationship between source metamodel elements and target metamodel elements (see figure 2).

The MADES approach focusses on three areas:

- Generation of platform-specific embedded software from architecturally-neutral software specifications.
- Generation of hardware descriptions of the modelled target architecture.
- Verification of functional and non-functional properties.

Development effort starts with building design and analysis models using the MADES modelling language, which is based on a combined subset of OMG MARTE [4] (a UML profile for modelling real-time embedded systems)

*Supported by EU Framework 7 research contract FP7-248864

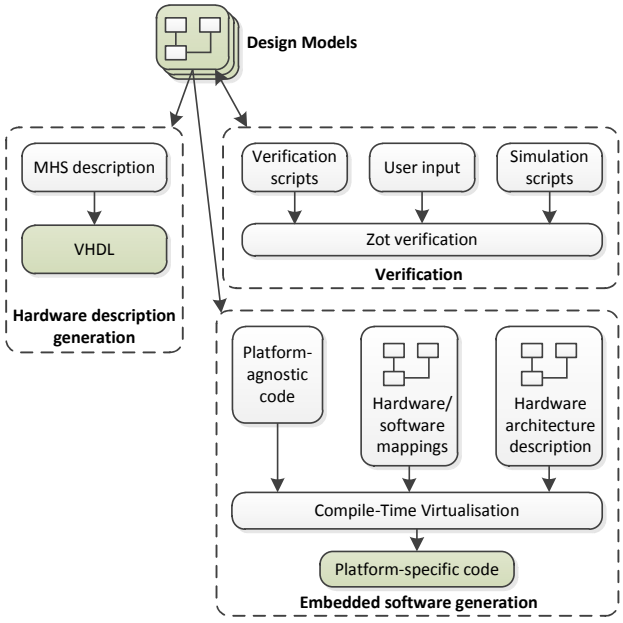

Fig. 1. Overview of the artifacts in the MADES approach

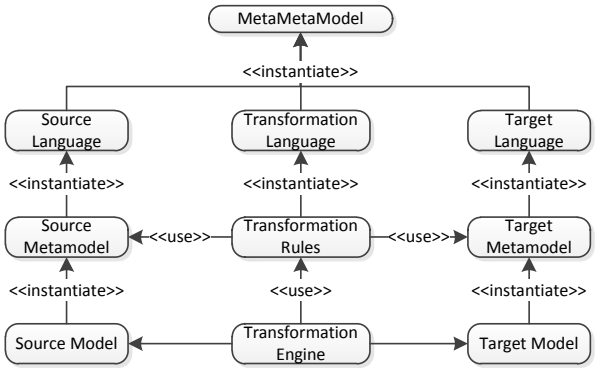

Fig. 2. MADES Mapping Scheme

and SysML [6] (a general-purpose modelling language for systems engineering applications). The language aims to overcome shortcomings of these existing languages and to provide:

- Specifications of functionality in an architecturally-neutral way.
- Descriptions of target hardware.
- Deployment diagrams that map functionality to hardware/software.
- Timing and non-functional properties for early and frequent verification.
- Code-reuse, component-based design, and maintainability.

Verification and simulation play a key role in the MADES approach. Such activities are present during:

- Verification of key properties on designed artifacts (for example, whether a system will meet a specified deadline, or be able to support a specified volume of data).
- Closed-loop simulation based on detailed models of the environment (for functional testing and early validation).
- Verification of designed transformations (from high-level system models down to low-level hardware/software implementations).

In order to provide verification and simulation in the MADES toolset, the Zot tool [5] is used. The verification phase aims to provide rapid and early verification of the system to reduce design time and guarantee correctness of the final system.

The MADES code generation phase allows the designer to model the target hardware at a high-level of abstraction and use deployment diagrams to map the input code (which is provided in an architecturally-neutral form) to elements of the target hardware. A technique called Compile-Time Virtualisation (CTV) [1], [2] is used to hide the complexity of embedded software development through the provision of a Virtual Platform (VP). The VP is an idealised view of the underlying hardware which provides a simple programming model that is compatible with the chosen source language. The result of this is that the programmer can write code as if it is to be executed on the VP, and CTV will then automatically retarget this code for execution on the actual platform. If the actual platform changes during development (for example, due to hardware redesign or changing requirements) the same input code can be automatically retargeted to the new hardware and does not manually ported.

Finally, the MADES approach also considers generation of synthesisable hardware descriptions from the hardware model. The deployment mappings of the code generation phase use a high-level description of the capabilities of the desired target architecture (for example, “three processors connected with a common bus, two banks of shared memory”). This can be reified into an unambiguous hardware description for implementation using the MADES approach.

The Epsilon platform [3] is used to implement both the model-to-model and model-to-text transformations used in the MADES approach. Epsilon (Extensible Platform of Integrated Languages for mOdel maNagement) is a platform for building consistent and inter-operable task-specific languages for model management tasks such as model transformation, code generation, model comparison, merging, refactoring and validation. Epsilon can provide traceability information produced by the various transformations, which is of paramount importance for embedded systems design due to the need to comply to particular standards such as the DO-178B Standard.

IV. Hardware generation example

This section presents the preliminary implementation of one of the three main areas of the MADES project, that of model-directed hardware generation. Following papers will discuss the areas of software development and verification.

The hardware generation approach is shown in figure 3. In this implementation, FPGAs are used to implement

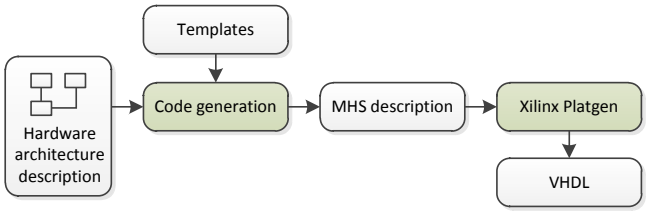

Fig. 3. MADES hardware generation

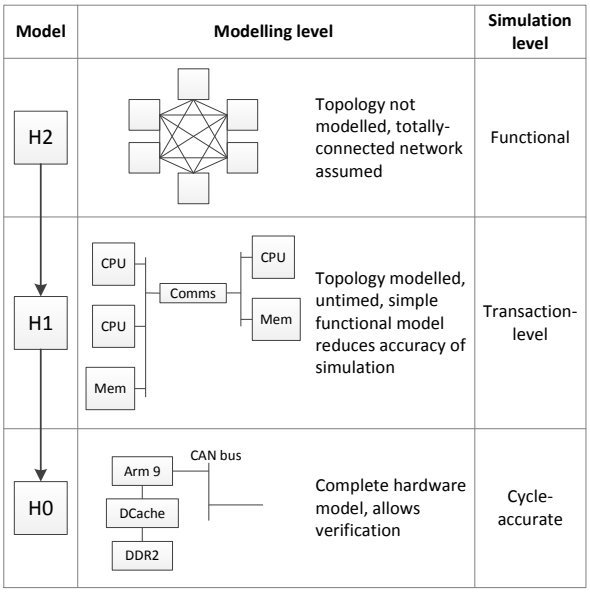

Fig. 4. MADES hardware models

and test the generated architectures. The hardware architecture description is provided using a set of MADES hardware modelling languages. These languages are briefly described below and detailed in figure 4, later papers will formally define these languages:

- *H2:* A high-level model of the target architecture. Its constituent elements are processors, memory spaces, communication channels, clocks, and custom hardware elements. Architectural links (for example CPU → Memory) are used to connect elements. This model does not describe any lower-level details (such as bus topologies).
- *H1:* A refinement of H2, H1 is a lower-level model which codifies the bus topology of the system and specifies I/O. It still contains all information from H2, it is a true refinement. H1 does not define the specific types of each hardware instance. For example, at this level the model will still denote 'processor' rather than 'Arm9'.
- *H0:* H0 instantiates the H1 model in terms of the MHS language (see below). Generated by model-to-model transformation from H1 using transformation rules and in-place refinement, known as 'polishing rules'.

H0 demonstrates an equivalent level of abstraction to that of the Microprocessor Hardware Specification (MHS) language [9] used by Xilinx Corporation's FPGA Development Tools. MHS files are generated by the MADES tools from an H0 model using model-to-text transformation. The Xilinx tool 'platgen' [8] is then used as an underlying HDL generator, as it can take an MHS description and generate a set of synthesisable VHDL files for implementation on an FPGA. This allows the generated architectures to be realised and tested. The development process is therefore:

MARTE stereotype	H2 class
«hwProcessor»	Processor
«hwMemory»	Memory
«hwClock»	Clock
«hwResource»	Channel
«hwComponent»	OtherHardware
«hwProcessor» → «hwMemory»	CPUMemoryLink
«hwProcessor» → «hwResource»	CPULink
«hwProcessor» → «hwComponent»	CPULink

TABLE I
MARTE TO H2 MAPPINGS. A RIGHT ARROW (→) DENOTES AN ASSOCIATION BETWEEN TWO CLASSES.

1) Initial hardware model is specified in a high-level UML MARTE model.
2) This model is translated using the Epsilon toolchain with a model-to-model transformation into H2. The mappings for this are shown in table I.
3) H2 model is refined through H1 to an H0 model using the Epsilon framework.
4) The H0 model is converted to an MHS file using a model-to-text transformation.
5) The MHS file can then be passed to the Xilinx tools to generate VHDL.
6) The generated VHDL is then synthesised and implemented using the tools of the FPGA vendor.

The use of MHS files and Xilinx FPGAs is not required by the MADES toolchain, but it is a useful language to work with as it already has robust industrial-quality tool support available. If a different language or implementation fabric is required, another model-to-text transformation can be used. The modularity of the Epsilon framework assists the development of multiple output translations without affecting the high-level models.

Figure 5 shows an example result of the MARTE to H2 transformation. As can be seen, this translation maintains the same level of information as its classes correspond to MARTE's hardware stereotypes (annotations and properties can also be carried over) and is used to convert the model into a form that can be used in the rest of the Epsilon-based MADES toolchain.

More translation work is required in the H2 to H0 translation as H0 contains a greater amount of information than H2. This information is filled in using a template-based solution. For example, instances of H2's processor class can be assigned a 'type' property that informs the translation process which specific processor type should be used to implement it. If this is set to a Microblaze soft processor [7] then the translation process will instantiate the Microblaze template. Templates are composed of MHS

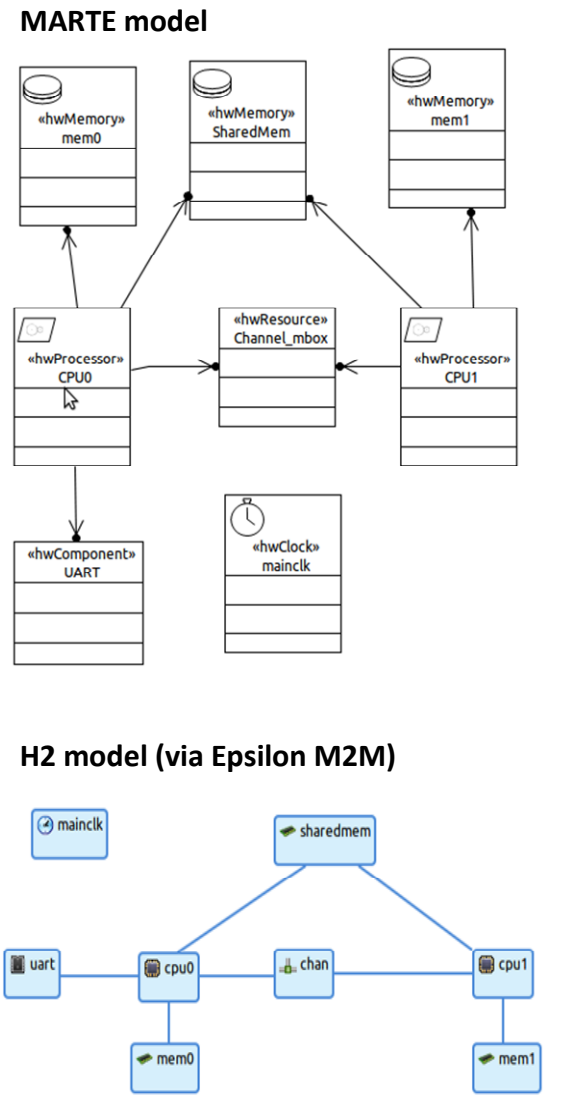

Fig. 5. Transformation example between MARTE and H2

segments that describe a the hardware element in question and any additional support peripherals that it requires. (The Microblaze example also instantiates an interrupt controller and clock generation logic.)

Templates are also used to specify bus topologies. The default template instantiates a single peripheral bus for each processor of the target architecture. If a peripheral is connected to multiple processor buses but only supports a single bus interface than a new shared peripheral bus is created and bus bridges used. The list of supported templates is currently small, but is being constantly expanded.

Processor address maps can be automatically determined (by successively assigning addresses from 0x00000000 upwards) or specified by the designer using properties in the MARTE model which are then carried into the M2 model by the transformation.

This preliminary implementation of the MADES hardware generation system is currently being used to assist development of the MADES modelling languages. Through experimentation it can be determined which hardware modelling features are important and if there are any features that are not already covered by MARTE.

V. Conclusion

The MADES project aims to use model-driven engineering techniques to aid the development of embedded systems. It uses a systems modelling language based on MARTE and SysML that allows the developer to express their system at a high-level of abstraction, and then to iteratively refine their design to reach the final implementation. MADES differentiates itself from similar work through three unique features. First, extensive use of model transformations is used to facilitate development and provide traceability. Second, verification and validation are key parts of the MADES design flow, allowing early and frequent verification of the system being developed. Third, Compile-Time Virtualisation (CTV) is used to assist the development of embedded software.

This paper has shown how the MADES model transformation framework is being used to generate synthesisable hardware descriptions of non-standard embedded systems. Whilst only a preliminary implementation, the translations can already produce complex hardware descriptions quickly from high-level system models with only minimal designer input. Model transformation techniques are also being applied in the MADES project to assist the mapping of software to the generated architecture.

References

[1] I. Gray and N. Audsley. Exposing non-standard architectures to embedded software using Compile-Time Virtualisation. *International Conference on Compilers, Architecture, and Synthesis for Embedded Systems (CASES '09)*, 2009.

[2] I. Gray and N. Audsley. Supporting islands of coherency for highly-parallel embedded architectures using Compile-Time Virtualisation. In *13th International Workshop on Software and Compilers for Embedded Systems (SCOPES)*, 2010.

[3] D. S. Kolovos, R. F. Paige, and F. A. C. Polack. Eclipse development tools for epsilon. In *In Eclipse Summit Europe, Eclipse Modeling Symposium*, 2006.

[4] Object Management Group. UML profile for MARTE: Modeling and Analysis of Real-Time Embedded Systems. http://www.omgmarte.org/, November 2009.

[5] M. Pradella, A. Morzenti, and P. San Pietro. The symmetry of the past and of the future: bi-infinite time in the verification of temporal properties. In *ESEC-FSE '07*, pages 312–320, New York, NY, USA, 2007. ACM.

[6] T. Weilkiens. *Systems Engineering with SysML/UML: Modeling, Analysis, Design.* Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2008.

[7] Xilinx Corporation. Microblaze processor reference guide. UG081 v9.0, 2008.

[8] Xilinx Corporation. Embedded system tools reference guide - EDK 11.3.1. *Xilinx Application Notes*, UG111, 2009.

[9] Xilinx Corporation. UG642: Platform specification format reference manual. http://www.xilinx.com/support/documentation/sw_manuals/xilinx11/psf_rm.pdf, September 2009.