

# Key Research Challenges for Successfully Applying MDD within Real-Time Embedded Software Development\*

Aram Hovsepyan, Stefan Van Baelen, Bert Vanhooff, Wouter Joosen and Yolande Berbers

Katholieke Universiteit Leuven, Celestijnenlaan 200A, B-3001 Leuven, Belgium  
{Aram.Hovsepyan, Stefan.VanBaelen, Bert.Vanhooff, Wouter.Joosen, Yolande.Berbers}@cs.kuleuven.be

**Abstract.** Model-Driven Development (MDD) is a software development paradigm that promotes the use of models at different levels of abstraction and perform transformations between them to derive one or more concrete application implementations. In this paper we analyze the current status of MDD regarding its applicability for the development of Real-Time Embedded Software. We discuss different modeling framework approaches used to specify the various models, and compare OMG/MDA-based approaches (MOF, UML Profiles and executable UML) with a generic MDD-based approach (GME). Finally, we identify the key challenges for future MDD research in order to successfully apply MDD within RTES Development. These challenges are mainly situated in the field of modeling and standardization of abstraction levels, model transformations and code generation, traceability, and integration of existing software within the MDD development process

## 1 Introduction

Model-Driven Development (MDD) is a software development paradigm that promotes the use of models at different levels of abstraction and performs transformations between them in order to derive a concrete application implementation. MDD promotes the construction of high-level models which can be (semi-) automatically transformed to lower-level models and ultimately into optimal code for the selected target implementation platform. A model is a coherent set of formal elements built for some purpose that is amenable to a particular form of analysis. A model is expressed in a modeling language at some abstraction level which in itself can be defined by metamodels. MDD captures expert knowledge as mapping functions that transform between one model and another.

In the same manner as compilers raised the programming abstraction level from assembler code towards higher-level programming languages, thereby automatically transforming the language constructs into machine-level instructions,

---

\* The described work is part of the EUREKA-ITEA MARTES project, and is partly funded by the Flemish government institution IWT (Institute for the Promotion of Innovation by Science and Technology in Flanders).

MDD tries to upgrade the software development process artifacts from code towards models. Models will as such become the key development assets within software development. Such approach will allow to design an application once and target it towards distinct software and/or hardware platforms, even towards future platforms that are still unknown during the initial development. MDD will enable better integration and interoperability on top of the target platform and supports system evolution as platform technologies evolve.

Even though research on several aspects of MDD (modeling abstractions, transformations, processes, ...) has been going on for several years, there are still a number of issues to be solved in order to successfully apply MDD for software development in general and for Real-Time Embedded Software in particular.

In section 2, we compare the distinct approaches towards an MDD modeling framework and application model specifications at different levels of abstraction. On the one hand, a meta-metamodel approach could be used in order to create a DSML, using OMG's MOF or another meta-metamodel. The newly specified DSML can then be used in turn to specify dedicated application models. On the other hand a general purpose modeling language such as UML could be used, profiling it to simulate the required DSML. As a third approach, executable UML could be used as a high-level platform abstraction on top of which applications could be simulated or executed.

In section 3, we identify the key challenges for future MDD research in order to successfully apply MDD within Real-Time Embedded Software Development. These challenges are mainly situated in the field of modeling and standardization of abstraction levels, model transformations and code generation, traceability, and integration of existing software within an MDD development process.

## 2 MDD approaches

This section compares the different approaches towards an MDD modeling framework and application model specifications at different levels of abstraction.

### 2.1 MDD versus MDA

MDD is a generic software development paradigm that can be applied in different manners. The Object Management Group (OMG) has defined a variant of MDD called Model-Driven Architecture (MDA). MDA aims to represent systems using OMG's general-purpose Unified Modeling Language (UML) along with specific profiles, or using a Domain-Specific Modeling Language (DSML) expressed in OMG's MOF (Meta-Object Facility). The key idea behind MDA is to start with the specification of an Platform-Independent Model (PIM), extracting the common concepts of an application targeted towards a number of platforms and as such allowing a higher level of application specification. Starting from this PIM, a PSM is generated which defines an application model targeted towards a specific platform. A PSM is an elaboration of a PIM that includes platform

specific details. For many parts of the application model, the process of obtaining a PSM from a PIM can be automated and the transformation knowledge captured into platform-specific transformation rules. From this PSM, code can be (semi-)automatically generated for the target platform to a certain extent (skeleton, partial or full code generation).

Model-Integrated Computing (MIC) [1] is another variant of MDD, introduced by the Institute for Software Integrated Systems (ISIS) at Vanderbilt University. MIC uses DSMLs to represent system elements and their relationships as well as their transformations to platform-specific artifacts. We discuss in the next section how the meta-metamodel approach can support MIC.

## 2.2 Modeling Frameworks

Models and transformations are key concepts in MDD. Currently several different approaches exist for specifying application models at different abstraction levels within MDD software development. Although we do not aim to be exhaustive, we will discuss the main distinctive approaches.

**2.2.1 Meta-metamodel approach** A rather formal approach consists of firstly creating a generic modeling infrastructure in order to describe different kinds of metamodels. This meta-metamodel can then be used to create DSMLs which in turn are used to specify dedicated application models. The meta-metamodel should be able to describe what exactly should exist in the DSML in terms of concepts, how they relate to one another and which rules govern their existence and behavior.

**MDA-based MOF** Probably the most known instance of a meta-modeling framework is the MOF. The MOF architecture conceives of four “meta-levels”. The highest level (M3) is the MOF meta-metamodel which is an instance of itself. The MOF can be used to create a DSML (M2). This newly created DSML can be used to model an application (M1), which in turn contains a run-time instantiation model (M0).

Currently there are very limited number of tools that support a DSML definition based on the complete MOF 2.0 metamodel. Several shortcomings of MOF and its implementation are outlined by [8] and [9]:

- As any standard, the MOF standard prescribes no choice for implementation.
- The standard API for browsing through models in a MOF repository is too low-level to be really efficient.
- MOF lacks any standard mechanisms for specifying DSML concrete syntax. There is no standard way to declare a particular graphical notation.
- MOF’s lack of support for associations incorporating state makes the definition of some DSMLs awkward.
- Interoperability between “MOF-compliant” tools is a huge issue (see the discussion on XMI in section 3).

**Generic Modeling Environment** Even though MDD is mostly associated with MDA and OMG standards there are modeling frameworks other than MOF. The Generic Modeling Environment (GME) [7] is an alternative implementation of the meta-metamodel approach, and supporting the MIC approach.

GME is a configurable toolkit for creating DSML and program synthesis environments. The configuration is accomplished through metamodels specifying the modeling language of the application domain. The metamodel contains all the syntactic, semantic, and presentation information regarding the domain and the concepts to be used for constructing models. The modeling framework also specifies what relationships may exist among those concepts, how the concepts may be organized and viewed by the modeler, and rules governing the construction of models. The metamodels specifying the DSML are used to automatically generate a target domain-specific modeling environment. This environment can then support the specification of dedicated domain models that are stored in a model database. From these models, applications can automatically be generated. GME has full-featured universal predicate expression language (based on OCL), which can represent very complex relational constraints.

**2.2.2 “Lightweight” approach using UML Profiles** OMG advocates the more pragmatic approach for a modeling framework based on the idea of developing a DSML using UML Profiles. UML has been improved with modeling language extension features which can raise UML’s abstraction level. Stereotypes and tagged values are the extension mechanisms that can be grouped in a profile. The profiling information is used by modeling tools, model transformers and code generators in order to perform domain-specific actions. An example of such an extension is the UML profile for Modeling and Analysis of Real-Time and Embedded systems (MARTE) [11], which is discussed in section 3.

Even though extending UML is considered to be easy, UML as well as its extension mechanisms are quite complex. It is also not clear how well tools will be able to manipulate and exchange these UML Profile extensions.

**2.2.3 xUML approach** Executable UML (xUML) [12] is a third approach to MDD, whereby compilers for the UML modeling language (or a specific subset) are built, treating UML as a programming language on its own. Developing applications using xUML offers several advantages, such as precise and complete semantics, and model visualization and simulation at early stages. However an executable UML model does not have the expressive power of programming languages at the current moment. For example, an executable UML model does not specify issues regarding distribution, the number and allocation of separate threads, or the organization of data. In addition, xUML does not have the power of domain specific modeling languages and supports only a small subset of UML.

**2.2.4 Discussion** There are two OMG visions on MDA, supporting respectively MOF and UML extensions (UML Profiles and xUML). Non-OMG MDD

approaches mostly rely on meta-metamodel approaches in order to define dedicated DSMLs. GME is an example of such generic MDD modeling framework. GME as most of other non-OMG approaches introduces proprietary standards which inhibit the interchange capabilities. This is precisely one of the strongest points of MOF, which introduces unique means of model interchange and storage. However from a modeling framework perspective, MOF still misses generic editor and generator definitions for DSML creation, and proper tool support. On the other hand, GME and other similar non-OMG modeling frameworks do offer tools to support their modeling frameworks, although they are rather specific for the underlying approach.

There is no reason why UML cannot be used as a base for the development of a domain specific modeling framework, although it is questionable whether using such approach is a good way to build modeling frameworks. Since UML Profiles are defined on top of the whole UML standard, any DSML that is defined as a UML Profile carries the whole UML metamodel within (if not specifically excluded from the profile). The profile approach also restricts the DSML from using the full semantic power of object-oriented class modeling that a true meta-metamodel approach could offer. In addition, within a UML Profile one cannot declare new associations among UML metamodel elements or among stereotypes.

### 2.3 MDD Promises

The MDD and MDA approach promise a number of important benefits to significantly improve the software development process once a full MDD software development process is in operation.

***Time savings*** A full MDD development process supported by adequate tool support can provide significant time savings by generating dedicated code for a specific execution platform from the high-level models. Advanced tools will generate code from dynamic models and even provide suitable code for the realization of constraints expressed in e.g. OCL (Object Constraint Language). In addition, reuse of architectures and designs will be actively supported.

***Quality improvement*** A well-defined architecture incorporates adequate solutions for the realization of the system quality attributes, such as performance, availability, security, modifiability, scalability, reliability etc. In traditional software approaches the system architecture is well-designed during the first iteration, but tends to get diluted by subsequent iterations and new upcoming requirements. Because models and automatic transformations are central in an MDD approach, the system architecture can always be enforced and updated as new requirements arise. The transformation and code generation mechanisms will be created and extensively tested by experts. This raises the quality of every step in an MDD-based development process.

***Cross-platform development and enhanced platform migration*** As platforms change over time, software applications must continuously be re-implemented. Typically software developers either start everything from scratch or try to port the application to the new platform. In the first case the previous solution gets (partially) lost, while in the second case developers often invent low-level hacks

in order to get the application running on the new platform. MDA offers support for cross-platform development and enhanced platform migration by introducing a PIM representation of a software system. When the platform changes, the application can be preserved by reusing the PIM in order to generate a PSM and code of an application targeted towards this new platform.

### 3 Key Research Challenges

This section presents a number of key issues that currently obstruct the application of MDD for the development of Real-Time Embedded Software.

#### 3.1 Modeling Levels

Even though models are a central concept in MDD, it is not yet obvious which abstraction levels and notations are the most suitable.

**3.1.1 Models for Embedded Platforms** Although MDD has already been successfully applied in a number of embedded pilot projects, it is not yet very clear how to integrate the variations of the software platforms, hardware platforms, and available services and devices of a system. There is no simple notion of a “platform” as in the case of more general software development (e.g. J2EE, .NET). Moreover, the gap between an embedded platform and an application model abstraction is usually larger. A number of UML profiles have been designed to assist modeling for embedded systems.

SysML [10] is a domain-specific visual modeling language for System Engineering. SysML supports the specification, analysis, design, verification and validation of a broad range of systems and systems-of-systems. These systems may include hardware, software, information, processes, personnel and facilities.

UML profiles for System on Chip (SoC) and SystemC are designed in order to assist integrating UML modeling into the current SoC design process. The UML 2.0 profile for SystemC captures both the structural and the behavioral features of SystemC language. It makes translation from a high-level platform independent to a lower-level platform dependent SystemC model straightforward.

The MARTE profile intends to provide a common way of modeling both hardware and software aspects of Real-Time and Embedded Systems. As a result, interoperability between development tools used for specification, design, code generation etc. will be possible. Quantitative and partitioning predictions regarding hardware and software characteristics will be fostered. The profile is intended to provide a foundation for applying transformations from UML models into a wide variety of analysis models. The MARTE profile defines precise semantics for time and resource modeling. These precise semantics allows to automatically transform models to lower abstraction level models such as UML for SoC for hardware/software simulation or into C++ for implementation purposes.

However all the profiles are still under development and not yet officially standardized. Moreover they tend to overlap and their interrelationships are still unclear.

**3.1.2 Concepts to Model** While static class and component diagrams describe the software structure, dynamic models describe its behavior. Clearly we cannot expect a generator to produce more than just skeleton code if we do not provide information about the behavior.

The MDD community has progressed very little concerning the code generation from dynamic models. Even though most of the modeling frameworks allow users to create dynamic models, the lack of uniform methodology to generate code from models decreases the added value of creating and maintaining dynamic models. It is also not clear how far we should go using dynamic models. Different mathematical algorithms (e.g. Fast Fourier Transforms) could be modeled using UML collaboration diagrams. But it is unclear whether the benefits still outweigh the complexity of creating complete models. Very often, such algorithms are easier and shorter to write directly in code.

## 3.2 Model Transformations

Even though model transformations and code generation are central concepts in MDD, it is not yet obvious how to define and apply the model transformations in order to establish an adequate MDD transformation chain.

**3.2.1 Transformation Implementation** One of the biggest limitations of the MDA approach is the lack of a unique standard for specifying transformations between models, as well as between models and code. Many custom solutions have been introduced such as Atlas Transformation Language (ATL)[4], usage of OCL to generate code, and Velocity Templates[6]. However these solutions are not standardized and work only with specific tools.

OMG has issued two Requests for Proposals (RFP) for MOF 2.0 QVT[5] and for MOF Model to Text Transformation (MOF2T) [3]. MOF2T is still in its early development stage. QVT describes the needs for a new standard that should be able to manipulate any model based on the Meta-Object Facility (MOF) meta-model. Since the RFP issue, there have been several submissions which were ultimately merged into one [5].

**3.2.2 Transformation Composition** We expect that transformations should be able to incorporate functional, non-functional (e.g. memory management) and technical (e.g. J2ME specifics) concerns. Obviously, such transformations could become very cumbersome and complex. Ideally, there should be a chain of transformations each addressing only one concern so that it becomes easy to implement and to reuse. However most of the current MDD practices imply MDA's monolithic forward PIM-to-PSM and PSM-to-code transformations.

We believe that it is better to feed a model to a chain of several transformations that each manipulate the model with regard to one specific functional, non-functional or technical concern. Multiple transformations would allow us to better modularize the transformations themselves. As a consequence the individual transformations would be easier to implement and reuse. When introducing

a transformation chain we should not only identify the transformations, but also pay attention to their interdependencies in order to obtain composable, loosely coupled transformations.

**3.2.3 Model Interchange and Storage** In order to use MDD for software development, a wide range of model transformations need to be applied and different tools need to be connected for performing all required operations. This creates the need for linking the output of a process step to the input of another process step. Due to the heterogeneity of tools in both functionality and the way users interact with them, connecting tools is very difficult.

OMG tries to solve this problem by introducing XMI, an XML standard for interchanging MOF-compliant models. However XMI versions 1.x are known to lack strong semantics which has forced each tool provider to interpret the standard differently. Versions 2.x are said to fix the issues from previous versions however no implementation results are available yet.

An alternative to the OMG solution is to admit the heterogeneity of model representation and storage and try to implement a “model bus” [13] which realizes model interchange. The idea behind a model bus is to ensure functional and protocol connectivity. Functional connectivity means that the input and output of each transformation should have compatible metamodels in order to be connected. Protocol connectivity should ensure that transformation connections can be realized. In particular, the connected transformations must agree in a model representation form and interaction styles.

### 3.3 Traceability

Traceability is often associated with the tracking of requirements across all artifacts throughout the software development process. However, it can also refer to the logging of transformation operations and their source/target model element mappings. We define traceability as the ability to extract transformation history out of a transformed model. A specific model element can then be traced back to their originating artifact, which can be another model element, a use case, a requirement, etc.

We can distinguish between generic/full traceability and specific traceability. Full traceability could be automatically accomplished by the transformation engine by linking every changed element in the output model to its counterpart in the source model. This makes traceability complete but not necessary very usable in subsequent transformations since the created links are tightly coupled with the particular implementation of the previous transformation. Specific traceability does not aim to link every source/target tuple but rather aims to form tuples that have a more semantically rich meaning without overcrowding that traceability model. Such specific links can be used more easily across transformations, but may require the developer to insert them manually. It is also not very clear how to determine which kinds of specific links could be useful for subsequent transformations.

Related to the levels of traceability is the issue of standardization of traceability models. It is important to think about what information exactly will be stored in the traceability model. For example, do we want to link as far as textual requirements, do we want traceability across different types of models, do we want to record responsible developers, etc. The advantage of having a single standardized traceability metamodel is that every transformation can always understand the included information, while the disadvantage probably would be its genericity. A possible solution is to have a basic but extensible traceability metamodel that can be adapted to specific needs while still allowing interoperability and interchange between tools and people.

Finally there is the issue of representation of traceability links and integration of traceability in models. Should we store traceability information inside our models themselves (intra-model) or rather externally in a separate traceability model that refers to the elements of the former model (extra-model)? One could argue that the intra-model approach, which can for example be realized with profiles in the UML case, leads to a certain pollution of the model. In the extra-model approach, we need a mechanism to refer to model elements from within our traceability model, for example unique identifiers. In this case a problem would be keeping both models synchronized. Using two separate models can potentially make transformations more complicated since they need to take an extra input if they want to use traceability information.

### 3.4 Integration of Existing Software

Programming languages usually provide a rich set of libraries which contain implementations of complex mathematical functions, different algorithms, text manipulation, etc. These libraries of existing functionality can save developers a huge amount of time. It is however not always very clear how to make these functions available to the modeler. Two possible approaches that are taken by tool manufacturers are the following:

- Provide a domain abstraction layer that captures all knowledge on these API calls. This approach does not scale and each time a new programming language or API version arises the tool should be modified to include the new mappings.
- Reverse engineer the whole language API into a model. This results into a rather huge and unstructured API model. Moreover we can reverse engineer only to the lowest level of abstraction, and thus cannot easily use higher-level API-related design patterns in the PIMs.

Besides integration of standard libraries it is also important to consider the integration of COTS software and legacy applications into new MDD efforts. Existing systems often serve as a starting point for new developments since they have proven their strength and are considered to be “trustworthy”. Therefore it is unrealistic for an MDD project to assume starting from scratch. COTS and legacy system integration is often a matter of representing existing interfaces in

our modeling environment using adequate wrappers. However, it is not always clear how to do this. Should we just take the raw interfaces (e.g. by capturing them in UML interfaces) or should we hide the interfaces behind a domain concept? Many problems in this field are worked on by OMG's ADM (Architecture Driven Modernization) task force.

## 4 Conclusion

MDD is based on the idea to describe the software using a model and apply an automated transformation which creates the source code from the model. MDD is a generic paradigm which does not aim to specify how and which models and transformations should be specified. MDA and MIC are specific visions of MDD which present different modeling frameworks and guidelines. It is still unclear at this moment whether to favor a Meta-metamodel based MDD approach above a UML Profile-based MDD approach.

Projects that have successfully applied MDD do exist, but they tend to fill the gaps in an ad-hoc way. In order to obtain the benefits from a full MDD Software Development Process for Real-Time Embedded System Development, there are a number of research challenges that still have to be addressed adequately, including proper standardization and tool support, in the field of modeling levels, model transformations, traceability and existing software integration.

## References

1. Model-Integrated Computing - <http://www.isis.vanderbilt.edu/>
2. "OMG/RFP/QVT MOF 2.0 Query/Views/Transformations RFP" - <http://www.omg.org/docs/ad/02-04-10.pdf>
3. OMG RFP: MOF Model to Text Transformation RFP
4. Atlas Transformation Language - <http://www.sciences.univ-nantes.fr/lina/at1/>
5. QVT-Partners : MOF Query/Views/Transformations - <http://www.omg.org/docs/ad/05-03-02.pdf>
6. Velocity Templates - <http://jakarta.apache.org/velocity/docs/vtl-reference-guide.html>
7. The General Modeling Environment - <http://www.isis.vanderbilt.edu/projects/gme>
8. GME-MOF: A MDA Metamodeling Environment for GME - M. Emerson
9. "A Critical Analysis of MDA Standards through an Implementation: the ModFact Tool" - X. Blanc, S. Bouzitouna and M.-P. Gervais
10. SysML - <http://www.sysml.org/>
11. ProMARTE - <http://www.promarte.org>
12. "Executable UML" - S. Mellor, M. Balcer
13. "Model Bus: Towards the interoperability of modeling tools" - X. Blanc, M.-P. Gervais, P. Sriplakich