MODEL DRIVEN PROCESS FOR REAL-TIME EMBEDDED SYSTEMS SOFTWARE DEVELOPMENT AND VALIDATION

Samaa A. Abdel Samie

Department of Computer Systems, Faculty of Computer and Information Sciences, Ain Shams University, Cairo, Egypt

ABSTRACT

Embedded systems shape our world nowadays. It's almost impossible to imagine our day to day life without it. Examples can include cell phones, home appliances, energy generators, satellites, automotive components …etc. it is even far more complex if there are real-time and interface constraints.

Developing real-time embedded systems is significantly challenging to developers. Results need not be only correct, but also in a timely manner. New software development approaches are needed due to the complexity of functional and non-functional requirements of embedded systems.

Due to the complex context of embedded systems, defects can cause life threatening situations. Delays can create huge costs, and insufficient productivity can impact the entire industry. The rapid evolution of software engineering technologies will be a key factor in the successful future development of even more complex embedded systems.

Software development is shifting from manual programming, to model-driven engineering (MDE). One of the most important challenges is to manage the increasing complexity of embedded software development, while maintaining the product's quality, reducing time to market, and reducing development cost.

MDE is a promising approach that emerged lately. Instead of directly coding the software using programming languages, developers model software systems using expressive, graphical notations, which provide a higher abstraction level than programming languages. This is called Model Based Development (MBD).

Model Based Development if accompanied by Model Based Validation (MBV), will help identify problems early thus reduce rework cost. Applying tests based on the designed models not only enable early detection of defects, but also continuous quality assurance. Testing can start in the first iteration of the development process.

As a result of the model based approach, and in addition to the major advantage of early defects detection, several time consuming tasks within the classical software development life cycle will be excluded. For embedded systems development, it's really important to follow a more time efficient approach.

KEYWORDS:

Embedded Systems,Model Driven Engineering, Model Driven Development, Model Driven Testing

1. EMBEDDED SOFTWARE DEVELOPMENT

Embedded systems shape our world nowadays. It's almost impossible to imagine our day to day life without it. Examples can include cell phones, home appliances, energy generators, satellites, automotive components …etc. it' even far more complex if there are real-time and interface constraints.

As an example, the following figure shows the evolution of automotive embedded systems since 1985

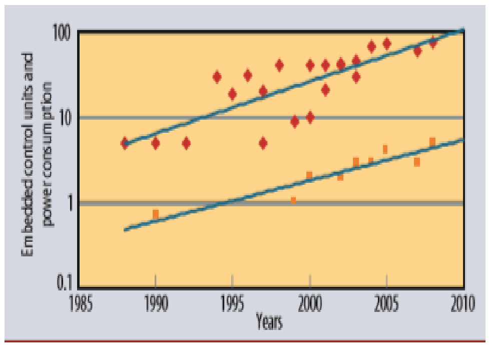

Figure 1: Automotive embedded systems

Embedded software is not a stand-alone product. It's a single element in a product that consists of mechanics, electronics, and software. This is why embedded systems development must consider various constraints.

Embedded software developers should consider resource limitations and environmental conditions, thus creating product-specific platform. The need to create specialized hardware and software platforms is a major difference between IT systems and embedded systems.

Developing real-time embedded systems is significantly challenging to developers. Results need not be only correct, but also in a timely manner.New software development approaches are needed due to the complexity of functional and non-functional requirements of embedded systems.

Non-functional requirements are more important in embedded systems than IT systems, such as software safety, reliability, and timeliness, which should be integrated into the development process.

Also, embedded systems developers require deep domain knowledge. For example, developing a vehicle control system is impossible if one doesn't understand the vehicle dynamics.

Embedded systems are found anywhere. They are information processing systems embedded into products. The growing complexity in embedded systems require the use of more hardware and software components to implement all the functionalities. Such increasing functionality leads to a growing design complexity which should be managed properly. As the complexity of embedded systems keeps growing, software quality problems arise. The following figure shows the complexity growth of embedded systems.

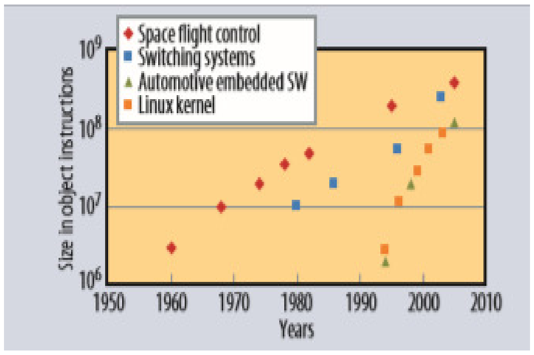

Figure 2 - Complexity growth of Embedded Systems

Due to the complex context of embedded systems, defects can cause life threatening situations. Delays can create huge costs, and insufficient productivity can impact the entire industry. The rapid evolution of software engineering technologies will be a key factor in the successful future development of even more complex embedded systems.

Embedded software is highly specific to its environment, thus posing extraordinary challenges to the software engineers. The main source of complexity is the large number of interactions among the various parts of the system, which have the following common features:

- Functionality represented by states and events
- Real-Time behavior
- Combined hardware/software systems
- High demand on availability safety, security, and interoperability
- Long lived systems in which embedded software is expected to work reliably.

1. MODEL DRIVEN ENGINEERING

The main principal of Model Driven Engineering is that "Everything is a Model". Modeling, in the broadest sense, is the cost-effective use of something in place of something else. It is an abstraction of reality in a simpler, cheaper, and safer way. It helps avoid the complexity and irreversibility of reality. Although models are used in various engineering domains, they only recently played a role in the development process of embedded systems.

1.1 Model Driven Development

Software development is shifting from manual programming, to model-driven engineering (MDE).One of the most important challenges is to manage the increasing complexity of embedded software development, while maintaining the product's quality, reducing time to market, and reducing development cost.

Model Driven Development (MDD), as any other development activity, should controlled by a development process. A lack of tools support makes it very difficult to cover the complete development life cycle using MDE.

Ensuring product's quality is crucial in such development, particularly in safety-related systems. Applying validation and verification techniques before code generation can ensure software quality.

To improve the automation of the design from the initial requirement specification until the final system, researchers look for modeling methods to specify, analyze, and verify embedded systems in a fast and accurate way.

A model must clearly communicate its purpose and must be easy to understand and develop, in order to be reliable and effective. Computer Aided Software Engineering (CASE) tools emerged as an attempt to use models in order to rise abstraction and automate development tasks. They provided graphical representations for requirements specifications, and automatically generated code from them.

The pressure to reduce time-to-market and the growing design difficulties require new research efforts and approaches to cope with that. MDE is the current approach to raise the design abstraction level and improve portability, interoperability, maintainability, and reusability of models.

Model Driven Engineering aims to improve productivity and to facilitate the system's development by creating, maintaining, and manipulating models. A system is developed by refining models, starting from higher and moving to lower levels of abstraction until code is generated. A MDE approach can imply several different developers, modeling languages and tools that model the same system.

Challenges in MDE:

- **Quality of developed models**: model quality metrics have to be defined and agreed on.
- **Model at run-time**: the use of models can be extended to phases beyond the design and implementation. A core aspect is the research of code generation techniques that, if well accomplished, could lead to a considerable reduction in the coding effort.
- **Requirements modeling**: all requirements have to be modelled, and traceability between written informal specification and formal models has to be maintained.
- **Modeling languages**: languages, methods, and tools have to be selected to fit all needs of the product that has to be created. That will be the basis of further modeling phases.
- **Model verification and validation**: a way to verify, validate, and test models and generated code has to be found and automation of test cases generation is a core issue.

Designing and building embedded systems is a challenge. Systems not only need to fulfill their functional requirements, but also their runtime and performance requirements. To help increase confidence in embedded software architectures, designers often turn to executable model based approaches to assess their systems.

Due to the increasing complexity of embedded software development over time, organizations have invested in modeling, starting by early approaches till the establishment of Unified Modeling Language (UML).

UML is a standardized modeling language consisting of a set of diagrams. It is now an international standard. The system is mainly modeled using class diagrams and state machines where an action language is used to describe the actions to be executed within each state, and on the transitions between states. Events are used to trigger those transitions.

The promise of model based design approach is to develop design models and subject them to analysis, simulation, and validation prior to implementation. Performing analysis early in the development life cycle allows early detection of problems and thus fixing them at a lower cost.

1.2 Model Driven Validation

Embedded systems are present, and enormous in number. Their growing complexity calls for not only new design approaches, but also new powerful, automated and enhanced testing methods.
Finding and fixing a software problem after delivery is often 100 times more expensive than finding and fixing them during the requirements and design phase.

Finding and fixing bugs is overall the most expensive activity in software development. Embedded software tends to perform more kinds of testing than other forms. The following figure shows that the cost of fixing bugs rises exponentially with each phase of the software development life cycle.

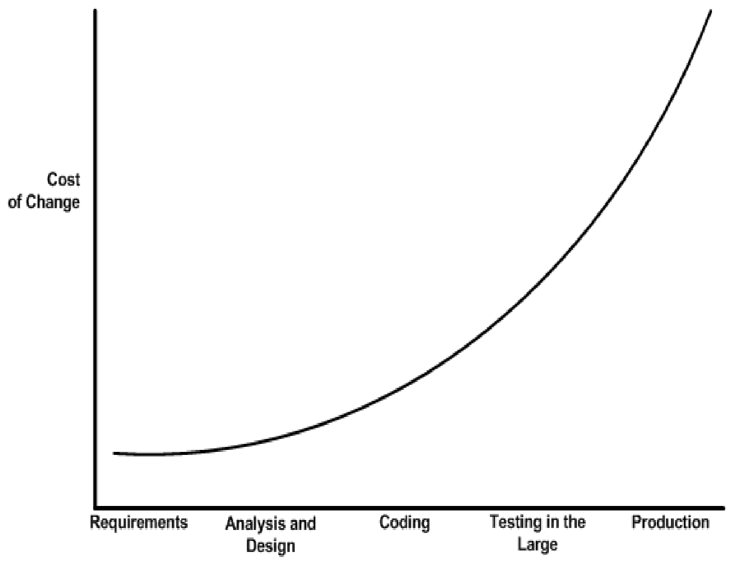

Figure 3 - Phases of SDLC vs. Cost of Bug Fixing

Model Driven Development if accompanied by Model Driven Validation will provide information at an early stage, before the system is implemented. It helps identify problems early thus reduce rework cost. However, models are usually generated manually thus require time and effort.
Specifically for embedded systems, late defect correction costs much more than that of other software types due to the close hardware interaction.The clear focus therefore is to detect defects as early as possible. Best would be in the phase where they are inserted.

The two major cost drivers in embedded software development are requirements and test.

- **Requirements**

  We often develop the wrong things due to lack of reviewing and analyzing requirements or missing and vague requirements. 40% of all software defects in embedded systems result from insufficient requirements and analysis activities.

  The typical effort allocated to requirements engineering is 3 to 7% of total project cost. Doubling this effort has the potential to reduce life cycle cost by 20 to 40%. The cost reduction mostly is due to earlier analysis and defects removal during specification and requirements verification.

- **Test**

  3o to 40% of the project's effort is allocated to testing. That's almost half the cost of the product. A major advantage would be, how to minimize test effort? The answer would be first, to detect defects close to the phase where they are created, and second, by removing redundancy.

  Model-Based Testing is the technology in which test cases can be executed on a model. After their execution, test cases can also be automatically evaluated and documented. In all cases, the user must work with a model and follow a systematic working method in testing.

For MBV, the following is necessary:

- **Requirements / Specification**: system's description is absolutely necessary to construct a model.

- **A model based on the requirements**: the actual step in the process is the implementation of the requirements in a model, which requires a huge effort. In the future phases of development, the actual software is derived from the system's model. To permit testing, a separate test model can be also developed.

- **Test execution environment**: to enable execution of the test scripts, a suitable environment must be provided.

  As the integration level changes during development (model, component, software, ECU), individual test execution environment also changes.

Four levels are explained here:

- **Model in the loop:** the developed model is subject to a test. In other words, inputs are changed, and outputs are monitored. Feedback is always needed to maintain model's compatibility with the original requirements. Many modeling environments offer the capability of executing tests on this level.
- **Software in the loop:** at this level, code is generated from the model. A partial manual implementation can be also added. Testing the code is then executed on a simulation environment.
- **Processor in the loop:** the code will be then added to the target processor, and testing is executed then.
- **Hardware in the loop:** now the physical environment is present, and the transition from PIL integration is smooth.

The traditional V-model is a commonly used approach as a software development life cycle. Its acceptance is due to its simplicity and ease of application. The following figure shows the different tests across the development v-cycle model

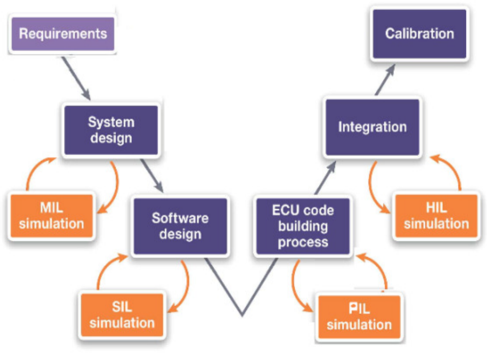

Figure 4 - MiL, SiL, PiL, HiL across V-Cycle

Two major points represent a challenge in MBV:

- The test sequence used on a specific level should be reusable.
- It must be possible to automate all test sequences, in order to achieve reproducible results.

The MBV approach can be described as a form of black box testing, where structured models are used to automatically generate test cases, thus automate the test cases design process.

In general, the testing activity can be classified into: 1) test cases generation, 2) test cases execution, and 3) test cases evaluation

The model driven approach to software development has not only changed the way software systems are built, but also the way they are tested. For such systems, model based validation is favored and recommended since it's aligned with the new model driven development technique that favors model over code with the overall objective is to reduce time to market, while maintain and improve product's quality.

The main advantage of model based validation is increased quality and productivity by shifting the testing activities to an early stage of the software development life cycle and generating test cases that are independent of any implementation.

Very few testing model based validation techniques use the developed models as a testing target. While all other techniques focus on testing the implementation through the test cases generated from the models.

II. PROPOSED APPROACH

A technique to deploy MBV for embedded systems without affecting their real time behavior is poorly supported. Our approach is to provide a tool chain to support MBV automation, specifically during MIL.

Our proposed approach consists of the following steps:

1- **Model Based Development**: in this phase, a model is developed based on the requirements. A general overview of the system can be modelled using class diagrams, and then the detailed behavior can be represented by means of statecharts, with states and transitions between them.

2- **Model based validation**: based on the model obtained from step 1, a test model is created as a first step in MBV approach. In our approach, the test model is a series of statecharts, each representing one test scenario to be executed on the original model. Each test scenario consists of a series of inputs, and expected outputs.

During step 2, the proposed tool chain will include:

- Automatic test cases execution, where the developed test statecharts are automatically executed on the system's model.
- Automatic test cases evaluation, where the execution results are documented and analyzed.

The tool chain will allow the model to be tested in its original form, to test its functional behavior and conformance to the software requirements. After the model is tested and stabilized, conversion process can then start to auto-generate the code to be used in SIL.

The proposed flow is shown in the following figure. First, the requirements/specifications are interpreted by both, developers and testers. The developers will construct the system model, and the testers will construct the test model. After that and as a first step, the test model will be executed against the developed system's model and the results will be analyzed. During this stage, continuous feedback is needed, and ongoing system model and test model corrections are made, to ensure compatibility between the requirements and the model.

After that, the code can be generated from the model, which will then has a high degree of reliability. The same test scripts generated by the testers can now be executed on the generated code, either on a PC-simulated mode (SIL) or on the target processor (PIL).

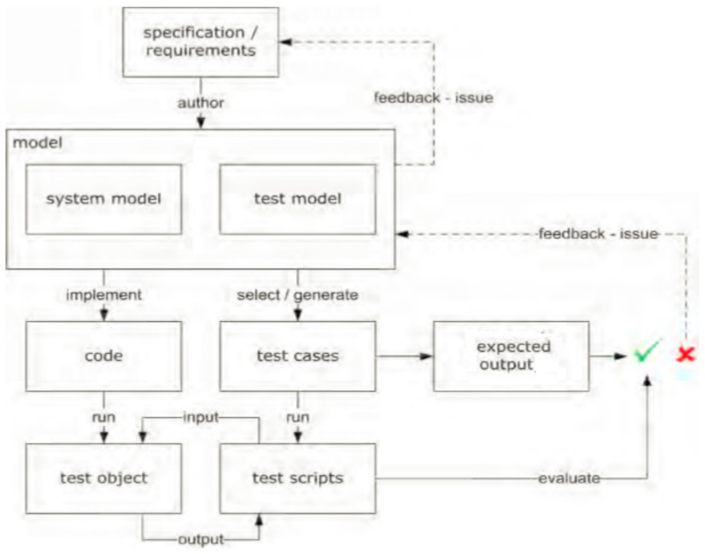

Figure 5 - Overview of MBV approach

Note that it's preferable to keep the development and testing tasks separate, allowing for different interpretation of the requirements. This normally helps discovering specifications errors and ambiguity.

As shown in the below table, model based approach excludes several time consuming tasks within the classical approach. For embedded systems development, it's really important to follow a more time efficient approach.

The following table shows a brief comparison of the proposed approach and the classical development workflow regarding design, coding, and testing. Requirements specification is the same in both approaches.

	Model Based Development and Testing flow	**Classical development and testing flow**
Design	Not needed, as requirements are modeled, and code will be generated automatically	Effort of creating design document is required
Coding	Code auto generated, review and analysis efforts are minimal	Effort of source code implementation, review, and analysis is required
Testing	Only modeling effort is needed. Testing is done on the model level using proposed tool chain. Same test scenarios can be then used on generated code and on the hardware.	Effort of creating test documents, scenarios, and implementation is needed at every stage.

Table 1 - proposed vs classical flow

REFRENCES

[1]. M. -L Boukhanoufa, A. Radermacher, F. Terrier, “Towards a Model-Driven Engineering approach for developing adaptive real-time embedded systems”, Pages: 261-266, NOTERE 2010

[2]. P. Liggesmeyer, M. Trapp, “Trends in Embedded Software Engineering”, Volume: 26 , Issue: 3, Pages: 19-25, 2009

[3]. F. R. Wagner, F. A. M. Nascimento, M. F. S. Oliveira, “Model-Driven Engineering of Complex Embedded Systems: Concepts and Tools”

[4]. C. Ebert, C. Johnes, “Embedded Software: Facts, Figures, and Future”, Volume: 42, Issue: 4, Pages: 42-52, 2009

[5]. F. Ciccozzi, “Model-Driven Engineering of Embedded Real Time Systems”, Research Planning Course 2009/2010

[6]. M. Tischer, D. Widmann, “Model based testing and Hardware-in-the-Loop simulation of embedded CANopen control devices”, Pages: 06/19-06/28, iCC 2012

[7]. L. J. BenAyed, Y. H. BenDaly, “Translating Graphical Conceptual model from Statemate to FNLOG”, Pages: 1801-1806, IEEE 2007

[8]. J. S. Fant, H. Gomaa, R. G. Pettit, “A Comparison of Executable Model Based Approaches for Embedded Systems”, Pages: 16-22, SEES 2012

[9]. J. Gong, “A UML Based Design Methodology for Real-Time Embedded Systems”, Pages: 776-779, 2002

[10]. P. Iyenghar, C. Westerkamp, J. Wuebbelmann, E. Pulvermueller, “An Architecture for Deploying Model Based Testing in Embedded Systems”, FDL 2010

[11]. C. Shih, C. T. Wu, C. Y. Lin, H. P. A. Hsiung, N. L. Hsueh, C. H. Chang, C. S. Koong, W. C. Chu, “A Model Driven multicore software development environment for embedded systems”, Volume: 2, COMPSAC 2009

[12]. M. Mussa, S. Ouchani, W. Al Sammane, A. Hamou Lhadj, “A survey of model driven testing techniques”, QSIC 2009

[13]. C. Bunse, H.-G. Gross, Peper, Christian, “Applying a model based approach for embedded systems development”, EUROMICRO 2007

[14]. Z. Micskei, I. Majzik, “Model-based Automatic Test Generation for Event-Driven Embedded Systems using Model Checkers”, DepCos-RELCOMEX 2006

[15]. H. Kashif, M. Mostafa, H. Shokry, S. Hammad, “Model-Based Embedded Software Development Flow”, IDT 2009

[16]. L. Tan, J. Kim, O. Sokolsky, I. Lee, “Model-based Testing and Monitoring for Hybrid Embedded Systems”, IRI 2004

[17]. B. Boehm, V. R. Basili, “Software defect reduction, top 10 list”, vol. 34, no. 1, pp. 135-137, January 2001