Online ISSN 1848-3380 , Print ISSN 0005-1144
ATKAFF 52(4), 329–338(2011)

Josip Babić, Siniša Marijan, Ivan Petrović

Introducing Model-Based Techniques into Development of Real-Time Embedded Applications

UDK 519.6: 681.515
IFAC 4.0.5

Original scientific paper

This paper investigates the feasibility of integrating legacy software processes and tools into the paradigm of model-based development of industrial real-time embedded systems. Research has been conducted on the example of using legacy assembly code for automatic code generation scheme inside MATLAB/Simulink environment. A sample Simulink model has been presented, code has been generated from it and its correctness has been validated by back-to-back comparison with the simulation results.

Key words: Model-Based Development, Real-Time Embedded Systems

Uvođenje tehnika zasnovanih na modelu u razvoj aplikacija za ugradbene sustave s vremenskim ograničenjima. Ovaj rad ispituje mogućnost integriranja naslijeđenih procesa i alata za razvoj programske podrške namijenjene industrijskim ugradbenim računalnim sustavima s nametnutim vremenskim ograničenjima u paradigmu razvoja zasnovanog na modelu. Istraživanje je provedeno na primjeru korištenja naslijeđenog asemblerskog programskog koda pri automatskoj generaciji izvršnog koda unutar MATLAB/Simulink okruženja. Prikazan je primjer Simulink modela iz kojega je generiran kod čija je ispravnost utvrđena usporedbom s rezultatima simulacije.

Ključne riječi: razvoj zasnovan na modelu, ugradbeni sustavi s nametnutim vremenskim ograničenjima

1 INTRODUCTION

Embedded systems are used in an ever growing number of applications, from simplest toys to highly complex industrial, military and space systems. Along with faster and more complex hardware, embedded software is gaining importance and makes up to 85% of the value of the entire embedded system [1]. Under the market pressure the software must be produced quickly and as bug-free as possible. Driven by these two opposing requests, Model-Based Development (MBD) has emerged as the design approach of choice. As stated in [2], MBD is not just application of graphical domain-specific languages, i.e. "programming by drawing". A good definition of MBD is given in [3], and a part of it is reproduced in the following:

In model-based development, the model is the central artifact and is used and systematically refined through the entire development process which is literally based on or centered around it.

When considering MBD of embedded applications it is necessary to take into account various specific characteristics of these systems. They are usually reactive, meaning that they interact either with their environment or some larger system, and can contain continuous and discrete subsystems, or both, [3]. Embedded systems control and/or monitor a specific device or function, they are self-starting and self-contained. If, besides functional requirements, the correctness of embedded system operation depends on its timeliness, they are said to have (soft or hard) real-time constraints, [4]. Many industrial embedded systems, to which results of this work are intended to be applied, fall into hard real-time category with severe timing constraints.

As many industrial embedded systems are safety critical, it is no wonder that certain resistance towards new methodologies exists among the development community. Time tested and proven products and processes are hard to discard, and should not be neglected lightly. In them, many man-years of development and testing were invested and immeasurable amount of knowledge and experience is condensed. That is why they should be preserved and integrated into MBD process, to the greatest extent possible. This issue is addressed in our work by integration of existing software development process into MBD paradigm. The existing process is based on the proprietary integrated development environment (P-IDE) and has been used for development of applications in rolling stock and power engineering for a number of years, [5–7].

The paper is structured as follows. Second section introduces the existing embedded software development pro-

cess and tools, outlines their advantages and drawbacks. Third section provides overview of works on MBD of embedded applications. Fourth section describes a way to fit the existing development process into the MBD process, and constitutes the core of this work. At the end, conclusion and outlooks for future work are given.

2 EXISTING DEVELOPMENT PROCESS

2.1 Integrated development environment

Existing development process evaluated in this work is based on a proprietary integrated development environment consisting of: Graphical Application Programming Tool (GRAP), microcontroller specific project structure and proprietary RTOS (Real-Time Operating System) and a PC-based service and debugging application ZZT. The graphical environment is used for the development of application programs (AP) that are to be executed on hard real-time systems and it supports several different architectures of microprocessors, microcontrollers and signal processors, [5–7].

GRAP is used for constructing and editing APs and also for invoking compilation and linking tools. This is a completely graphical environment, so there is no need for textual coding. SW components on higher level of complexity are built by interconnecting basic SW components represented by graphical symbols, similar to block diagrams. On a code level, each basic component is a macro program, hand-written in assembly language, carefully optimized and thoroughly tested. Macro library, as well as compilation and linking tool for the given target, are included in a project structure. This is a folder structure that besides library directory contains folders with application files. Successfully built AP is loaded onto the target, on which RTOS is running, using ZZT.

2.2 Application development process

Application, as considered in this work, consists of one or more APs, each executed on a separate microcontroller based HW module (HWM), that together perform desired functionality. The development of an application can be observed in several phases, Fig. 1. Firstly, the project leader (PL) divides the functionality of the application between the HWMs, defines interfaces between them and assigns the development of each application program to one application engineer (AE). AE divides his AP into functional modules (FM), defines their interfaces and distributes their development to module developers (MD). Individual FMs can't usually be thoroughly tested on their own so, after more or less partial testing by test engineers (TE), they are handed to the responsible AE for integration into AP. When AP is successfully integrated, i.e. all FMs are compiled and linked, limited amount of testing by TEs is performed on laboratory equipment. In order to fully test APs, they need to be integrated into final application by PL and run on the target hardware.

Depending on the complexity of the project, PL can play the role of AE, i.e. be responsible for one or more APs, and AE can play the role of MD, i.e. he can develop some or all modules of the AP he is responsible for. All through the development, PL, AEs and MDs are supported by system engineers (SE). Their responsibility is the maintenance of development environments of individual HWMs and the development of new basic SW components, if requested by AEs or MDs. The described application development process is illustrated by Fig. 1, development roles are depicted by light grey circles and software artifacts are represented by dark grey squares.

Parallel to the application development, hardware development process is conducted. Proprietary HW platform is a modular one, so existing modules are used where possible and new modules are developed if necessary. Hardware development is not in the scope of this work and will not be discussed further, but details can be found in [8].

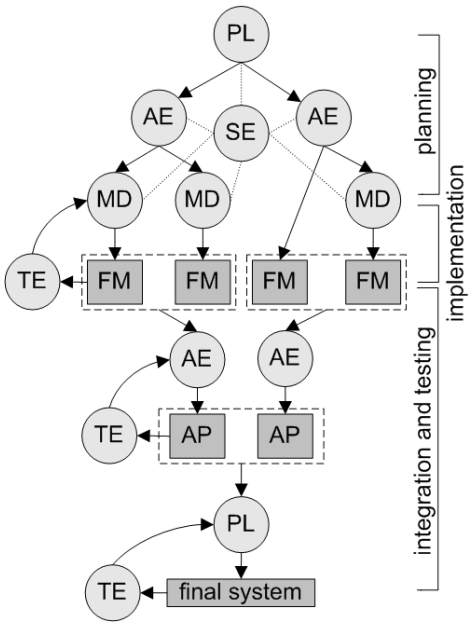

Fig. 1. Existing software development process

2.3 Advantages and drawbacks

Two main characteristics of the P-IDE are graphical programming language and usage of assembly macros as basic SW components. The first trait brings higher level of abstraction, in comparison with textual programming languages, which enables non-software engineers to implement their ideas in a target independent manner. Modularity is accomplished through the division of AP into FMs.

As applications are built completely in graphical environment, without a single line of handwritten code, the program can serve as the documentation. Every functionality and parameter is visible in the block diagram scheme so that, when the AP is completed, it can simply be printed and supplemented with minimal additional documentation.

Usage of assembly macros for code generation brings low-level control of all processes on the target, enables the creation of highly optimized [9] and thoroughly tested basic components and introduces another level of modularity. By creating own code generation scheme, substantial knowledge is accumulated and kept in-house. SEs with such knowledge have shown to be invaluable when it comes to supporting AEs during AP debugging.

The described existing application development process is flawed by two major drawbacks: by the absence of strong link between requirements management (REQM) and P-IDE and by inability to perform complete functional tests of FMs before their integration into APs, and of APs before their deployment into final system. The requirements on software artifacts, starting from basic components up to full APs, are often informal or semi-formal, always in prose and in a separate document that has no automatic link to the respective artifact. This approach might lead to misunderstandings that result in more development iterations than necessary.

Algorithmic parts of FMs can be independently tested by replaying pre-recorded signals at their inputs and recording and analyzing the results. The drawbacks of this kind of testing are: FM must be altered (IO parts removed and signal playback added), closed control loops are broken and the tests must be performed on the target processor. Complete target HW configuration often isn't available during the initial stages of AP development, so testing is performed on evaluation boards and laboratory hardware setups. These resources are scarce and can replicate the final system only to a limited extent. All this sums to the fact that complete and thorough software tests can be performed when most of the HW and SW is available, late in the development process when bugs are most expensive to correct.

3 MODEL-BASED DEVELOPMENT

MBD process is illustrated in Fig. 2. It starts with requirements elicitation and formalization, proceeds with modeling and code generation. Two-way traceability between various development artifacts should be ensured – from requirements through model to code and test cases. Testing is performed throughout the development: model is checked against requirements while code, executed on PC or on the target, is checked against the model.

3.1 Requirements Management

Requirements on embedded systems are specific because, besides functional requisites, often a number of non-functional requirements and constraints must be satisfied. Non-functional requirements can relate to performance, safety and availability. In [10] requirements traceability is achieved by defining four links between requirements and between requirements and other modeling artifacts: *derive* represents derivation of requirement from another, *refine* indicates that an element is a refinement of a textual requirement, *satisfy* shows the satisfaction of requirement by design and *verify* links requirement with test case that verifies it.

Stronger link between requirements and testing is proposed in [11], where requirements are formalized by constructing *Timed Usage Models*, out of which tests are automatically generated. Motivation of this approach is to alleviate greatest drawbacks of current REQM methodologies: lack of systematical analysis and informal and ambiguous requirement description.

Requirements traceability is the focus of DARWIN4Req approach presented in [12]. This is a metamodel that establishes two-way links between three independent and heterogeneous models: requirement model, solution model and verification and validation model.

Tighter integration of REQM and design is achieved by AutoRAID extension to AutoFOCUS MBD development environment, [13]. Here, requirements are classified as use cases and architectural, modal or data type constraints and their stepwise refinement leads to consistent and complete *Requirements Engineering Product Model*. Elements of the design are produced from requirement model using *motivate* and *associate* functions. This approach ensures seamless transition from REQM to design and enables requirement analysis; a number of automatic consistency checks are described in the paper.

3.2 Modeling

In context of MBD, system design is performed by modeling. Various approaches to modeling methodology are available. In already mentioned AutoFOCUS [14] environment, separation of concerns is achieved by graphical modeling based on views: *Data Definition View* defines data types and basic functions as basis for further development, *System Structure View* describes system structure including components, interfaces and communication channels and *Behavior View* captures the behavior of each atomic component.

In order to generate implementation from the model, it must either be enriched with platform specific information or linked to external source of such information. First approach reduces potential for reuse so in [15] and [16] separate modeling of functionality and HW/SW architecture

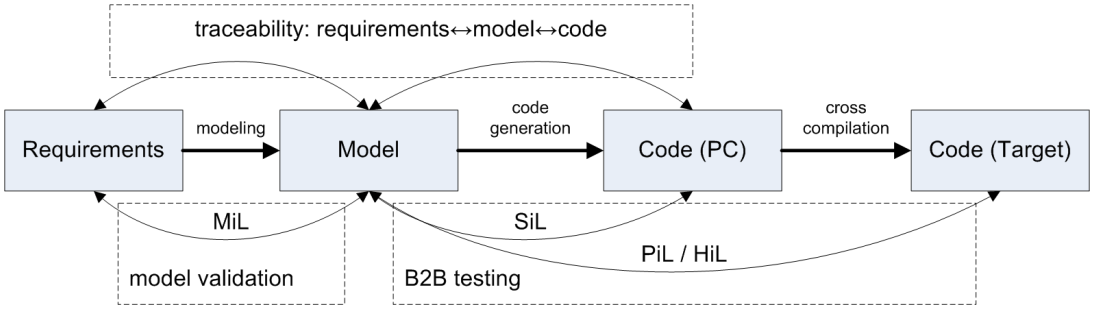

Fig. 2. Model-Based Development Workflow

is proposed. This way, independent and reusable functional and platform models are constructed and a set of rules, called model compiler, performs mapping between them. Similar approach called *Architecture Driven Development* presented in [17] consists of the following steps: 1) system architecture definition, 2) functional model construction, 3) HW/SW architecture definition, 4) mapping of functional architecture to HW/SW architecture, and 5) code generation.

Desired shift toward higher level of abstraction in MBD may be missing because many modeling languages (ML) are based on textual programming language concepts. With this motivation [18] introduces *Model Integrated Computing* methodology where system development starts with domain specific ML definition, after which necessary tools are constructed and finally system design is conducted. This approach ensures that model artifacts represent domain elements, and not the code. Model interpreters that translate domain specific models to simulation, analysis and implementation models are crucial here.

3.3 Model transformations

Model transformations can be conducted by direct model manipulations, through intermediate representation or using transformation languages. Direct manipulation of internal model representation is performed using standard procedural languages, but they lack abstraction so transformations are difficult to write and maintain. Intermediate representation for model transformation can be some standard format, as XML, or a specific formalism, as *Synchronous Reactive Model of Computation* that enables definition of inherently correct transformations, [19]. According to [20], domain specific languages for model transformations, e.g. ATLAS (*Atlas Transformation Language*) and GReAT (*Graph Rewriting and Transformation*), represent the best alternative. GReAT, [18], treats models as graphs and performs transformations by linking input and output metamodel into unified metamodel and by defining transformations on the new metamodel.

3.4 Automatic code generation

A seamless transition from models to executable code is accomplished by automated code generation (ACG) techniques. Code must be generated from the model consistently with limited use of a subset of the programming language that is considered to be safe and with limited use of well specified control and data structures. It should comply to specified complexity measures and it must be maintainable, testable, stable, changeable and analyzable, [16].

As stated in [4], model-based code generators differ from conventional compilers in that for them both source and target language is executable, i.e. it is possible to compare simulation results with code runs. Another difference is that ML semantics is often not explicitly defined but is instead embodied in the interpretation algorithms of the simulator. Finally, direct transformation of hierarchical model structure to syntax tree of the target language is not possible; automatically generated code is based on a sequence of computation derived by analyzing data dependencies of the model. In the same work, the issue of confidence in code obtained through ACG is raised, similar as was for compilers when transition from assembly to higher languages was taking place. Due to short development cycles and limited user base, validation of ACGs by exploitation is not applicable. By certifying generators, fitness for purpose can be guarantied, but only under strictly defined conditions of use. Third option is code generator testing that must be automated to be feasible because of great number of model variants and frequent new versions.

3.5 Model-based testing

It has been reported in [1] that the cost of finding and fixing defects grows exponentially in the development cycle so it is important to start testing as soon as possible. According to [21], up to 50% of the development effort of the critical systems is taken up by the testing process that can be reduced up to 50% by using model-based testing (MBT) techniques. This term is used to describe all testing activities in the context of MBD projects. According to [22], MBT is cheaper, faster and almost as effective

in terms of code coverage in comparison with traditional manual testing.

A generic MBT process can be described in five steps: system under test (SUT) model creation, definition of test selection criteria, test case specifications creation, test suite generation and test case execution, [23]. Because the same tests are repeated on various integration levels throughout development process, test automation is crucial. In [24] six such levels are defined, in the context of automotive industry: 1) *Model-in-the-Loop* (MiL), model and its environment are simulated, 2) *Software-in-the-Loop* (SiL), generated code is tested within a simulated environment, 3) *Processor-in-the-Loop* (PiL), embedded software runs on a target processor, 4) *Hardware-in-the-Loop* (HiL), software runs on the final hardware while environment is still simulated, 5) *test rig*, environment consists of physical components including special measurement and analysis equipment, and 6) *car*, finally software is tested in the car.

Requirements on the MBT process are enumerated in the same work. Testing should be automated due to interdisciplinary and iterative nature of MBD. To facilitate reuse, tests should be transferable between integration levels and means to compare test results from various levels should be provided. Such comparison is called back-to-back (B2B) testing. Systematical test design is crucial to achieve satisfactory coverage without redundancy and to keep track of hundreds, or possibly thousands, of test cases. Tests must be readable to allow stakeholders from different domains to participate in testing process. Closed loop or reactive testing, where test cases are dependant on system behavior, is preferable. To enable tests on target integration levels, timing constraints should be satisfied.

Classification tree method [25–27] is often used in MBT to generate test cases. It is performed by input domain partitioning into equivalent classes based on uniformity hypothesis. Using so obtained classification tree, combination table is constructed and test cases are designed by choosing combinations of classes based on various coverage criteria. Systematic, formal and transferable test design method called *Time Partition Testing* is presented in [24]. Functional tests are graphically modeled with state machines. Variation points enable transparent representation of a great number of different test cases with single diagram. In order to evaluate time-dependent signals in B2B testing of embedded systems, a concept for signal comparison and accompanying tool MEval are presented in [28]. Difference-matrix preprocessing algorithm implemented in this work allows independent evaluation of amplitude deviations and time shifts. Test design methodology during initial phases of design, when no referent system responses are available and no B2B testing is possible, is presented in [1]. Requirements are broken down to signal features and than a test harness, that consists of test data generator, SUT, test specification and test control units, is automatically generated by application of test patterns.

4 INTEGRATION OF EXISTING TOOLS AND PROCESS INTO MBD

In order to implement processes of the MBD, appropriate tool support is necessary. The in-house development of entire model-based tool environment would be exceedingly expensive, so integration of legacy tools and processes into an off-the-shelf environment has been evaluated. MATLAB tool family by MathWorks has been chosen because

- it is widely used in academia as well as in industry,
- it is easily extensible by a variety of toolboxes,
- Simulink (SL) toolbox enables creation of block diagrams and their simulation,
- powerful scripting language enables user-built extensions,
- there exists prior in-house experience with this family of products,
- in [3] it is shown how MATLAB can be used to create a complete MBD environment for industrial embedded systems.

To preserve knowledge accumulated in P-IDE basic SW components, they need to be used for code generation from SL models. For this work, a proof-of-concept procedure for generating "P-IDE Code" out of SL models has been developed. In essence, SL has been used as a replacement for GRAP in building graphical modules, their compilation and linking into executable applications. The approach has been verified for C2000 family of Texas Instruments digital signal controllers, but can easily be expanded to other P-IDE supported architectures.

4.1 P-IDE code generation

Generation of executable code from graphical application program in P-IDE is illustrated in Fig. 3. Firstly, the graphical application program is built using basic SW component symbols from a graphical library. Application can be divided into an arbitrary number of FMs, which can then be reused. Each FM has its own `.mdl` file, accidentally the same extension as for SL model files, but these two file types are not compatible.

From graphical FMs, `.src` source files are created. For every basic component symbol in a FM, a macro call with all the necessary arguments is placed in the source file. In

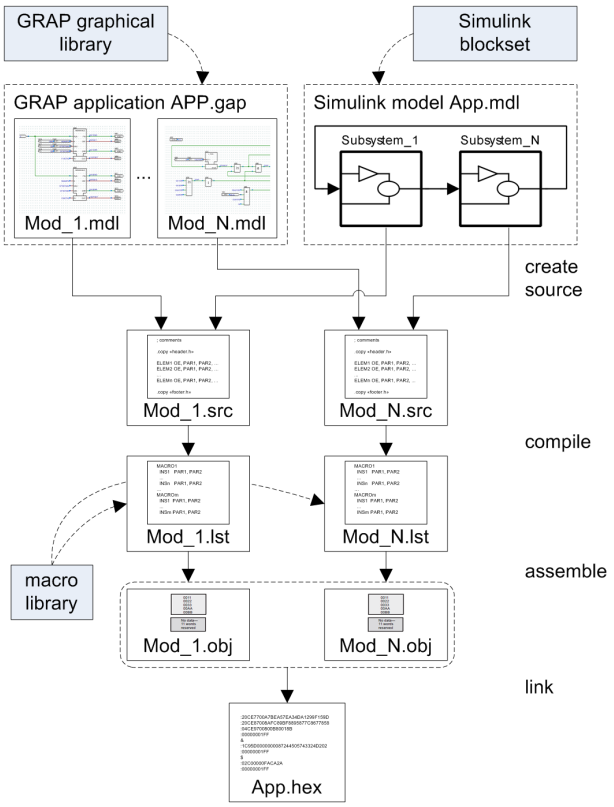

Fig. 3. P-IDE code generation

the compilation step, these macro calls are expanded using macro library resulting in a complete assembly listing of each FM. By executing assembler on these files, COFF (Common Object File Format) object files are generated that can be linked into executable `.hex` files.

Instead of graphical programming and code generation in GRAP, SL and MATLAB can be used as shown in Fig. 3. SL is used to create P-IDE compatible model and a set of M-functions enables its conversion into `.src` files. Further code generation is performed by calling the same tools as was case with GRAP, only here they are called from within MATLAB environment. The crucial step in this process is creation of source files appropriate for code generation from SL model, i.e. conversion of SL models into FMs. Assembled object files corresponding to FMs designed in both GRAP and SL can be linked together. This way, algorithmically intense parts of AP can be developed in MATLAB environment and linked to HW dependant input/output FMs or legacy FMs developed in GRAP.

The greatest advantage of using SL for AP creation is its simulation capability. Built-in SL components are simulated in usual fashion, while P-IDE specific components are modeled in a way to closely emulate their assembly counterparts so model of the AP should during simulation behave much the same as the generated code during execution on the HWM. Besides simulation, SL provides access to various other MATLAB features like REQM and model checking tools.

4.2 Simulink to P-IDE conversion

For a SL model to be convertible into P-IDE application, a number of restrictions must be obeyed. For example, only supported components, grouped into blocksets named by HWMs, can be used for model construction. Further, fixed-step discrete solver must be used and a few naming conventions are introduced: block and line names must not contain space characters and should be eight or less characters long, *Outport-Inport* block pairs that pass signals between subsystems must be named and have the same name.

The conversion utility consists of the main M-function that calls additional M-functions to perform conversion tasks. The tasks are: obtain data on all components in the model, remove redundant components like *Scope* and *Terminator* from the list, group remaining components into modules, resolve SL line (i.e. GRAP signal) names, construct macro calls, write source and application files, compile and link application. Name resolution is necessary because signals in P-IDE are passed between assembly macros by their names. If a signal in an AP is not named, GRAP assigns it a generic name. SL, on the other hand, connects components using tree-like line structure with branches and numerical designators. A SL line with all its branches is translated to a signal that is named based on source block name, parent line name or parent line designator.

4.3 Proof-of-concept implementation

Proof-of-concept code generation system has been developed for a HWM based on TMS320F28335 digital signal controller from TI's C2000 microcontroller family. Initial HWM Simulink blockset, Fig. 4, consists of seven built-in SL components and seven user-defined components.

To each built-in component a conversion M-function that deals with its specific properties has been assigned. For example, *Add* SL component is converted into one or into a combination of various P-IDE basic components, depending on number of inputs and *List of signs* parameter value. User defined components can be implemented with a masked subsystem consisting of built-in components, as is the case with IIR2C shown in Fig. 4, or they can be implemented by masked M-file S-function blocks. The two approaches were validated by comparing subsystem-type components MEMPLAY and RECORDER, and their functional M-file S-function counterparts MEMPLAYS and

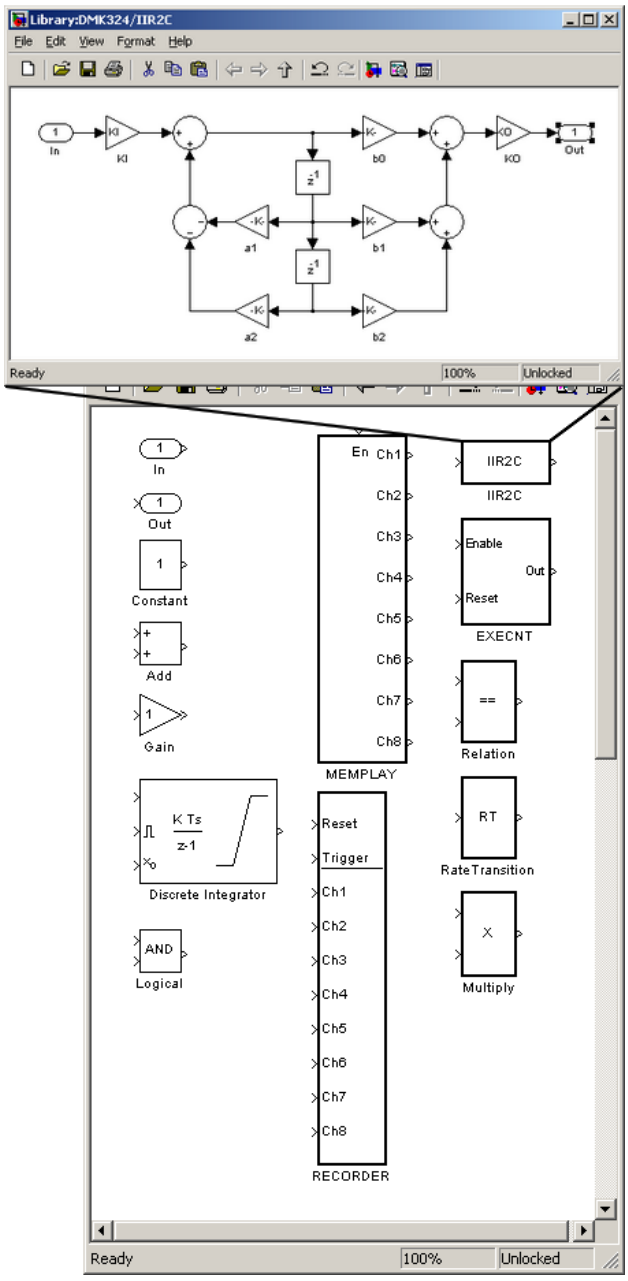

Fig. 4. Initial HWM SL blockset

RECORDERS. Although writing M-file component from scratch offers greater freedom in design, subsystem-type components have shown to be much faster during simulation so this was the preferred choice. Custom SL components are easier to convert than the built-in ones, because there exists injective mapping between them and P-IDE basic components.

Besides AP generation, tool support for communication between simulation environment and target is necessary. Microcontroller used for presented evaluation implementation enables real-time communication through JTAG emulation port using TI's *Real-Time Data Exchange* technology, [29]. This option however was not pursued, because the goal was universal solution applicable to HWMs based on microcontrollers of different architectures. All target HWMs posses service serial RS232 port, so M-file functions were developed that allow MATLAB to interact with a target using this communication channel. They can be used for loading AP onto target or retrieving HWMs memory content.

Generic PiL test harness, that can be implemented using presented tools, is shown in Fig. 5. It consists of *Input Generation* that generates test vectors, *Application Model* which represents SUT, *Code Execution* that refers to generated code executed on the target and *Output Evaluation* part that performs B2B comparison of simulation and code execution results. After model has been designed and satisfactory simulation results obtained, executable code can be generated and loaded onto target using described procedures. Code execution results are recorded to HWMs memory and retrieved through RS232 communication. Test results can be analyzed and evaluated when both simulation and execution results are available.

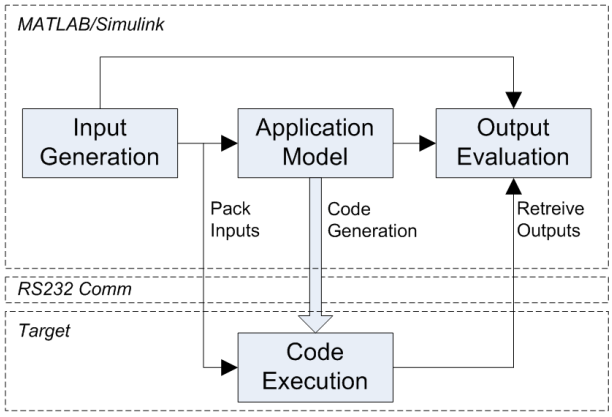

Fig. 5. PiL setup with RS232 communication

4.4 Experimental results

Validation of the approach has been conducted using test model in Fig. 6. It consists of MEMPLAY component, that reproduces signals from workspace (or from memory when executed on target), two IIR2C and two Integrator components, that together perform double integration of high-pass filtered signal, and RECORDER with auxiliary components, Constant, RateTransition and OutPorts. RECORDER stores signals at its inputs to workspace, when simulated, or to HWMs memory, when executed on target. Input signals for MEMPLAY have been recorded

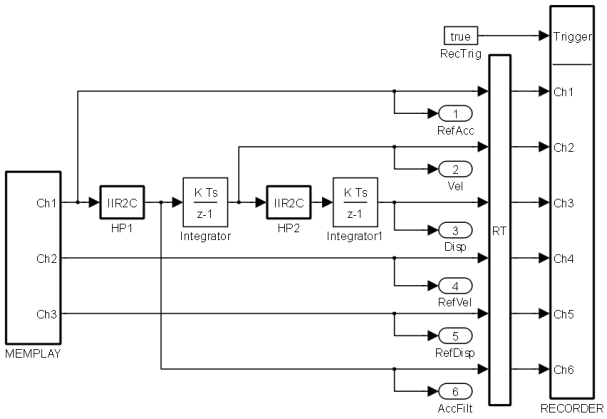

Fig. 6. Simulink test model

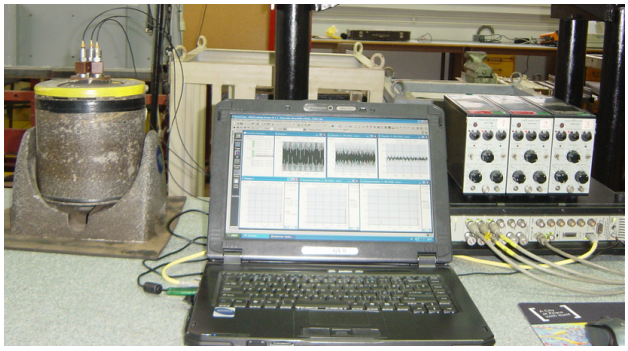

Fig. 7. Recording real-world signals

using laboratory model shown in Fig. 7. Signal recording setup consists of signal generator, vibrator with its amplifier, three accelerometers, each with one amplifier, and acquisition device. Accelerometer amplifiers can filter and integrate input signals, so this was used to record acceleration, velocity and displacement. Model in Fig. 6 is intended to emulate HW filtration and integration, that is to produce velocity and displacement from recorded acceleration signal.

High pass filters have been designed in MATLAB environment and simulated to compare results with recorded referent signals. After satisfactory results were accomplished, the block diagram has been converted to P-IDE application, the executable file was loaded onto the target, recorded signals have been retrieved and compared to the signals obtained by SL model simulation. This comparison is presented in Fig. 8. Recorded referent signals in the figure are in full lines, simulation results are in dashed lines and signals from microcontroller program execution are in dotted lines. A perfect match of simulation and code execution signals can be observed, thus proving the correctness of SL to P-IDE conversion. Also it can be seen

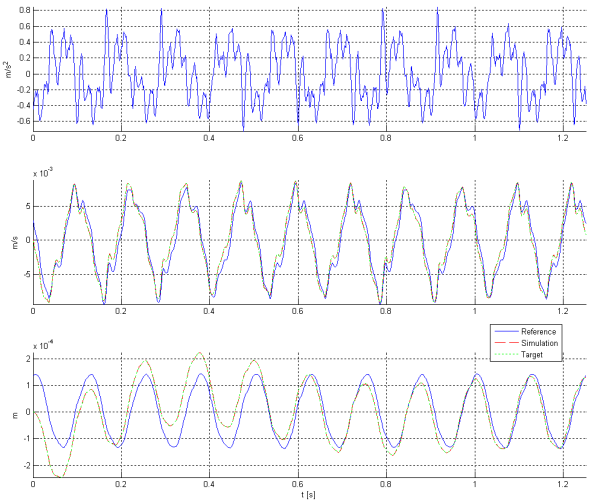

Fig. 8. Comparison of test signals generated by SL simulation and by μC program execution

that, after filter transients have passed, these two signal sets closely approach referent signals.

5 CONCLUSION AND FUTURE WORK

By using prototypical tools developed inside MATLAB/Simulink environment, it has been shown how legacy artifacts, in this case assembly macro routines and code generation procedure, can be successfully integrated into MBD toolchain. Sample SL model has been constructed using built-in and user-defined components and executable code has been generated from it. Signals resulting from simulation have been verified to be the same as signals produced by microcontroller code execution.

For an integral MBD environment, three major challenges remain. Firstly, efficient and coherent REQM system must support all stages of development. One approach is to integrate requirements into the model while the other is their separate handling. In the first case model transparency must be preserved, e.g. by defining views of the model, and in the second tracing of the requirements to the model, and vice versa, must be ensured.

For a productive MBD environment not only legacy artifacts are sufficient, it must be modular and extensible. A framework for integration of newly generated code, e.g. C code generated by *RealTime Workshop*, into existing software platform should be developed. This means that a mixture of legacy assembly code and automatically generated C code, an application produced by the new MBD environment, must successfully execute on the target running proprietary RTOS.

The importance of testing is unquestionable so test procedures must be defined for each stage of the development,

starting from testing of basic components, through testing of functional modules, application programs and finally testing of entire systems. For the back-to-back tests appropriate signal comparison algorithms should be implemented. As high as possible automation level in test construction and execution is desirable.

ACKNOWLEDGMENT

The authors would like to thank Marijan Bogut, head of the *Noise and vibration laboratory* at *Končar – Electrical Engineering Institute, Inc.* for his help in the experimental part of the presented work.

REFERENCES

[1] J. Zander-Nowicka, *Model-based Testing of Real-Time Embedded Systems in the Automotive Domain*. PhD thesis, Technischen Universität Berlin, 2009.

[2] B. Schätz, A. Pretschner, F. Huber, and J. Philipps, "Model-Based Development of Embedded Systems," tech. rep., Technische Universität München, 2002.

[3] A. Rau, *Model-Based Development of Embedded Automotive Control Systems*. PhD thesis, University of Tübingen, 2002.

[4] I. Stürmer, *Systematic Testing of Code Generation Tools A Test Suite-oriented Approach for Safeguarding Model-based Code Generation*. PhD thesis, Technischen Universität Berlin, 2006.

[5] S. Marijan, "Control electronics of TMK2200 type tramcar for the City of Zagreb," in *Proc. International Symposium on Industrial Electronics, ISIE 2005*, (Dubrovnik, Croatia), pp. 1617–1622, June 2005.

[6] S. Marijan, "Vehicle control unit for the light rail applications," in *Proc. 13th International Conference on Electrical Drives and Power Electronics, EDPE 2005*, (Dubrovnik, Croatia), September 2005.

[7] S. Marijan and I. Petrović, "Platform based development of embedded systems for traction and power engineering applications – experiences and challenges," in *Proc. 5th IEEE International Conference on Industrial Informatics, INDIN 2007*, (Vienna, Austria), July 2007.

[8] S. Marijan, *Sustainability of embedded control systems for rail vehicles and power generation units*. PhD thesis, University of Zagreb, 2011.

[9] J. Babić, S. Marijan, and I. Petrović, "The comparison of MATLAB/Simulink and proprietary code generator efficiency," in *Proceedings of the International Conference on Electrical Drives and Power Electronics, EDPE 2009*, (Dubrovnik, Croatia), pp. 12–14, October 2009.

[10] J.-L. Boulanger and V. Q. Đao, "Requirements engineering in a model-based methodology for embedded automotive software," in *IEEE International Conference on Research, Innovation and Vision for the Future, RIVF 2008.*, (Ho Chi Minh City, China), pp. 263–268, July 2008.

[11] S. Siegl, K.-S. Hielscher, and R. German, "Model Based Requirements Analysis and Testing of Automotive Systems with Timed Usage Models," in *18th IEEE International Requirements Engineering Conference*, (Sydney, Australia), pp. 345–350, September 2010.

[12] H. Dubois, M.-A. Peraldi-Frati, and F. Lakhal, "A model for requirements traceability in a heterogeneous model-based design process: Application to automotive embedded systems," in *15th IEEE International Conference on Engineering of Complex Computer Systems*, (Oxford , UK), pp. 233–242, March 2010.

[13] E. Geisberger, J. Grünbauer, and B. Schätz, "A Model-Based Approach To Requirements Analysis," in *Methods for Modelling Software Systems, MMOSS*, (Dagstuhl, Germany), pp. 1862–4405, August 2007.

[14] D. Wild, "AutoFOCUS 2: The Picture Book," tech. rep., Technische Universität München, 2006.

[15] A. Gambuzza and D. Koert, "A Concept for Improving the Reusability of Mechatronic System Models," in *In Proc. of the Workshop on Object-oriente Modeling of Embedded Real-Time Systems, OMER3*, (Paderborn, Germany), pp. 43–48, October 2005.

[16] A. J. Mellor and T. Ulber, "Executable and Translatable UML," in *3rd Workshop on Object-oriented Modeling of Embedded Real-Time Systems, OMER3*, (Paderborn, Germany), pp. 69–72, October 2005.

[17] G. Raghav, S. Gopalswamy, K. Radhakrishnan, J. Hugues, and J. Delange, "Model based code generation for distributed embedded systems," in *European Congress on Embedded Real-Time Software,ERTS 2010*, (Toulouse, France), pp. 1–9, May 2010.

[18] A. Agrawal, G. Karsai, and F. Shi, "A UML-based graph transformation approach for implementing domain-specific model transformations," *International Journal on Software and Systems Modeling*, pp. 1–19, 2003.

[19] M. Baleani, A. Ferrari, L. Mangeruca, A. L. Sangiovanni-Vincentelli, U. Freund, and H. J. W. E. Schlenker, "Correct-by-Construction Transformations across Design Environments for Model-Based Embedded Software Development," in *Proceedings of the conference on Design, Automation and Test in Europe, DATE05*, (Munich, Germany), pp. 1044–1049, March 2005.

[20] S. Sendall and W. Kozaczynski, "Model transformation: the heart and soul of model-driven software development," *IEEE Software*, vol. 20, no. 5, pp. 42–45, 2003.

[21] M. Broy, M. Feilkas, M. Herrmannsdoerfer, S. Merenda, and D. Ratiu, "Seamless Model-based Development: from Isolated Tools to Integrated Model Engineering Environments," *Proceedings of the IEEE, Special Issue on Aerospace and Automotive Software*, vol. 98, no. 4, pp. 526–545, 2010.

[22] M. A. Mäkinen, "Model Based Approach to Software Testing," Master's thesis, Helsinki University of Technology, 2007.

[23] M. Utting, A. Pretschner, and B. Legeard, "A taxonomy of model-based testing," tech. rep., University of Waikato, 2006.

[24] E. Bringmann and A. Kramer, "Model-Based Testing of Automotive Systems," in *1st International Conference on Software Testing, Verification, and Validation*, (Lillehammer, Norway), pp. 485–493, April 2008.

[25] K. Lamberg, M. Beine, M. Eschmann, R. Otterbach, M. Conrad, and I. Fey, "Model-based testing of embedded automotive software using MTest," in *SAE World Congress 2004*, (Detroit, USA), March 2004.

[26] M. Conrad, "A Systematic Approach to Testing Automotive Control Software," in *Proc. of Convergence 2004*, (Detroit, USA), October 2004.

[27] M. Conrad and I. Fey, "Systematic Model-Based Testing of Embedded Automotive Software," *Electronic Notes in Theoretical Computer Science*, vol. 111, pp. 13–26, 2005.

[28] M. Conrad, S. Sadeghipour, and H. W. Wiesbrock, "Automatic Evaluation of ECU Software Tests," in *SAE 2005 World Congress*, (Detroit, USA), April 2005.

[29] D. Keil, "Real-Time Data Exchange," tech. rep., Texas Instruments, 1998.

Josip Babić Josip Babić received B.Sc. degree in 2006 from the Faculty of Electrical Engineering, University of Zagreb. Since 2007 he has been with Section for Embedded Systems of Power Electronics and Control Department at Končar - Electrical Engineering Institute in Zagreb, where he is currently employed. He is responsible for research and development of embedded systems based on digital signal microcontrollers. His main research interests are: model based development in the field of embedded real-time systems.

Siniša Marijan Siniša Marijan was born in 1960. He received B.Sc. and Ph.D. degrees from the Faculty of Electrical Engineering, University of Zagreb. He is involved in the research and development of proprietary modular HW and SW platforms and strategic decisions related to the distributed real-time control systems. So far, he was responsible for research and development in the fields of embedded systems for power engineering and rail vehicles. This enabled the production of several significant and complex proprietary products: embedded control systems for digital voltage regulators, locomotive main control systems, tramcar vehicle control unit, main control units for diesel and electrical trains, wind turbine control systems, HW/SW of tramcar and train traction converters, train heating converters, multi-system converters for passenger coaches, auxiliary power supplies for locomotives, trains and trams.

Ivan Petrović received B.Sc. degree in 1983, M.Sc. degree in 1989 and Ph.D. degree in 1998, all in Electrical Engineering from the Faculty of Electrical Engineering and Computing (FER Zagreb), University of Zagreb, Croatia. He had been employed as an R&D engineer at the Institute of Electrical Engineering of the KonÄŤar Corporation in Zagreb from 1985 to 1994. Since 1994 he has been with FER Zagreb, where he is currently a full professor and the head of the Department of Control and Computer Engineering. He teaches a number of undergraduate and graduate courses in the field of control systems and mobile robotics. His research interests include various advanced control strategies and their applications to control of complex systems and mobile robots navigation. Results of his research effort have been implemented in several industrial products. He is a member of IEEE, IFAC – TC on Robotics and FIRA – Executive committee. He is a collaborating member of the Croatian Academy of Engineering.

AUTHORS' ADDRESSES

Josip Babić, B.Sc.
Sinša Marijan, Ph.D.
Končar - Electrical Engineering Institute, Inc.,
Fallerovo šetalište 22 , HR-10000, Zagreb, Croatia
email: jbabic@koncar-institut.hr, smar@koncar-institut.hr

Prof. Ivan Petrović, Ph.D.
Department of Control and Computer Engineering,
University of Zagreb,
Faculty of Electrical Engineering and Computing,
Unska 3, HR-10000, Zagreb, Croatia
email: ivan.petrovic@fer.hr

Received: 2011-05-24
Accepted: 2011-10-21