ISSN 1407-7493 COMPUTER SCIENCE DATORZINĀTNE 2009-7493

APPLIED COMPUTER SYSTEMS
LIETIŠĶĀS DATORSISTĒMAS

SEVERAL OUTLINES ON MODEL-DRIVEN APPROACH FOR TESTING OF EMBEDDED SYSTEMS

DAŽI MODEĻVADĀMĀS PIEEJAS PAMATPRINCIPI IEGULTO SISTĒMU TESTĒŠANAI

Jurijs Grigorjevs, Mg. sc. ing., PhD student, Riga Technical University, Meza 1/3, Riga-LV1048, Latvia,
Integration testing team leader, TietoEnator SIA, Lacplesa 41, Riga-LV1011, Latvia,
jurijs.grigorjevs@gmail.com
Oksana Nikiforova, asoc. professor, Dr.sc.ing., Riga Technical University, Meza 1/3, Riga, LV 1048, Latvia,
oksana.nikiforova@rtu.lv

Model-driven testing, embedded systems

1. Introduction

Embedded systems are specific because of multiple interfaces with external environment. The cooperation with external environment requires special hardware and software as well as approaches of software development to provide quality of an embedded system. Therefore embedded systems are characterized by the set of special features, which require special internal structure, internal processing principles and mechanisms. In OMG Model Driven Architecture (MDA) [1] models are the primary artefacts of the software developments process. According to the main statement of MDA system development is based on keeping the appropriate level of abstraction – that is, separate the overall system design from its implementation on the specific technology platform. Central to MDA is the notion of creating different models at different levels of abstraction and then linking them together to form an implementation [2]. Some of these models exist independent of software platforms, while others are specific to particular platforms. Each model is expressed using a combination of text and multiple complementary and interrelated diagrams. Existent researches in the context of MDA are mostly focused on system development from functional, structural and physical aspects point of view [3], [4]. In [3] authors deal with structure and functional behavior of embedded systems in context of applying MDA principles and provide the approach of model transformation for such models. Business requirement traceability through different MDA models is discussed in [4]. In [5] in the context of MDA authors describe model-based testing without testing model derivation from MDA models. In our research we focus on testing model derivation from system MDA models.
The one of the main ideas of this paper is to show general model-driven testing principles and their realization in the context of MDA. The second object of research in this paper is an effort to apply main principles of model-driven development and foundations of model transformations to the testing of several features of embedded systems, which requires a specific approach for verification and validation during such a system development.
To create projection of theory to practical context, in this paper we have chosen one embedded systems feature – timing. For timing specification on system level, we took OMG suggested UML profile for Schedulability, Performance and Time. For test case specification we have used also OMG provided UML standardized Testing profile. Using these modeling approaches, we presented simplified models that support mentioned timing feature specification.

2. Principles of model-driven approach in the context of system testing

Generally model-driven testing defines test case development strategy using the model of the system under test [6]. This means that using abstract functional model of the software, which represents also

functional as well as non-functional requirements of the system, sets of test data, preconditions and test exit criteria are created. There are two different ways to implement test cases: manual and automated. In manual test case generation there should be at least one testing engineer, who analyzes given abstract functional model of the system and creates test cases. Test cases could be written in native language as well as using some formal definition. Such test case generation technique is widely used and can be combined with a non-model-driven testing, for example based on requirements document.

Automated test case generation becomes more and more popular. There are 2 major reasons for that. The first reason is to cut time-to-market and to deliver products faster with higher quality [7]. The popular idea to start testing earlier is going to be realized in model-driven testing. Analysis phase produces models of the system, describing its functional and technical aspects and including previously prepared requirements (sometimes requirements could be detailed also in analysis phase together with construction of models). This means immediate starting of the testing process without waiting of programming activities. The second major reason to use model-driven testing is compliance to principles of MDA, where models (with some specific formal OCL restrictions) are the only possible system specification documents. In this case model transformation and generalization is compliant to classical testing V-model [6], where the main idea of the model is not to show appropriate related activities, but to represent program abstraction level (for example, to show that system testing deals with the whole fully integrated system and user acceptance testing verifies and validates end-user requirements). Figure 1 presents the accordance between model transformations under MDA and V-model of testing.

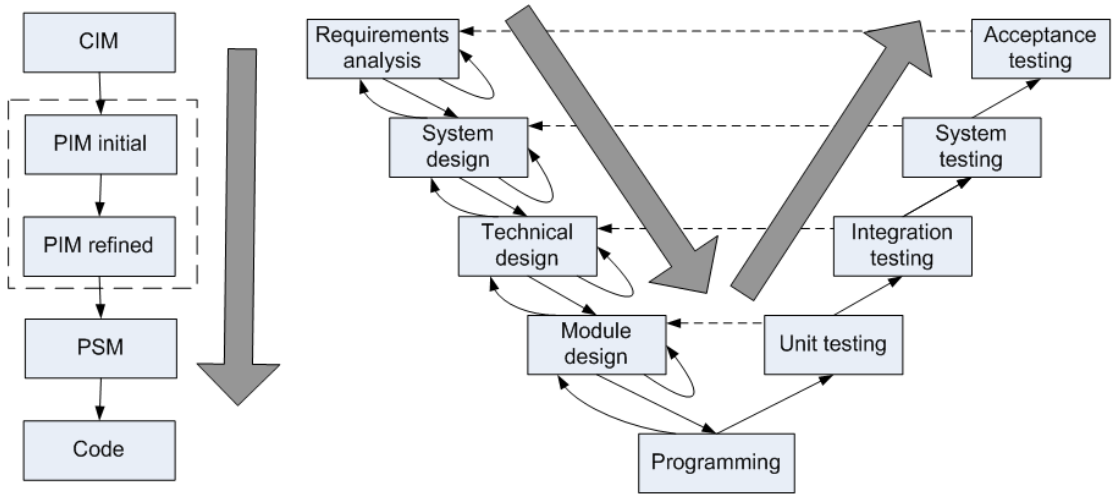

Fig.1. MDA model transformation and testing V-model

MDA principles specify models on different abstraction levels. The model with highest abstraction level is CIM (Computation independent model), which represents requirements for the system to be build. Such model sometimes is call a domain or a business model. In MDA specification CIM requirements should be traceable to the PIM and PSM constructs that implement them and vice versa [2]. On Figure 1. in testing V-model requirements analysis phase is the appropriate system abstraction to CIM model. The result of requirements analysis is the business and functional aspects of developed system. From the testing point of view such abstraction level is covered in acceptance testing, where the required system functionality is validated. The next abstraction model is Platform independent model (PIM). The model describes the system and its structure, but does not show details of its use. PIM does not show any platform specific features and is used to specify general systems functional principles. On testing V-model this abstraction level is presented by system design and system testing activities, where similar aspects of the system are described. The Platform specific model (PSM) is produced by the transformation from PIM model. If PSM model contains all necessary information needed for system implementation, then the model could be used for program code generation. In classical program development this correspond to concrete module design, programming and unit testing.

2.1. Modeling artefacts in context of testing

Models are the primary artefacts during model-driven development. Model consists of sets of elements that describe some physical, abstract, or hypothetical reality. UML is the central component of MDA and is used for presentation of system model at different levels of abstraction. It can be appended with formal notations written in OCL. A metamodel is a model of a modeling language. It defines the structure, semantics, and constraints for a family of models. A model is captured by a particular metamodel. For example, a model that employs UML diagrams is captured by the UML metamodel, which describes how UML models can be structured, the elements they can contain, and the properties those elements exhibit. In turn, a metamodel may describe some properties of a particular platform, not only the UML, while a platform's properties may be described by more than one metamodel.
By the nature, test case also can be represented as a model, because it also consists of a set of elements that describe abstract reality, which is set of preconditions and data inputs for some program functionality as well as set of expected results for described abstraction. In model transformation test case generation means some set of transformation rules necessary to get test case destination model from systems model, presented in the form appropriate for test case generation.
A projection of MDA definitions of model, metamodel and model transformations into process of system testing gives us the possibility to define test case generation process in terms of model transformations under main statements of model-driven approach. Before model specification, appropriate metamodel should be defined. Metamodels specify all possible elements and relations in between of target models. Metamodels are based on Meta Object Facilities (MOF). Intermodels dependencies for test case generation presented in the terms of model driven approach are shown on Figure 2.

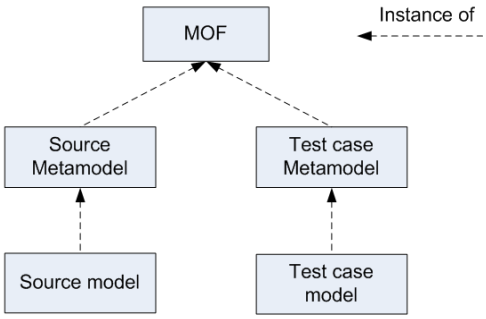

Fig.2. Application of MDA principles for test case presentation

Figure 2 presents dependencies between source and Test case models of the current research. The source model is based on the timing concepts defined in UML profile for Schedulability, Performance and Time and Test case model is based on the concepts taken from UML Testing profile. The Testing profile provides also standalone metamodel describing all necessary artefacts mentioned in the profile. The first profile, sometimes called also Real-time profile, does not contain standalone metamodel for model definition, but includes additions to standard UML metamodel.
MDA principles are based on the model transformation, where new models are created from existent models applying special transformation. Transformation between models is made by transformation definition, which is a collection of transformation rules in the form of an unambiguous specifications of the way that (a part of) one model can be used to create (a part of) another model [8]. Transformation rules are presented with predefined transformation definition language. Automated transformation of models can be provided only by special tool, which is able to understand a notation of a source and target models and to apply transformation rules according to their logic to given source model. The result of such process is a set of destination models. Basic MDA framework described in [8] defines a general scheme for transformations between models. It is applied to a test case generation and the result is shown on Figure 3. Same principles are applied in test case model generation, where several test case models are created from one or more system models. The Figure 3 shows model transformation principles, for transformation needed artefacts and relations between them. In this

article we do not discuss transformation in more detailed view and above mentioned diagram is presented here to show test case generation principles in the context of MDA.

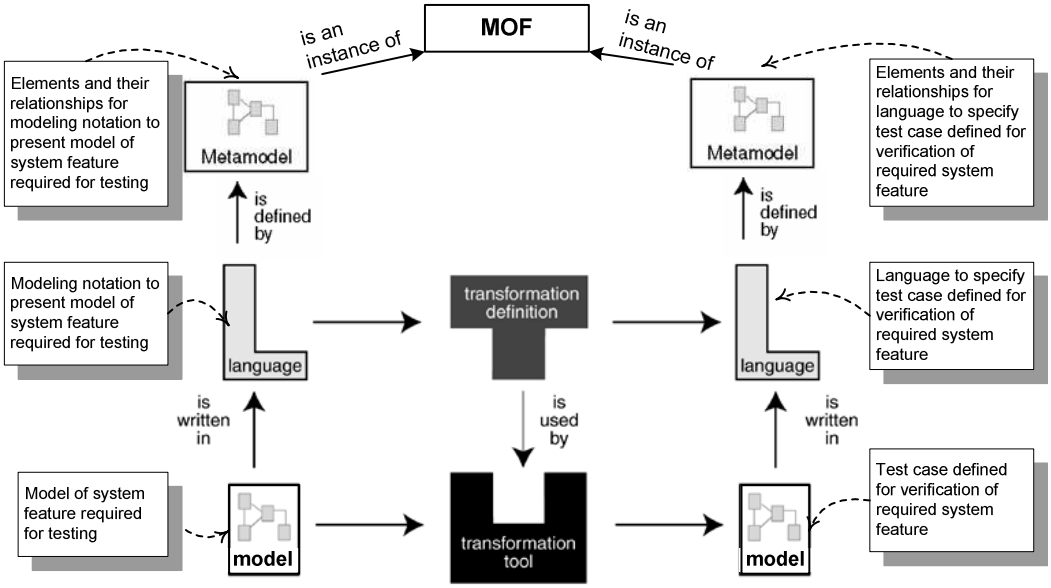

Fig.3. Application of basic MDA framework [8] for test case generation

In the case of test case generation we need a notation for modeling of features of a system and a notation for description of such a test case.

2.2. UML Testing profile

For testing purpose OMG has created and standardized UML Testing profile, which is used for test case model development. Profile provides standardized approach for test cases and test suites development in model-driven development. It represents MOF-based standalone metamodel for testing aspects modeling. Figure 4 presents MOF-based metamodel of testing profile [9].

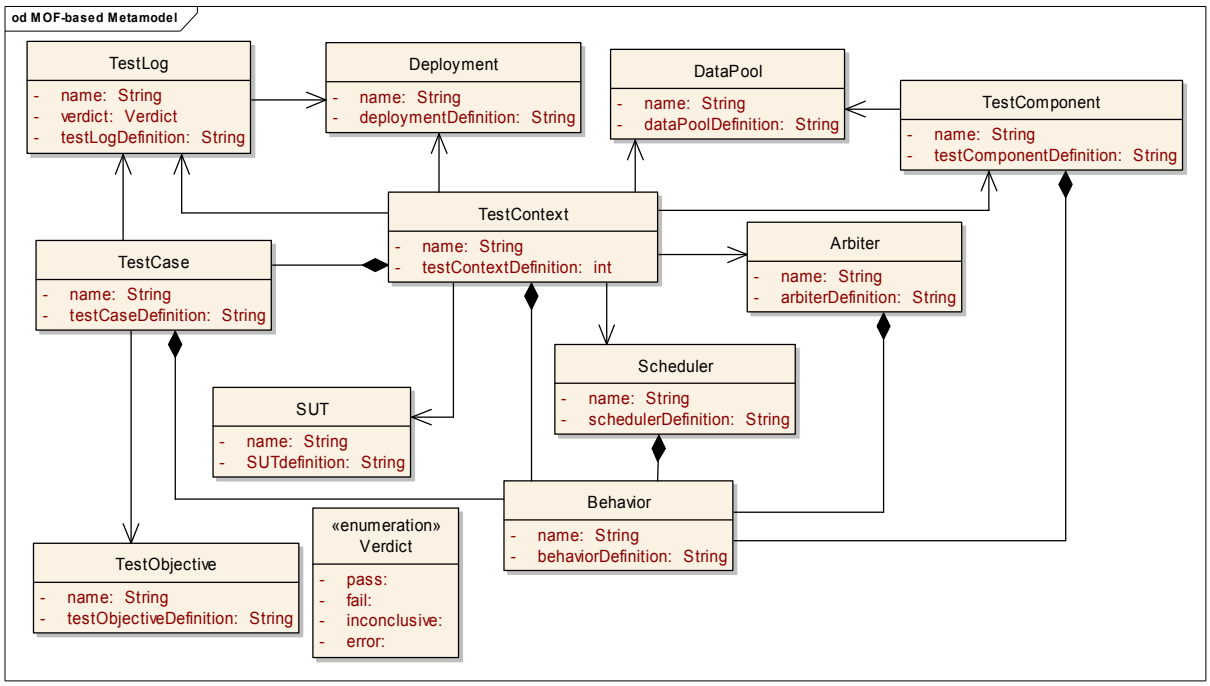

Fig.4. Testing profile metamodel

This profile suggests completely different and new approach for testing management, which consists of test case, test data, program's behavior and test results. Using Testing profile test cases and to them related data could be stored and represented using in profile defined elements of metamodel.
The profile describes major classes of testing related aspects and represents general approach of test case management. As it shown on Figure 4, standardized Testing profile represents general view on testing aspects and specifies high abstraction level of test case management structure. Profile does not specify any system related aspects and leaves this for concrete systems developers. Such approach provides profile to be widely used for different types of systems including embedded systems too. Standard UML 2.0 version is supposed to be used in general system specification. For specific systems such as embedded systems, there exists special UML profile called UML Profile for Schedulability, Performance and Time [10]. This profile provides additional modeling elements for embedded systems specification.

3. Embedded systems

Embedded systems are systems where hardware and software are the part of complex system and are scheduled for functioning of specific usage without interference of humans. The term embedded system as a research object examined in this article means software for management of such systems. To start the analysis and research of the specific of testing process of embedded systems, it is necessary to analyze the specific character of embedded systems and to define properties and restrictions of such systems. Figure 5 describes structure and features of embedded systems [11].

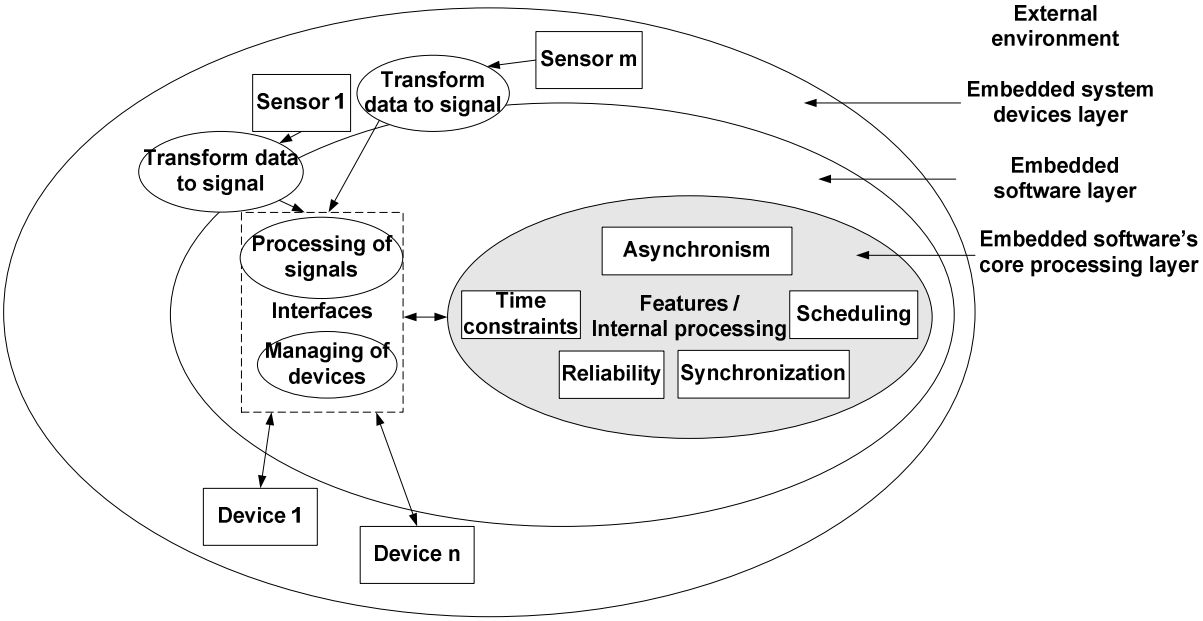

Fig.5. Structure of embedded systems

Embedded systems should interact with external environment to get its parameters and measures, e.g. parking sensors in parking alert systems in cars, thermometers in refrigerators, weight sensors in elevators and others. Such devices are represented in the layer of embedded system devices in Figure 5. After receiving external signals, devices transmit data to appropriate interface for further converting and processing in the computer (is represented as the layer of embedded software). Core processing layer of embedded system (highlighted as grey) represents software, which is focused on support of logical functionality of whole system. To establish valuable functionality of embedded systems, core processing layer should support in Figure 5 mentioned features. These five specific features of embedded systems are selected for detailed examination in this research in the context of testing of embedded systems [11][12][13].

3.1. Specifics of Embedded systems

To fulfill managing of devices without human interaction and other specific activities embedded software should support next non-functional requirements [12][13]:

- Asynchronism is a property of the systems, where events and actions can occur independently in time and order. While embedded systems require full time tolerance. Both facts require specific processing and control of asynchronous events and actions to meet strict time constraints and fulfill appropriate processing.
- Synchronization is mandatory non-functional requirement in asynchronous systems with multiple devices. There are 2 types of synchronization: data synchronization and synchronization of processes. Process synchronization refers to the coordination of simultaneous threads or processes to complete a task in order to get correct runtime order and avoid unexpected race conditions. Data synchronization is the process of keeping multiple copies of the set of data coherent with one another.
- Scheduling is a process of setting an order and time for planned tasks. In systems with multiple processes there are competitive processes therefore it is necessary simultaneously handle several tasks according to developed strategy. The order of planning the running of tasks ensures two possibilities: management of resource consumption and foreseeing of worse cases when defined planning algorithm is used. There are one processor scheduling and multiprocessing (using more then one CPU).
- Time constraints are the modeling elements to define time limits for some activity or activity set. The concept of real-time system supposes that preciseness in time is more important than preciseness of software function. Time restrictions set usefulness of program activities dependence on time. In addition to deadline there are another time restrictions, for example, according to run time of some activity a special function determinates usefulness of it.
- Reliability is an ability of a system or component to perform its required functions under stated conditions for a specified period of time. Embedded software needs to be fault tolerant and during the compilation it is necessary to intercept defined types of software or hardware failures (for example some optional device is broken and it is necessary to continue to perform main operations or in case of incorrect data received, system should handle this inconsistence and continue adequate processing).

Above mentioned requirements are typical for the most of embedded systems and often could be required in other processing systems. Therefore could be also used in modeling and verification of different non-embedded systems with similar features. Mentioned specifics of embedded systems require special representation of metamodel level. Timing conditions as the example of embedded system features are selected for detailed discussion in next chapters.

3.2. Timing conditions in models

Embedded systems in general are real-time systems and inherit all timing aspects. Almost every popular modeling notation has it own on time focused adjusted notations. For example, Petri net has Timed Petri net and PRES+ models, there are Timed Data flow diagram, Timed State transition diagram and others [14].

For real-time systems OMG has created special profile called UML profile for Schedulability, Performance and Time. Although profile does not contain standalone metamodel for concepts specification, it defines set of additions for UML standard metamodel necessary for in profile defined concepts specification. Profile is separated into several domains: Time, Concurrency, Schedulability and Performance domains. Each domain defines set of concepts focused on embedded system feature definition. Figure 6 presents all timing concepts defined within Time domain and relations in-between.

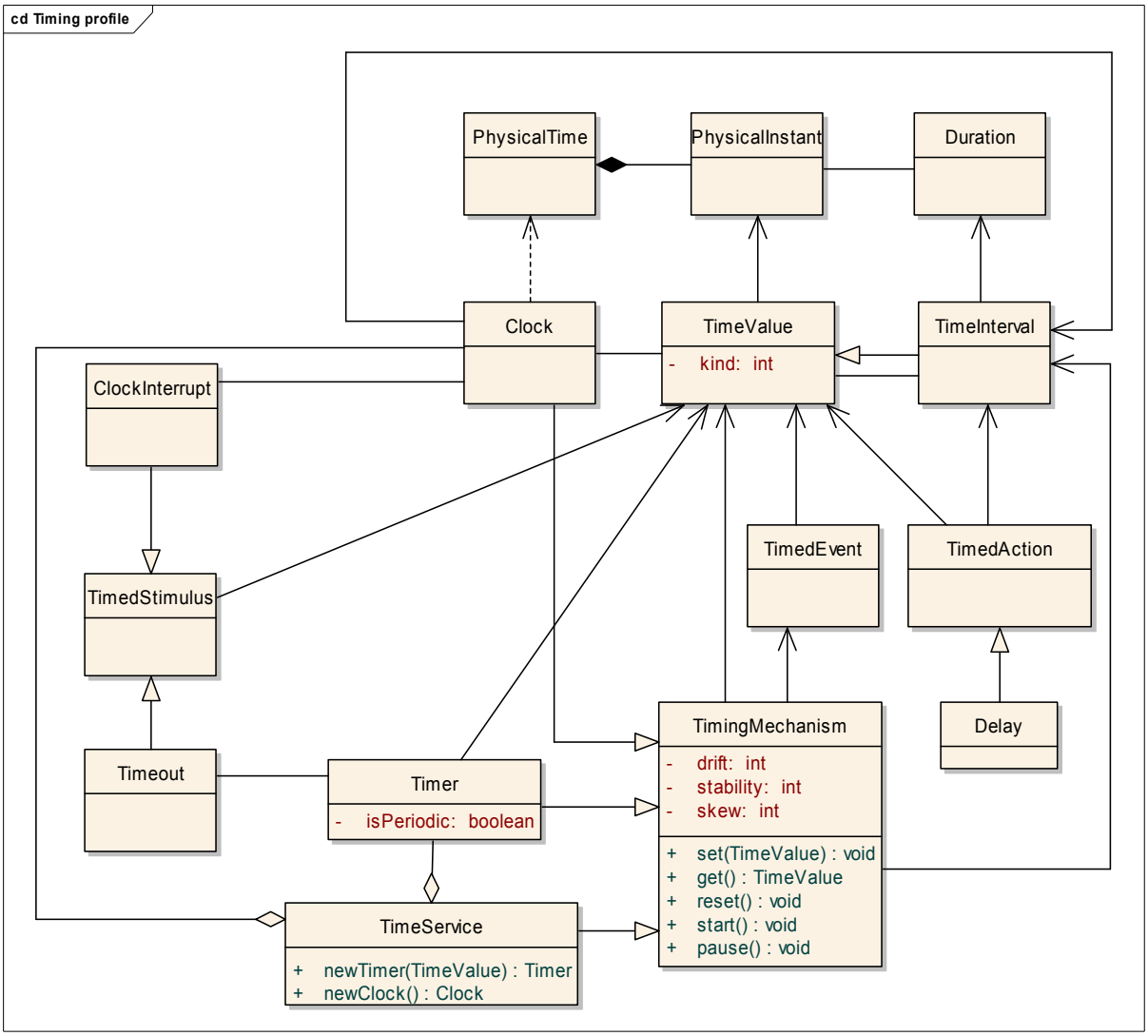

Fig.6. Time domain in UML profile for Schedulability, Performance and Time

All defined concepts of Time domain are grouped into 4 packages: TimedEvents, TimeModel, TimingServices and TimingMachanisms. Each package covers their specific needs in timing constraint specification.
OMG assumes that time progresses monotonically and only in forward direction. For this purpose they define such classes as: PhysicalTime, PhysicalInstant, Duration, TimeValue, TimeInterval, Clock, Timer, ClockInterrupt, TimedEvent, TimeOut and so on. For detailed computers timing mechanism specification profile includes specification of clock interrupts, timers, time events and others timing related artefacts. All of this provides possibility to model timing aspects during systems specification and further processing in model transformation and test case generation.

4. Practical outline of model construction

As the examples of model construction we discuss 2 program interactions with timing restrictions. For this example 2 simple models, representing system behavior and structure of test cases, are shown below. Taking into account model-driven testing approaches and MDA principles, the first step is to define modeling notation (modeling language) for models representation. For timing concepts specification in system model we use UML2.0 notation with profile for Schedulability, Performance and Time. This provides all necessary modeling elements for timing specification. In the same time for

test case specification Testing profile is applied. Given model examples are built for the purpose to show simple timing concepts realization in system model and interpretation in test case model. Automated model transformation is not discussed here and is out of scope of this paper. Figure 7 presents such Sender-Receiver example model appended with RT timing notations.

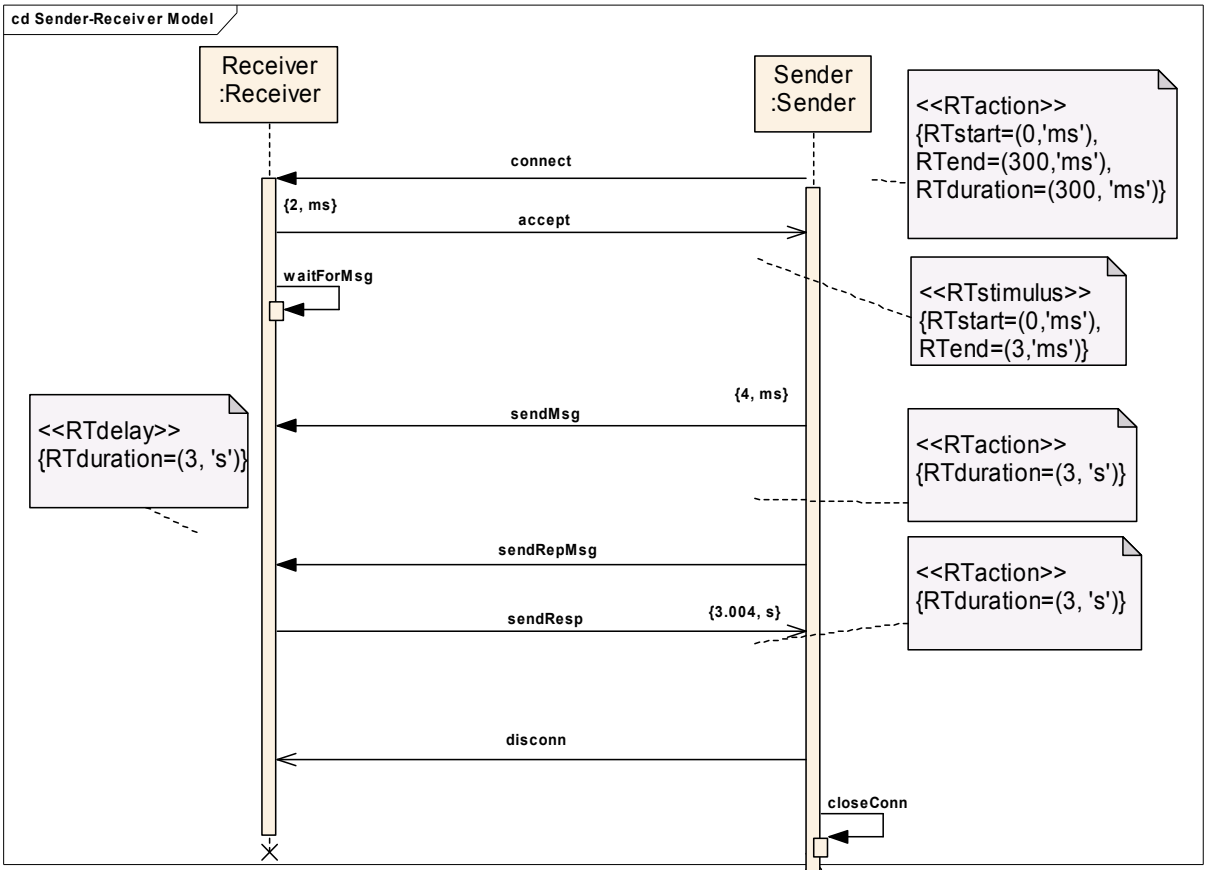

Fig.7. Sender-Receiver model with timing notations

Standardized UML profile approach provides unified time aspects specification in system modeling. Using stereotypes RTstimulus, RTaction and RTdelay (other possible stereotypes weren't used in this example) and to them related tags and tag values, we have specified actions and events processing time and time restrictions.

To provide testing of timing aspects of system behavior, it is needed to specify them on test case level. UML Testing profile is quite general and does not provide structures for collection of timing aspects. Using metamodel defined in mentioned profile, we have created class diagram for testing artefacts presentation to support timing aspects. This model is simplified example to show the principles of test model development in the context of necessary task or program. Figure 8 shows test case model for previously presented Sender-Receiver model.

The main classes of the model are taken from Testing profile and appended with necessary elements. Such approach provides common standardized testing model development for different technology systems. Additional classes *ParamsType* and *Params* are necessary to make the model more unified and to provide program parameterization using different types of input data. For timing specification main artefacts are classes *RunPlan*, *RunScheduler* and *Programm*. Class *RunPlan* provides logical structure of timing restrictions described in *RunScheduler* for appropriate *Programm*. Class Checker is necessary to validate actual functional result as well as timing aspects according to planned result and to decide about the test case successful or unsuccessful processing.

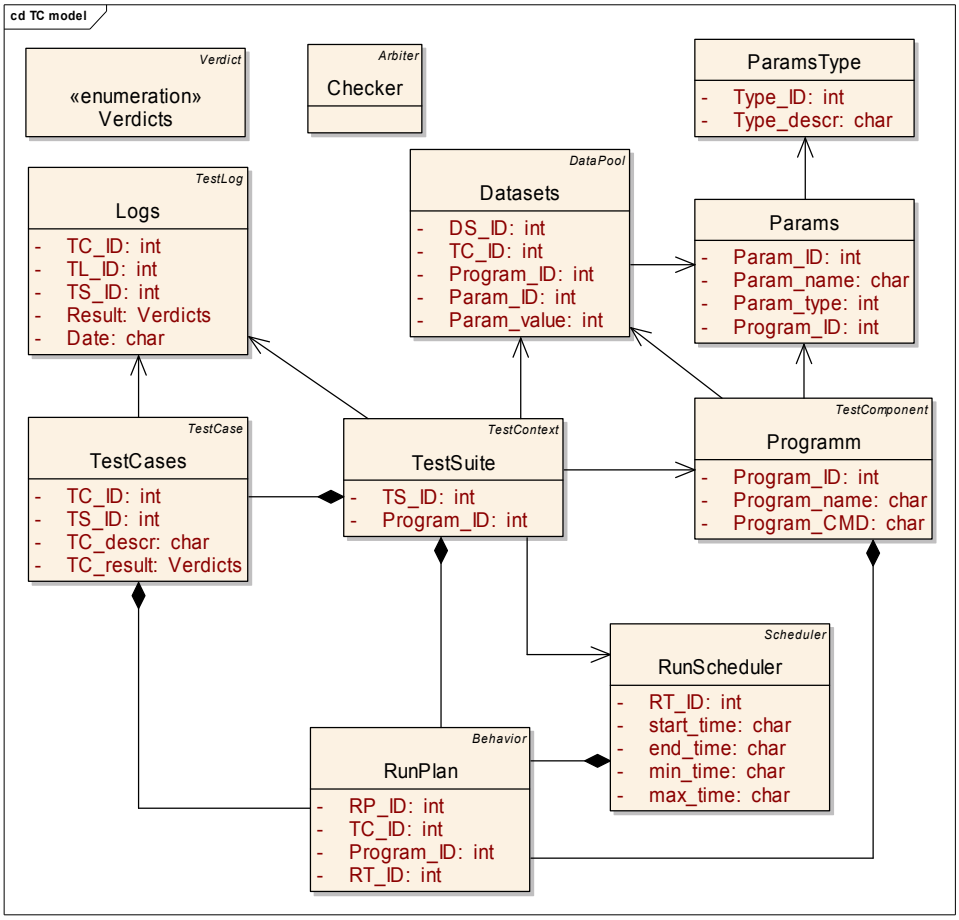

Fig.8. Test case model

This model is able to capture simple timing restrictions associated to some program processing. For more detailed timing aspects definition this model should be appended with necessary elements, but the core structure of testing model stay unchanged.

5. Conclusion

Model-driven testing is an integral part of MDA, although it is also widely used in other software development methodologies. The main idea of the approach is to use formally described model-based specification as for test case generation. In general model-driven testing test case are derived using manual methods of test cases implementation. MDA makes this more efficient and supposes test case automated generation by applying set transformation rules. For such transformation test case model includes all necessary testing artefacts, which are mainly defined within UML Testing profile. This profile provides standardized structure of testing elements and is common enough to be used for different technology systems as well as for embedded systems.

Embedded systems are specific because of multiple interfaces with external environment. The cooperation with external environment requires special hardware and software as well as approaches of software development to provide quality of an embedded system. Therefore embedded systems are characterized by the set of special features, which require special internal structure, internal processing principles and mechanisms. To cover such specific processing OMG has created special UML profile for Schedulability, Performance and Time, which is focused especially on real-time systems.

To provide practical system and testing model construction we have chosen one embedded systems feature and created appropriate systems and testing models for timing representation. For systems modeling we used Time domain defined in mentioned UML profile for Schedulability, Performance

and Time, which includes all necessary timing artefacts for this feature representation. Using UML Testing profile metamodel we created class diagram for testing management that is focused especially on timing aspects for system behavior and provides timing restriction representation on test case level. Such approach is the first mandatory step for model-driven testing in the context of MDA. In next steps these models are going to be used in model transformation to provide automated test case derivation from system model.

The research reflected in the paper is supported by Grant of Latvian Council of Science No. 09.1245 "Methods, models and tools for developing and governance of agile information systems".

References

1. OMG Model Driven Architecture [Electronic resource] / OMG, 2008. - http://www.omg.org/mda/ - accessed 12.10.2008.
2. MDA Distilled: Principles of Model-Driven Architecture / Mellor S.J., Scott K., Uhl A., Weise D. – Addison Wesley Professional, 2004. – P.176.
3. Metamodels and MDA Transformations for Embedded Systems [Electronic resource] / Research Laboratory of the University of Sciences and Technologies of Lille, 2004. - http://www2.lifl.fr/west/publi/BoDuDe04.pdf– last accessed 22.03.2009
4. Towards a traceability model in a MARTE-based methodology for real-time embedded systems [Electronic resource] / SpringerLink 2008. www.springerlink.com/content/5053442378721908, accessed 22.03.2009
5. Model Driven Testing of Real-Time Embedded Systems [Electronic resource] / Fraunhofer, 2006. - fokus.fraunhofer.de/de/motion/ueber_motion/unser_team/zander-nowicka_justyna/ModelDrivenTestingOfRealTimeEmbedddedSystems.pdf – accessed 22.03.2009.
6. Spillner A., Linz T., Schaefer H. Software Testing Foundations. – Santa Barbara: Rocky Nook Inc., 2007. – P.272
7. Towards Model-Driven Unit Testing [Electronic resource] / CiteSeer, 2006. - http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.62.200 – accessed 12.10.2008.
8. Kleppe A.G., Warmer J., Bast W. MDA Explained: The Model Driven Architecture: Practice and Promise. – Addison Wesley Professional, 2003. – P.170.
9. UML Testing Profile, V 1.0 [Electronic resource] / OMG, 2008. - http://www.omg.org/technology/documents/formal/test_profile.htm - last accessed 12.10.2008.
10. UML Profile for Schedulability, Performance, and Time, V 1.1 [Electronic resource] / OMG, 2008. –http://www.omg.org/technology/documents/formal/schedulability.htm – accessed 12.10.2008.
11. Grigorjevs J., Nikiforova O. Features of embedded systems that require specific testing approaches // Scientific Proceedings of Riga Technical University, 5, Computer Science. Applied Computer Systems – Riga: RTU, 2007. – P. 229-241.
12. Grigorjevs J., Nikiforova O. Modelling of Non-Functional Requirements of Embedded Systems // Scientific Proceedings of 42nd Spring International Conference MOSIS2008 – Ostrava: MARQ, 2008. – P. 13-20.
13. Grigorjevs J., Nikiforova O. Testing of Embedded System's Non-functional Requirements // Scientific Proceedings of the 8th International Baltic Conference Baltic DB&IS 2008 – Tallinn: Tallinn University of Technology Press, 2008. – P. 411-414.
14. Grigorjevs J., Nikiforova O. Complience of Popular Modelling Notations to Non-functional Requirements of Embedded Systems // Proceedings of the International Scientific Conference Informatics in the Scientific Knowledge 2008 – Varna: University publishing house VFU "Chernorizets Hrabar", 2008. – P. 139-149.

Grigorjevs J., Ņikiforova O. Daži modeļvadāmās pieejas pamatprincipi iegulto sistēmu testēšanai

Šīs raksts ir veltīts modeļvadītas testēšanas pamatprincipiem kontekstā ar iegulto sistēmu testēšanu. Rakstā ir sniegts iegulto sistēmu īss apraksts ar iepriekš izdalītām sistēmu īpatnības, kas prasa speciālās pieejas to testēšanai. Tas ir: uzdevumu plānošana, laika ierobežojumi, uzdevumu sinhronizācija, asinhronā darbība un drošums. Laika ierobežojumu īpatnība ir izvēlēta, lai demonstrētu īpašo pieeju šīs īpatnības specificēšanai un

tās ietekmi uz testēšanas modeli. Sistēmas funkcionēšanas modelī laicīguma definēšanai tika paņemts UML profils uzdevumu plānošanai, ātrdarbībai un laikam, kas piedāvā speciālas pakotnes laicīguma definēšanai. Saskaņa ar MDA pamatprincipiem, rakstā ir prezentēta standartizēta OMG pieeja testēšanas modeļa definēšanai – UML testēšanas profils, kas piedāvā neatkarīgo metamodeli testēšanas modeļa konstruēšanai. Raksta galvenā ideja ir parādīt modeļvadītas testēšanas pieejas pamatprincipus un sniegt vienkāršotus modeļus iegulto sistēmu laicīguma īpatnības modelēšanai. Rezultātā ir piedāvāti sistēmas funkcionēšanas modelis, kas balstās uz UML profili uzdevumu plānošanai, ātrdarbībai un laikam, un testēšanas modelis balstīts uz UML testēšanas profili. Abi modeli nodrošina vienkāršotus modeļus laicīguma specificēšanai. Turpmākais darbs modeļvadītas testēšanas virzienā ir transformācijas definēšana starp izveidotiem modeļiem ar mērķi nodrošināt automātisko testēšanas gadījumu ģenerēšanu no sistēmas funkcionēšanas modeļa.

Grigorjevs J., Nikiforova O. Several outlines on model-driven approach for testing of embedded systems
This paper is devoted to model-driven testing approaches in the context of embedded systems. The article discusses specifics of the embedded systems as well as specific testing approaches for them. As the testing objects of embedded systems next non-functional requirements were previously selected: task scheduling, time restrictions, synchronization, asynchronisms and reliability. Timing restrictions are selected for detailed analysis in model specification. For this purpose Time domain from UML profile for Schedulability, Performance and Time is presented and discussed. Testing model as destination model is specified using standardized UML Testing profile, which provides general purpose metamodel for such model definition. The main idea of the paper is to show general principles of the model-driven testing and to represent simplified example of testing of the one specific feature of the embedded system. Discussed model-driven testing approach is based on the model transformation, where the source model describes the feature of the system and the destination model is the test case model. Transformation rules in future works will provide test case generation using model of the system.

Григорьев Ю., Никифорова О. Основные принципы тестирования вспоенных систем, основанного на моделях системы
В статье уделено внимание основным принципам тестирования, основаного на моделях в контексте встроенных систем. Авторы предоставляют краткое описание системных свойств встроенных систем, требующие специального подхода в тестировании. Такими являются: планирование задач, временные ограничения, синхронизация задач, асинхронные операции и надёжность. Свойство временных ограничений выбрано для демонстрации особого подхода к описанию этого свойства в функциональной модели системы и его влияния на модель тестирования. Представленная функциональная модель системы основана на UML профиле для систем реального времени, а модель тестов на стандартизированном UML профиле тестирования. Основной целью данной статьи является описание основных принципов процесса тестирования, основанного на моделях системы, а так же демонстрация упрощённых моделей для отображения временных свойств встроенных систем. В статье предоставлены функциональная модель системы, основанная на UML профиле для систем реального времени, и тестовая модель, основанная на стандартизированном UML профиле тестов. Обе модели обеспечивают облегчённую спецификацию временных ограничений системы. Последующими шагами данного исследования является создание законов трансформации между созданными моделями с целью обеспечить автоматическую генерацию тестовых случаев на основе функциональной модели системы.