MDA APPROACH IN EMBEDDED SYSTEMS WITH STRICT REAL-TIME RESPONSE AND ON-THE-FLY MODELLING REQUIREMENTS

Otto ŽELEZNÍK, Zdeněk HAVLICE
Department of Computers and Informatics, Faculty of Electrical Engineering and Informatics,
Technical University of Košice, Letná 9, 042 00 Košice, tel. 055/602 4220, e-mail: o.zeleznik@gmail.com, zdenek.havlice@tuke.sk

ABSTRACT

The paper presents a proposal of a method for MDA based "on-the-fly" modelling of Musical Real Time Applications. The method is based on use of the Executable and Translatable UML approach (xtUML) with regards to implementation on embedded systems. Since the goal is to implement a real-time application requiring certain real-time Quality of Service (QoS) properties, the real-time response requirement influence on modelling is discussed as well. To provide a solution for the main user requirement, which is a possibility for an application model modification in real-time (during execution), we discuss a method, which splits the application into two parts. The first part is the implementation environment – the programmer/provider system, which will provide for the runtime environment of the second part, the user model itself, modelled using xtUML on-the-fly. The advantage of such separation results in an increased modelling easiness and reasonable real-time performance on user modelling as well as low signal processing latency.

Keywords: *MDA, software architecture modelling, embedded systems, real-time response, information system, xtUML*

1. INTRODUCTION

The modern era brings among us an evolution of our lives in every aspect. This is particularly true in the area of applying new methods in software architectures design especially focused on embedded systems used for sound and signal processing, broadcast applications, musical production and musical real-time applications. The last mentioned applications can be considered as specific software-hardware "musical instruments" as they provide the user/artist for creating or altering the musical piece in real time – while the artist is performing his show. We can consider this application kind as a special embedded information system since it processes information in form of sound and its properties.

The Musical Real-Time Applications usually consists of a combination of a specific signal processing software, which is implemented on a viable hardware platform whereas each part has specific properties and requirements (QoS, good real-time response). Since the fundamental purpose of these applications is sound effects execution based on a command query from the user, the implementation platform is usually based on a well designed embedded system. A powerful Digital Signal Processor (DSP) is usually a core of such system cooperating with various peripherals, sensors, actuators and converters. The converters perform environment signal conversion (input and output sound in our case) into the digital domain and vice versa, the mentioned DSP processor then realises a signal alteration following the user commands and settings using various signal processing methods [1]. Sensors and actuators on the other hand gather and scatter the controlling user input commands and output feedback providing user with various status information necessary to generate a correct input commands within the application. The figure 1 depicts an example overall architecture of such Musical Real-Time Application defined in the above paragraph.

Mentioned application kind covers for various embedded systems devices, such as sound effectors with real-time control [2], musical samples generators, variable digital filters, etc.

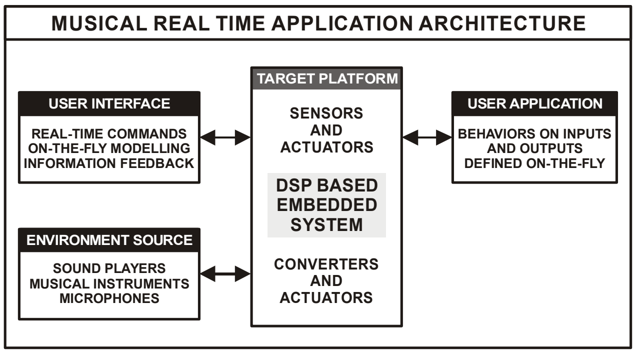

Fig. 1 Typical architecture of a Musical Real-Time Application

2. STANDARD MODELLING APPROACH AND REQUIREMENTS

Since the user – artist expects to achieve the best possible performance, when using such tools, there are two major and most important properties of these applications. One of them is a very good real-time response on user commands including as low sound processing latency as possible, the other one is the ability to configure the sound alteration scheme (algorithms) as freely as possible, e.g. the ability to model what the system has to do with the sound in real time.

2.1. Real-time response requirement influence on modelling

Today's embedded applications are designed exclusively in digital domain using modern DSP processors. The critical requirements in Musical Real Time Applications are a minimal input-output signal latency and very good overall real-time response of the system. The first specification exists due to the nature of human hearing and sound perception [3], the second is a result of necessity to process signals with various feedback control signals e.g. user interface input. All of these requirements strongly affect choice of software architecture in the end. Recent research in the domain of embedded systems has demonstrated rather strong link between hardware and software architecture. The application modelling therefore must be measured from various viewpoints considering the resources of the target hardware architecture such as computational power of the core processor, various specific properties of memory types, memory sizes, communication channels between peripherals, language types and evaluation of mixed or hand-optimised programming need for critical parts of the system (OS scheduler, peripheral drivers, etc).

When considering for example a common PC platform where properties like memory size and/or CPU power are insignificant, these can accommodate very extensive and even complex software architectures and very large data structures with nearly no limitation however with a trade-off for response performance. Embedded systems on the other hand are mostly designed on smaller processors and less powerful hardware platforms where specific architectural factors are due. One of these factors is small available memory and specific memory model within existing CPU. Also memory model must strictly be questioned before the design is due since embedded hardware architectures [4] usually support only two access memory types of unequally partitioned sizes:

- Fast Layer 1 memories for execution critical code and data storage (available only in several kBytes),
- Slower Layer 2 memories for execution non-critical code and data storage (depending on CPU available up to max. few Mbytes).

There is also a cache support, but with limited performance, especially when executing critical code. Layer 1 memories are usually 3-10times faster than Layer 2 memories and full CPU performance is obtained only when executing code from Layer 1 memories. Adhering to the above facts, programmer or software architecture designed must carefully evaluate which data structure will be stored and operated from Layer 1 memory and which will be stored and operated from Layer 2 memory. The same applies for code separation as well. Available memory size also affects the way, how data and information is handled in the embedded system. Proper algorithm design helps reducing size of temporary data structures used for data processing. Using rather one common variable/buffer for data storage and processing in all processing algorithms is one way how to use memory properly. Moving less critical data buffers into Layer 2 memory and more critical data buffers into Layer 1 memory improves performance as well.

Choice or rather a necessity of using a certain programming language in embedded systems defines another group of constrains in software architecture selection possibilities. Real-time embedded systems require very optimised and dense code since it may enhance execution performance. This is achievable usually by designing assembly written routines and so avoiding use of any higher-level compiler available for embedded systems design. Programming and design experience indicates that even the best-of-the-class compilers are unable to achieve performance of well-optimised hand-written assembly code. Gain in performance is about 5 to 25 per cent in favour of hand-optimised code. It is up to programmer to decide carefully, which part of application requires such a high performance (routines running most of execution time) and which part of application can be designed using higher level programming languages like C and running least of execution time but with high software architecture complexity. Real-time embedded systems usually work fine with assembly language used for audio processing routines. These usually require very basic data structure types (circular buffers, simple variables) use with nearly no abstract models and only Layer 1 fast memory partitioning. On the other hand C language is mostly used for control algorithms, which are of higher software complexity. Data structures become also more complex with use of dynamic memory allocation and management. They are however strictly located in Layer 2 memory region only. This complexity gain also gives a possibility to employ more complex and perhaps more useful software architecture components and interconnection in-between them.

As mentioned earlier, one of the main real-time embedded systems requirements is a minimal input-output signal latency and good overall real-time response. The good overall real-time response is basically due in case of algorithms, which are controlled from outside by external signals (feedback control). Since these need rather immediate reaction, they are strictly designed using hand-optimised assembly code and located in Layer 1 memory to assure safe critical execution. Minimal input-output signal latency is another area of focus. In order to understand how these criteria affect software architecture choice, we need to know how signal is processed within the hardware architecture itself. One of the main principal requirements for correct digital signal processing is a sampling and processing theorem so-called Shannon-Kotelnikov theorem [1]. It generally requires that any input signal with a certain bandwidth is being sampled at double bandwidth sampling frequency F_S. If there was an ideal case with virtual hardware architecture, we could achieve the latency of the system as good as $1/F_S$. The only specification would be the ability of the hardware architecture to apply the process algorithm on every sample individually. Real situation is however rather complicated. First need in the digital signal process chain is to convert analog form of the sound into the digital representation (AD) and vice versa (DA). AD and DA conversion serves for this purpose. Group of converters used for sampling audio signals due to its principle [1] unfortunately insert additional 2msec of signal latency into the digital stream. Tolerable overall signal latency based on human hearing perception is about 4-7msec depending on given application [3]. Signal processing within the embedded hardware architecture is basically controlled and managed using interrupt service routine. Each processing routine invocation requires several system clock cycles (varying from 2 to 10 per cent of all available cycles on one sampling period, depending on

CPU architecture). If invocation is done on every sample basis, the CPU spends service cycles pretty often, the overall input-output latency would only be $1/F_S$ however wasting a lot of computational power of CPU on mentioned service cycles. A solution to this computational loss is called a block processing technique [5]. This improves overall performance of embedded system by 2 to 10 per cent. The input-output latency requirement forces the embedded system software architecture to be designed so that it will be able to achieve such a performance. Most of the time, all of signal processing routines are designed using assembler hand-optimised code and executed from Layer 1 memory region for the highest performance. Only the simple software architectural components are allowed to use mostly in conjunction with circular buffers and direct variables.

2.2. Standard component modelling

So far we have been speaking rather of Real-time embedded system implementation and technical requirements. While it is crucial to adhere to these specifications strictly, they result in rather limited options for choosing real-time embedded system appropriate software architecture. There is no general software architecture available fulfilling all mentioned needs. Embedded systems however typically involve a combination of one or more partial software architecture types, which in proper utilization and combination create a very well performing software system. Basic partial software architecture types, which adhere to these specifications, are the following:

- Domain-specific systems: could be described as "reference" systems for a domain specific area of application. By specializing the architecture to the domain it is often possible to create an improved the descriptive power of structures used in the architecture. Embedded systems software architecture is assumed to be a domain-specific system indeed.
- Process control components: these architectural components are basically intended to provide very dynamic control over a signal processing environment with rather instant real-time response. In embedded systems, these are assembler written hand-optimised codes with very limited data structures focusing strictly on performance. They usually work with instant feedback control signals.
- Pipes and filters: are based on a structure of a black-box in the middle with a set of inputs and outputs. The black-box is the processing algorithm itself whereas it processes the inputs and generates the outputs. Their advantageous property is they could be interconnected in between each other in various orders thus creating possibility of highly configurable system if processing order on signal does make a difference in the final result. One of the example implementation in the real-time embedded system is any general audio processing algorithm used on a common circular buffer of audio samples. The advantage of using pipes-and-filters architecture here is that the order of process is easily defined since all algorithms work on the same data structure here. This gives programmer and user an easy way to modify the process itself on-the-fly in dynamic fashion thus achieving different process behaviours. The pipes-and-filters architecture is very well implemented in higher-level languages such as C due to its descriptive power to abstract structures rather than in assembly written code, which is more difficult to approach from the above mentioned point of view. Fast Layer 1 memories for execution critical code and data storage (available only in several kBytes)
- Layered systems: are ones of the most used software architecture components in embedded systems. Since a very precise scheduling of tasks is required in such systems, these are most useful for the given purpose. Some of the layered systems may even involve hiding certain outer layers from inner layers thus creating a sort of "protected" environment for sharper overall stability of the whole system. A good example is a real-time OS driven embedded application where core and peripheral drivers are the most inner layer in the system and the user applications are the most outer layer, communicating with the core via communication and scheduler layer. Inner layers usually also serve as interrupt service routines for sampling and processing. Since scheduling reliability of these must be 100 per cent they are usually (with the OS core when implemented) on the same layer with highest priority of execution. Otherwise missing process for input sample causes strong and audible signal distortion and degrades system performance significantly.
- Event-based invocation: is a specific software architecture component, which is useful for controlling user input and implementing user interface. The advantage of this architecture component is in how algorithms can be layered by their priority and still be able to communicate-invocate themselves in event-based fashion with very good real-time response. One good example for system of this kind is a simple volume regulator. The sound processing part of algorithm is running at very high priority applying gain constant which is read from the control component volume, while there is another software component on the outer layer priority which does not need such sharp timing and can run much slower (difference in schedule intervals is more than thousand times). The algorithm there serves reading the control component volume and storing appropriate gain value into the common gain variable used by both components. Another useful example is generating events on button components when these are pressed-released. Each given button has

assigned its own software component, which is always invoked only when an event occurs on the button itself.

To achieve a well performing overall system, standard modelling approach suggests careful evaluation of various partial software component architectures resulting in their balanced combination taking into account a strong software-hardware relation. Such approach guarantees very good real-time application performance however trading off easiness of software design, modelling and maintenance.

2.3. On-the-fly modelling requirement

We have already reviewed a real-time response requirement influence on modelling approach. There is however still a second user's requirement - ability to change sound processing model on-the-fly e.g. in real-time. Both mentioned qualifications of such Musical Real-Time Applications are nowadays solved rather in contradictory and limited fashion. Since the implementation and design (based usually on embedded systems) are rather time and programmer's resources consuming, the usual up-to-date solutions only offer a very limited range of configurability of applicable signal processing algorithms and meet only minimal ability for modification in real time.

The main reason of such limitations in standard implementations is a need for laborious hand-manner approach when designing embedded systems with focus on very good real time response. Such systems require careful evaluation of all requested parameters of the application as well as evaluation of the abilities of the target platform. Special care must be taken on the viewpoint of the Platform Model (PM) / Platform Specific Model (PSM) so to meet all Quality of Service (QoS) requirements requested by the Application as well as given by the Target Platform.

The application modelling must be measured from various viewpoints considering the resources of the target hardware architecture such as computational power of the core processor, various specific properties of memory types, memory sizes, communication channels between peripherals, language types and evaluation of mixed or hand-optimised programming need for critical parts of the system (OS scheduler, peripheral drivers, etc). Such design methods perform good in achieving very good real time response of the system, they also however causes nearly unsolvable difficulties when trying to implement the user's ability to modify user parts of application in real time - on-the-fly, all this without a need for programmer/architect or even recompiling whole project.

One of the ways how to deal with cases where modifiability on-the-fly is essential is to approach modelling using MDA and Executable and Translatable UML (xtUML).

3. MDA XT-UML APPROACH TO OUR APPLICATION TYPE MODELLING

MDA is defined by the OMG group [7], [8], [9] as modelling technique particularly focused on separating concerns while designing applications and systems. It basically defines three distinctive models used for various aspects description: PIM (Platform Independent Model), PM (Platform Model) and PSM (Platform Specific Model). Each mentioned model describes system through a specific viewpoint such as it separates concerns of the platform and the application from each other. The first mentioned model (PIM) characterises functioning and structure of the system with complete abstraction from the platform where the application suppose to be implemented. By employing such approach we can achieve complete separation of concerns and as a result the user – architect gains much better portability of the application between various platforms and architectures while meeting all requested properties, contents of the application and its behaviour as well. PIM model together with the PM model then server as base for transformation into PSM model, which includes all implementation specifications of the given hardware platform where the application has to be executed.

Such modelling technique is very beneficial for our application kind and it brings us several advantages. Sound and signal processing technologies are nowadays constantly evolving also in common PC computers. There is a superior trend in using Musical Real Time Application among artistic world as well. The MDA approach in the mentioned applications seems therefore a very spontaneous and valid modelling direction. Such approach assures a very good ability to implement the same model on various platforms – embedded systems as well as the platforms based on PC with operating systems while meeting all requested properties, contents and behaviour defined by models.

To use just the MDA for achieving all of the requested properties in case of our application kind will however be hardly satisfactory. One of the main required properties is the on-the-fly modifiability of the PIM user model. To implement such system only within the MDA prospects would be rather awkward especially due to unrealistic recompiling of the whole system in real time. We therefore propose a method which uses Executable and Translatable UML as the in-between layer within the PIM model. The resulting transformed PIM model is executed via xtUML specification within already defined architecture through the PM and PSM models, which are designed to execute the xtUML PIM model in real time with a very good real time response.

The xtUML standard [11], [12] is defined so that it can design comprehensible model for an application without any knowledge about how the internal structure of the software is made. This fact makes the xtUML a well abstract tool, which elegantly solves also our application kind modelling. Not to mention there is a possibility to implement such xtUML model on various already existing platforms also for verification purposes. This part includes the data on the measuring method and instruments as well as experimental results.

The figure 2 presents a proposal of modelling technique of our application kind using the MDA approach with xtUML models. Due to the need of modifiable user PIM model in real time, a middle layer is embedded in between the user PIM and the resulting xtUML PIM which provides for a transformation of the user PIM model into the xtUML specification. The user

PIM model is rather specified in a so-called Easy Modelling Environment. The resulting xtUML model is directly executed on a target platform.

Such transformation "middle-man" allows the user – artist skip unnecessary learning of the xtUML standard and at the same time allows achieve more important aspect of modelling the user PIM model - using very straightforward and easily understandable modelling environment (for example graphically specified). Such environment is to be especially designed with focus on Musical Real Time Applications specifications with high focus on ergonomics and time-savings while designing the model. The mentioned transformation middle-man as well as other parts of the proposed modelling method is subject for further research included in my ongoing PhD thesis.

The advantages arising from the above-mentioned transformation are multiple. One of them is the ability to simulate and verify [10] resulting xtUML model due its execution with addition of error feedback to the user about correctness of his model.

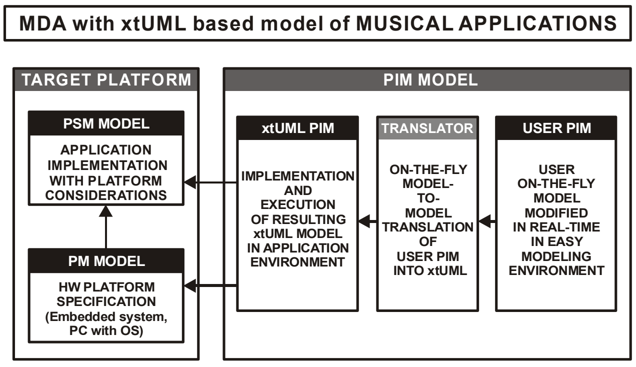
MDA with xtUML based model of MUSICAL APPLICATIONS
TARGET PLATFORM
PSM MODEL
APPLICATION IMPLEMENTATION WITH PLATFORM CONSIDERATIONS
PM MODEL
HW PLATFORM SPECIFICATION (Embedded system, PC with OS)
PIM MODEL
xtUML PIM
IMPLEMENTATION AND EXECUTION OF RESULTING xtUML MODEL IN APPLICATION ENVIRONMENT
TRANSLATOR
ON-THE-FLY MODEL-TO-MODEL TRANSLATION OF USER PIM INTO xtUML
USER PIM
USER ON-THE-FLY MODEL MODIFIED IN REAL-TIME IN EASY MODELING ENVIRONMENT

Fig. 2 A proposed MDA with xtUML based model for Musical Real Time Applications Common notes

4. CASE STUDY

To help understand and evaluate the needs and properties of the Easy Modelling Environment as well as the specifications of the model-to-model transformation into xtUML we propose to further study a real application case. The modelling environment for digital effector application with real time modelling ability is to be designed. The application should also implement the artificial intelligence methods, which should help improve ergonomics and easiness of the user PIM modelling process. The information such as musical genre, beats-per-minute, gathered from the musical material are to be used in mentioned AI methods as basis for modelling process optimization. As for the target platform we suggest to use a combination of two complementary platforms. An embedded system will provide for sound alteration algorithms execution as well as musical properties information analysis. The PC platforms based on standard operating systems will provide for the Easy Modelling Environment of the user PIM models and will even allow using multiple parallel-executed environments at the same time. This would allow more users - artists to perform at the same time thus creating more complex performance advancing creativity possibilities in Musical Real Time Applications. The scheme on figure 3 briefly describes such architecture example.

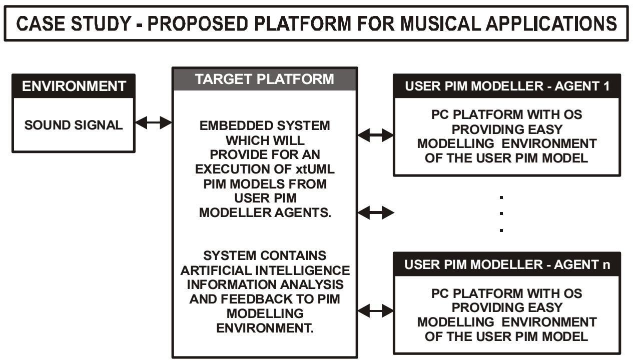
CASE STUDY - PROPOSED PLATFORM FOR MUSICAL APPLICATIONS
ENVIRONMENT
SOUND SIGNAL
TARGET PLATFORM
EMBEDDED SYSTEM WHICH WILL PROVIDE FOR AN EXECUTION OF xtUML PIM MODELS FROM USER PIM MODELLER AGENTS.
SYSTEM CONTAINS ARTIFICIAL INTELLIGENCE INFORMATION ANALYSIS AND FEEDBACK TO PIM MODELLING ENVIRONMENT.
USER PIM MODELLER - AGENT 1
PC PLATFORM WITH OS PROVIDING EASY MODELLING ENVIRONMENT OF THE USER PIM MODEL
USER PIM MODELLER - AGENT n
PC PLATFORM WITH OS PROVIDING EASY MODELLING ENVIRONMENT OF THE USER PIM MODEL

Fig. 3 A proposed case study platform for Musical Real Time Applications

4.1. Current experiment results

Due to the fact that the work on subject and described methods is ongoing, there are currently available experimental results of some subsystems only. These are briefly presented in the below text. Up to date a design of the implementation embedded system hardware platform was carried out together with the design and performance evaluation of basic core software parts of the embedded system as well as some sound processing components providing the user with the required sound effects.

A Blackfin digital signal processor [4] from Analog Devices was used as a basic core hardware component, since it includes rich set of multimedia instructions as well as all peripherals necessary to accommodate such sound processing system. Peripherals were necessary for analog-to-digital and digital-to-analog audio converters interconnection, additional peripherals were necessary for interconnection of the embedded system platform and PC platform via Ethernet interface which should provide transfer of model data in real-time. Due to already mentioned requirement of fast real-time response of user intervention and configuration while creating and interpreting the resulting user model, it was necessary to design the component/task scheduling core of the embedded system to meet the specified requirements. The designing methods of such system with focus on task scheduling and implementation are described in more depth in [13, 14]. Finally, the experimental embedded system software architecture core was designed where a simple interrupt-driver dynamic task priority scheduler was used for component scheduling. Time-critical core components such as audio drivers or communication drivers were implemented as direct interrupt service routines and are executed by a direct hardware-driven invocation depending on requests from various peripherals (converters, Ethernet port, memory, etc.). Due to eligible core processor hardware architecture choice, the interrupts have nesting possibility which provides for optimizing time-critical components execution by direct hardware means. Mentioned architectural advantage results in achieving reaction and configuration response times of all sound processing components in range of ones to tens of miliseconds which well meets the overall real-time response specifications.

Due to demonstration and study purposes only few basic sound processing components were designed and verified on the embedded system. These include the following:

- Filter component,
- Flanger component,
- Phaser component,
- Delay component,
- BPM meter component,
- Mixing component.

All of the mentioned components reaction and configuration response times were measured using logic analyzer. With dynamic execution and configuration applied, the maximum response times were obtained in range up to 15msec, depending on execution of other supplementary component functions within the embedded system. The measured response time can eventually degrade in some per cent due to necessity of PC platform and embedded system platform communication execution which is not yet done.

Further research work is planned on designing and implementing the Easy modelling environment for the PC platform as well as implementing the PC and embedded system Ethernet communication as a transformation layer between the user PIM model and the implementation itself. There is also a need to refine specification of the sound processing component models with their modelling properties on the PC platform as well as putting the artificial intelligence methods to work at optimizing modelling process with its properties.

5. CONCLUSIONS

This paper presents a MDA approach with use of the Executable and Translatable UML for modelling Musical Real Time Applications with special focus on the modifiability of the user PIM model on-the-fly. The core idea of the proposed method is to separate user PIM model from the implementation PIM model of the rest of the application and a method is suggested for modelling the user PIM model using a specific Easy Modelling Environment with high focus on ergonomics and easiness of the model design. Such modelling environment should facilitate time-saving while modelling as well as be very understandable in the definition to the user – artist. Resulting xtUML PIM model is achieved by transformation of such user PIM model and is afterwards executed on the target platform.

Such method also guarantees various advantages over standard modelling methods and approaches. One of the advantages is easy portability over various platforms enabling to use the embedded systems as well as common PC platforms for execution. The major advantage is however the ability to modify the user PIM model on-the-fly in real time.

ACKNOWLEDGMENTS

This work was supported by VEGA Grant No. 1/0350/08 Knowledge-Based Software Life Cycle and Architectures.

REFERENCES

[1] Lathi, B. P.: Signal Processing & Linear Systems, Oxford University Press, New York (1998).

[2] Pioneer Pro DJ, EFX-1000 performance effector, available at http://www.pioneerprodj.com/dj-equipment/effects/efx-1000.asp, 2006.

[3] Gold, B., Morgan N.: Speech and Audio Signal Processing: Processing and Perception of Speech and Music, Hardcover, August 1999.

[4] Analog Devices: ADSP-BF533 Blackfin Processor Hardware Reference, Analog Devices One Technology Way, USA 2003.

[5] Ko, M., Shen, Ch., Bhattacharaya, S..: Memory-constrained Block Processing Optimization for Synthesis of DSP Software, International Conference on Embedded Computer Systems, Samos Greece 2006.

[6] Garlan, D., Shaw, M..: An Introduction to Software Architecture, CMU Software Engineering Institute Technical Report, CMU/SEI-94-TR-21, ESC-TR-94-21.

[7] OMG, Model driven architecture guide v1.0.1. http://www.omg.org/mda, 2003.

[8] OMG, UML 2.0 OCL specification. http://www.omg.org/ocl, 2003.

[9] OMG, UML profile for modelling QoS and fault tolerant characteristics and mechanisms. OMG adopted specification, 2004.

[10] DeAntoni, J., Babau, J.P.: A MDA-based approach for real time embedded systems simulation, Ninth IEEE International Symposium on Distributed Simulations and Real-Time Applications, 2005.

[11] Mellor, S.J., Balcer, M.J.: Executable UML: A Foundation for Model-Driven Architecture, Addison-Wesley Professional, 2002.

[12] Mellor, S.J.: Executable and Translatable UML, available on April-2008 at Embedded Systems Design http://www.embedded.com/story/OEG20030115S0043, 2003.

[13] Železník Otto, Havlice Zdeněk: Software Architectures for Real-Time Embedded Applications for broadcasting, Information Systems and Formal Models, 10th International Conference on Information System Implementation and Modeling ISIM'07, 2nd International Workshop on Formal Models WFM'07, Hradec nad Moravici, 23.-25.4.2007, Opava, Silesian University in Opava, Faculty of Philosophy and Science, Bezruc Sqr. 13, 74601 Opava, Czech, 2007, pp. 63-70, ISBN 978-80-7248-006-7.

[14] Železník Otto, Havlice Zdeněk: On-the-fly MDA application modelling using Executable and Translatable UML, Model Driven Software

Engineering Workshop on Transformations and Tools, MDSE 2008, Berlin, 11.-12.12.2008, Germany

Received March 2, 2009, accepted September 28, 2009

BIOGRAPHIES

Otto Železník was born on 7.7.1977. In 2000 he graduated (MSc.) at the department of Computers and Informatics of the Faculty of Electrical Engineering and Informatics at Technical University in Košice. He is currently studying his PhD and his scientific research is focused on software architectures modelling in embedded systems.

Zdeněk Havlice was born on 14. 02.1958. In 1982 he graduated (MSc.) with honors at the Department of Computers and Informatics of the Faculty of Electrical Engineering and Informatics at Technical University in Košice. He defended his PhD. in the field of visual programming and user interface design in 1991; his thesis title was: "Design of User Interface for Dialogue Systems". Since 1999 he is working as an associated professor at the Department of Computers and Informatics. His scientific research is focusing on the area of special languages, compilers, CASE systems, software methodologies, methods and tools.