Knowledge Based Embedded System Modeling With Real-Time Response Requirements

Otto Zeleznik and Zdenek Havlice

Abstract—**The paper presents a proposal of a hybrid method for embedded system modeling with emphasis on constraints analysis. Since one of the major factors for most embedded system from the performance point of view is the real-time response of component execution the proposed method focuses on analysis of attributes affecting such performance requirements and optimizes the design process accordingly. The core idea of the method is to separate components into three distinctive groups as critical, hybrid and non-critical and select an appropriate memory and resources sharing model accordingly. Experimental case study is provided to verify the method performance compared to other available modeling methods like best-effort or dependable embedded system design methods. The hybrid method shows its advantages in more effective use of available implementation platform resources while meeting all of the required real-time response constraints.**

Index Terms—**Embedded systems, information systems, real-time response, software architecture modeling.**

I. Introduction

Information systems in recent decades made significant progress in their development and subsequent applications. Their practical use, thanks to technological development in recent years has further expanded into areas with which designers and software engineers in the past did not count at all. One such area is the application of information systems and related modified software engineering methods, tools and technologies on objects of the physical environment, i.e. embedded systems [1], [2].

The Embedded systems nowadays draw more attention mainly due to the positive trend in the development of new microprocessor technologies. Since embedded systems are a specific type of information systems, theirs design methodologies have always been different from the general information systems design methodologies. The main reason, despite the increasing computation performance of available embedded system technologies, is the fact, that there will always be a strong link between required system functionality, existing implementation platform and real environment objects on which the mentioned embedded system functionality is applied. Such interconnection between the embedded system computation process and the real physical environment arises on the basis of two main interactions: Reaction to the actual physical environment and computation process execution on the real implementation platform. Based on these two interactions we can define two types of physical constraints: the reaction constraints and the computation process execution constraints.

One of the important reaction constrain that creates a major influence on proper embedded system functionality is the real-time response and its management, discussed in more detail in [3]. When designing such embedded system, it is essential to take into account also the real-time response requirements of every component in the system. It may seem to one that achieving better real-time response could be solved simply by deploying computationally more powerful technology. The real-life situation however suggest that the current embedded system applications real-time response requirements are either on the edge or are far exceeding the capabilities of implementation technology available. To successfully resolve such problem it is therefore necessary to select an appropriate methodology for embedded systems design, optimize processes such as embedded systems component execution management and focus on each type of application problem separately to achieve the expected quality of service parameters.

Several domain specific attributes usually describe embedded systems typical properties. Apart from the already mentioned real-time response, those also include:

- *Reliability* – Embedded systems should provide certain degree of reliability due to their deployment in harsh conditions or in critical applications. Such embedded system usually includes several redundant subsystems and employs various means for improving reliability and error tolerance.
- *Use of specialized hardware* is selected when group of special functions is to be implemented. One suitable example is signal processing domain where the digital signal processors are the core hardware element.
- *Low cost* plays an important role in many areas of embedded system applications. We can consider it a key parameter in most of advanced digital electronic designs of today's commercial electronic devices. Such attribute then influences selection of the implementation technology and enforces designers to proceed with thorough design optimization.
- *Substantiality* is a key parameter for embedded system deployment in environments with harsh natural conditions. As a good example can server an automotive

Manuscript received August 24, 2011; revised December 25, 2011. This work was supported by the Slovak Research and Development Agency under the contract no. APVV-0008-10 and by the Scientific Grant Agency of the Ministry of Education - VEGA Grant No. 1/0531/12 Use of Knowledge in Software Processes.

O. Zeleznik is with the Department of Computers and Informatics, Faculty of Electrical Engineering and Informatics, Technical University of Košice, Letná 9, 042 00 Košice, Slovakia (e-mail: o.zeleznik@gmail.com).

Z. Havlice is with the Department of Computers and Informatics, Faculty of Electrical Engineering and Informatics, Technical University of Košice, Letná 9, 042 00 Košice, Slovakia (e-mail: zdenek.havlice@tuke.sk).

.

or aerospace industry. Embedded systems deployed in such environment must withstand extreme thermal stress, excessive mechanical jitter or significant power fluctuations. The designer must ensure the system will function properly even at such stress levels.

- *Flexibility* requirement forces some system designs to proceed with microprocessor solution rather than discreet components design. The resulting advantage is the ability to allow for flexible reprogramming of the same implementation platform for various different products using same implementation platform thus lowering overall cost.
- *Low power consumption* is utterly essential attribute when one attempt to design a battery powered embedded system. Such systems require architecture designers to be focused most of the time on lowering power consumption to extend the battery life as much as possible. They use techniques like dynamic system clock frequency decrease or individual processor segments power down to achieve it.

Another embedded system partitioning is based on several distinctive design criteria and principle viewpoints:

- *Distributed* embedded systems use components which can be physically located at different places distant from each other. Partial computations can be executed on multiple distant microprocessors interconnected via a communication network. Such solution is usually less expensive since several 8bit microprocessors have lower cost than its few 32bit rivals. Furthermore, multiple distributed microprocessors can increase failure resistance compared to one large type. Distributed systems also simplify and enhance complex systems ordering – they allow for individual subsystems to be installed at locations required by physical nature of given solution.
- *Centralized* embedded systems are usually designed as such because the implementation environment does not allow proceeding with distributed type. The main limitation factor can be physical size constraints of the implementation platform itself. The designed is then forced to use more complex microprocessors which are capable of providing all necessary implementation resources to perform all computations and communications at one place.
- *Reaction-based* type of embedded systems is a special group where the computation process requires a feedback type signal to be provided as a main computation parameter. Such approach creates a reaction on processed parameters with no regards to input parameters of the system.
- *Transformation-based* embedded systems are a complementary to the reaction-based types. The core difference is they do not process the feedback parameter at all, but they only process input parameters via internal transformations controlling thus output parameters according to the requirements and definitions.
- *Control-dominant* embedded systems are focused on regulation applications and are basically designed to achieve very good response on external asynchronous events which occur within the application environment. Such systems must carefully manage priority of processed events and they use special architectures to achieve that [4].
- *Data-dominant* embedded systems typically consist of architecture optimized on effective data operations and transfers. Data is often generated or gathered within a certain fixed time period e.g. sampling period. Signal processing architectures are considered an example of data-dominant embedded systems [5].

One of the main individualities when designing embedded systems is the necessity to define properties and constraints of the software as well as hardware architecture well in advance of the design. This fact forces the designers and programmers to use particular design methods which rather differ from design methods of standard information systems. When designing a general information system, one usually employs an abstract representation, such as model. This model is created from the input user/designer requirements and constraints. The mentioned model is later used as a formal specification for complete automated system generation. For example, in case of software design process, we need a compiler capable of generating output code. In case of hardware design, the designer uses an abstract hardware description. Such abstract description layer is later used in specialized software tools that generate the actual hardware schematic layout. Both design cases are however similar in sharing some common design elements used in design process. Adaptation and re-use of already designed components eventually found in libraries, systematic step-by-step model modifications are the methods that ensure safe fulfillment of the input requirements and constraints. Even though the described methods show lots of similarities with general information system design methods, the real-life situation is quite different when it comes to details in embedded systems design. The embedded systems by their nature almost always respond to the values coming from the real physical environment. It is therefore required to employ a more complex viewpoint at such systems in the design stage, joining knowledge from various areas of engineering, informatics, software design, hardware design, regulation theory, signal processing and others and applying those at the design at once.

II. Architectural-Semantic Embedded System Modeling Methods

In the chapter below are described few essential embedded systems modeling methods which emphasize architectural and semantic modeling. Since full description of these methods is out of scope of this paper, these are covered in greater detail in related papers referred in the paragraphs below.

A. Modeling Method Based on Programming Languages and Synthesis

The modeling method based on programming languages and synthesis is considered a standard approach in computer systems modeling. The programming languages modeling methods are based on software design. They are generally in close link with a particular programming language with its own runtime environment, provided rather with fixed priority multitasking capability. As examples of such systems it can be considered Ada, RT-Java. Synthesis methods on the other hand come from the hardware and electronic circuits design. They are based on processing the system specification in a structured form or in form of HDL language fragments, which are afterwards used for automated implementation generation basically adhering to specified input constraints [6].

B. Modeling Method Independent on Implementation Platform

Such method is considered a new generation of the design and modeling methods. It is basically enforcing a semantic separation of the design layer from the implementation layer. As such, the method gains an advantage of creating a new level of independence from any particular implementation platform. Several examples are available. Synchronous programming languages for example already embed the abstract hardware semantic layer (synchronization properties), implementation technologies are available on several various platforms not excluding architectures with timed execution synchronization. The SystemC programming language [7] combines synchronous hardware semantics layer with asynchronous execution algorithms coming from the software, the implementation layer requires fragmentation on components, which are then realized on the hardware platform as well as within the software itself.

C. Modeling Method Independent on Execution Semantics

Modeling method independent on execution semantic is based on object-oriented modeling approach types and modeling tools like UML (Unified Modeling Language) or AADL (Architecture Analysis and Design Language) [8], [9], [10], [11]. Mentioned modeling tools try to be as generic in abstract description domain, not only for the implementation platform but also for the execution platform and interaction semantics of the system. Due to this method, the designer acquires rather valued independence of any specific programming language and at the same time the method allows for emphasis on the architectural design itself, better organizing computation process criteria, communications and other resources which are provided by the given architecture. The method is dealt with in greater detail in [12], [13], [14], [15].

III. Embedded Systems Modeling Methods With Emphasis on Constraints Analysis

Embedded systems by their nature mostly interact directly with physical variables of the real ambient environment and thus control and analyze time-variant dynamic processes. To guarantee a correct behavior of implemented functions and components in a role of control processes, it is necessary to design and implement the application code the way that it will maintain correct execution not only from the semantics point of view but also from the strictly defined real-time response point of view. Let‘s consider the component ADD will perform an operation Y = A+B in time t1, let’s define t2 as component‘s required real-time response based on the application input properties. In case of t1 > t2, we can consider the performance of the mentioned ADD component inadequate for some embedded system scenarios thus invalidate its results completely no matter whether the semantic part of the component result is correct or not. Due to the existence of special areas of embedded systems applications it is of equal design importance to consider not only the semantic correctness of the components but also to guarantee component real-time response-correct execution. The area of embedded system applications where it is essential to analyze the real-time response of various variables and components is wide-spread. Most of the mentioned applications can be found in the theory of regulation and the signal processing, where the real-time response-rules for individual operations are defined very strictly. Detailed description on the signal processing components response theory can be found in [16].

The mentioned necessity to define the real-time response has direct influence on architecture design process for any selected embedded system. Most of the influence will appear as a strong link between software architecture and implementation hardware architecture. The application modeling 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-optimized 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 [17] 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 kilobytes), 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 optimized 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-optimized hand-written assembly code. Gain in performance is about 5 to 25 per cent in favor of hand-optimized 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 time-critical 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 [18]. 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.

For easy illustration purposes of the above mentioned modeling characteristics for all further descriptions we will consider using component architecture as framework architecture due to its self-declarative nature [19]. Let's consider that every component of such architecture has got its semantic definition well defined side by side with its proper real-time response requirement definition as well. Based on the above mentioned embedded systems input constraints with emphasis on modeling optimization we suggest defining two basic attribute types which evaluate the execution success rate for each component as follows:

- First attribute, so-called *rate of semantically-correct execution*, quantifies rate of successful component execution strictly from the semantics viewpoint. To consider any component to be semantically correctly executed, it is necessary for such component to generate a correct result (from the viewpoint of expectations) and that being generated with no association to time response needed for achieving such result.
- Second attribute, so-called *rate of real-time response-correct execution*, quantifies rate of successful component execution in comparison with the component's response time definition. To consider any component to be response-correctly executed, it is necessary to guarantee the execution to be performed within the response time smaller or equal of the defined real-time component response. This parameter however does not regard the semantic correctness of the output result.

When designing an embedded system, the designer usually evaluates which of the two attributes has priority over the other for the overall system performance and properties and that fact selects the appropriate design method. In case when the rate of real-time response-correct execution has the same priority as the rate of semantically-correct execution, the dependable embedded system design method is selected despite the increase in computation power requirements and advanced complexity and cost of the resulting implementation platform. Right the opposite case is an embedded system where very few to none components require deterministic execution in time, e.g. rate of real-time response-correct execution is not necessarily at 100% requirement, or another important criteria is the overall system cost. Occasional execution response delays in such cases do not jeopardize the overall functionality of the embedded system. Should some or all of the mentioned requirements be met, it is preferred to use the best-effort embedded system design method.

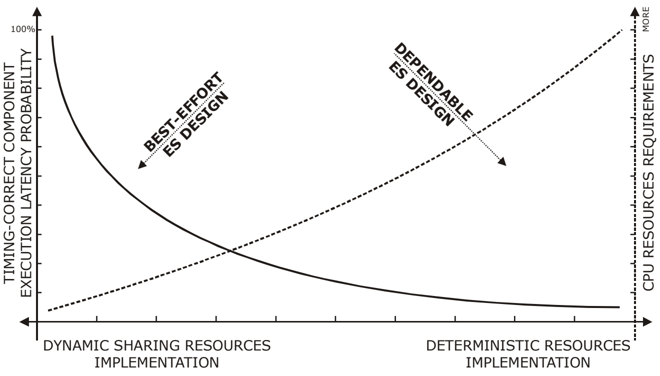

Fig. 1. Best-effort embedded system modeling vs. Dependable embedded system modeling properties.

The Fig.1 depicts characteristic parameters of both mentioned design methods. Illustrated is the relation between real-time response-correct component execution delay probability and the approach chosen for component design. It can be seen from the chart that best-effort design method saves lots of implementation platform resources at the resulting design, while it is necessary to use dynamic computation resources sharing. The trade-off here however is raising probability of real-time response-correct component execution delay. In case of deterministic design approach selection such as the dependable embedded system design method, the response delay probability is minimized although for a price of increased implementation platform complexity and increased required computation resources. The next two chapters describe in detail both the best-effort embedded systems design method and the dependable embedded systems design method.

A. Dependable Embedded Systems Design Method

Dependable embedded systems design method attempts to provide as the result the most dependable and reliable embedded system with no regards or constraints to the effort well spent on either design process or platform resources themselves. Individual components must fulfill the required (usually almost) 100% rate of real-time response-correct execution. Tolerance for response delay probability is extremely low and depends only on particular designed embedded system or its application domain constraints. Such requirements are imposed on the dependable systems despite very harsh implementation environment conditions mainly for their best performance and reliability interest. The described design method is thus based on as conservative as possible strict analysis and estimation of the system dynamics and employing rather static system resources allocation on any intended implementation platform. Such conservative estimation and analysis results later force designers to use mostly very simple implementation platforms with no operating system support. Another suitable processor architectures are those which are capable of deterministic code execution response evaluation. Typical example of dependable systems are control systems found in automotive or aerospace industry. Mentioned examples require the real-time response parameters to be defined as „hard". Such hard real-time response is achieved by employing the worst-case scenario execution timing analysis as well as static code execution planning. The maximum computation resources for code execution must be available within the implementation architecture at all times. Other ways to improve reliability and dependability of the embedded system is by using means of redundancy as well as implementing subsystems for error detection and correction. Appropriate example of dependable embedded system design methods is Time-triggered Architecture (TTA) [20], [21]. This architecture is used by the designers in the safety-critical applications like brake-by-wire, drive-by-wire, fly-by-wire, used in automotive and aviation applications. The TTA node is composed of the two subsystems: communication unit and main computation unit. All communication within the nodes of such system has strict ground rules defined well in advance and is based on static synchronous timing. Every communication unit has precisely defined message architecture description, which defines at what time is what node allowed to gain authorization to send or receive a message with what peer node, all of it precisely defined in advance. Synchronization protocol is further defined to provide means for error and fault correction and is distributed throughout the whole system as a part of synchronization process.

B. *Best-Effort Embedded System Design Method*

The dependable embedded systems design method employs the worst case code execution timing analysis to meet the required real-time response. The Best-effort embedded systems design method on the other hand prefers the code execution timing analysis to be done for an average case. There could also be situations where there is an a priori definition of tolerable component real-time response delay. Such method could also be described as optimization method since it attempts to dynamically optimize performance parameters of any given embedded system implementation while expecting the embedded system to work within the specified implementation constraints. Another considerable approach within the best-effort design methodology is dynamic resources allocating and sharing. This allows for effective computation and communication resources to be spread across the individual components or system sections thus optimizing system throughput and achieving balanced resources load which indirectly lowers overall cost of the implementation platform as well. The trade-off for such cost savings and simple architecture advantage due to the dynamic resources sharing is unwanted occasional performance decrease of individual subsystems that may at the given time request to use any particular system resource (communication peripheral, computation unit) but the resource is not available due to its allocation by other subsystem that may be busy as well. Such performance decrease may in extreme circumstances result even into total temporary failure of any given subsystem or component. Due to the soft real-time response requirement is however such behavior of the best-effort embedded systems reasonably accepted, depending in rate on given particular application. The quality-of-service parameter is in case of best-effort method provided thanks to the use of adaptive mechanism of code execution scheduling as well as thanks to use of feedback signals for dynamic control and optimization of executed code. Described methods ensure maximum computation process optimization as well as effective behavioral anomalies solving. As the characteristic examples of best-effort designed embedded systems may serve various multimedia and communication systems. These systems often use optimization methods which ensure that different services and processes are provided to different users with different priority in different time. The overall average system performance is however at the level specified yet in advance. Small deviations from the average performance figures (sometimes occurred due to timing errors) are however well tolerable and do not cause system malfunction.

IV. Hybrid Embedded System Design Method With Emphasis on Constraints Analysis

There are many examples in real life world when the designer needs to create an embedded system which shows requirements employed partly in both mentioned design approaches. Based on this fact, it is essential to make a compromise and rather choose either one or the other design method. The result is a system which is either too expensive and thanks to its complex deterministic-designed implementation architecture is hard to modify and maintain, or the system in case of best-effort method use is unreliable enough and in some critical situations is not uncommon to see either serious parameter worsening or partial system failure.

Fair resolution of depicted compromise scenario was to design so-called hybrid design method that combined advantages of both above described design methods with no obligation to employ any trade-off solution. Basic concepts of the hybrid design method are summarized in the following paragraph:

- Use of component architecture as framework architecture because of its simple self-illustrative nature.
- Estimation of required rate of real-time response-correct

execution depending on the input constraints and requirements.

- Embedded system component arrangement based on the rate of real-time response-correct component execution, split into three distinctive groups based on priority, memory model applicability and computation resources allocation as follows:
- Critical components – required rate of real-time response-correct execution 90%-100%, preferred static computation resources allocation model, preferred critical memory model (strictly deterministic memory type).
- Hybrid components – required rate of real-time response-correct execution 1%-99%, preferred dynamic computation resources sharing model, preferred hybrid memory model (use of deterministic and non-deterministic memory type, static and/or dynamic allocation).
- Non-critical components – required rate of real-time response-correct execution of no significant value, preferred dynamic computation resources sharing model, preferred non-critical memory model (external non-deterministic memory type).
- Application of conservative deterministic worst-case scenario time analysis for group of critical components, static a priori allocation of all computation and memory resources for the mentioned component group.
- Worst-case, best-case and average-case estimation for the hybrid component group, design of appropriate dynamic resources scheduling mechanism based on input requirements and rate of real-time response-correct execution, estimation of necessary partial static resources allocation used for correct hybrid components dynamic resources management.
- Worst-case, best-case and average-case estimation of rate of real-time response-correct execution for non-critical components.
- In case of unfit resulting parameters of non-critical or hybrid components, the component re- qualification is essential, depending on constraints, either into non-critical, hybrid or critical group, possible component attributes change, following repeated rate of real-time response-correct execution analysis.

A. Memory Layer Model of Hybrid Embedded System Design Method

Memory layer model of hybrid embedded system design method presents an interconnection of the three mentioned component groups (critical, hybrid, non-critical) with the type of memory available within the implementation architecture.

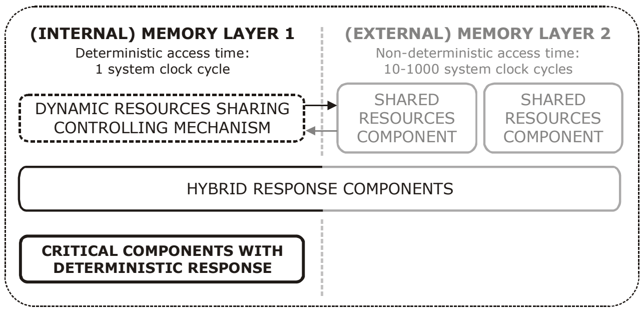
(INTERNAL) MEMORY LAYER 1
Deterministic access time: 1 system clock cycle
(EXTERNAL) MEMORY LAYER 2
Non-deterministic access time: 10-1000 system clock cycles
DYNAMIC RESOURCES SHARING CONTROLLING MECHANISM
SHARED RESOURCES COMPONENT
SHARED RESOURCES COMPONENT
HYBRID RESPONSE COMPONENTS
CRITICAL COMPONENTS WITH DETERMINISTIC RESPONSE

Fig. 2. Hybrid method memory model.

Fig. 2 of the memory mode depicts basic separation of two distinctive memory access types, e.g. deterministic and non-deterministic.

The deterministic memory type, most of the time of the highest throughput and shortest access time is used for critical components and the application has generally a priori defined static allocation.

Components of the hybrid group, depending on their rate of real-time response-correct execution, require reasonable decomposition of individual component parts and variables as into deterministic so well into non-deterministic memory types too. Appropriate ratio and partially static or dynamic allocation ensures achieving required rate of real-time response-correct component execution.

Non-critical component group puts the least requirements on memory access type and throughput. Since there is no significant requirement on rate of real-time response-critical execution, usually in most cases non-critical components may be assigned to a non-deterministic memory type with least speed grade and throughput. The non-critical component group also shares some of common computation and memory resources, it is therefore essential to employ a resources sharing and managing mechanism. Such component core is usually allocated statically in a faster memory type due to the need of effective and faster-than-component-itself behavior. Any particular allocation type however depends always on given embedded system application.

B. Knowledge Base Layer of the Hybrid Embedded System Design Method

The knowledge based artificial intelligence systems helps designer to solve complex modeling related tasks which are unsolvable by means of algorithmic resolution under normal circumstances. Such systems include also the knowledge based systems and expert systems. These can be well applied on optimization of embedded system design based on the implementation platform information analysis and input constraints processed by such system. Fig. 3 illustrates a framework architecture of a knowledge layer of the employed hybrid embedded system design method.

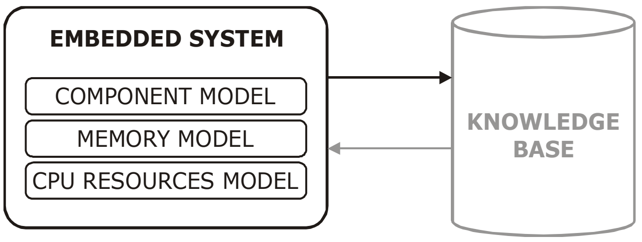

Fig. 3. Hybrid method knowledge-base architecture.

The advantage of using knowledge based layer when designing an embedded system is the capability to automate some of the design processes which are determined on the available knowledge base. Such (data)base often contains multiple application domain specific knowledge information that usually can not even be known by the individual designers due to its rather extended expert scope. If such modeling approach is employed, it is usually demonstrated that due to better expert knowledge of particular application details that are already gathered within the knowledge base (specific system dynamics, structural particularities, etc.), the overall system safety as well as component and architecture stability is greatly improved. For further architecture improvements, the designer is recommended to use the knowledge layer on various levels of modeling process. The knowledge base can be employed on individual modeling levels, also while performing any later system maintenance, or even while system run-time in real-time, if it is necessary to modify any part of the model on-the-fly as well as check its integrity and functionality as is suggested in [22].

The knowledge base layer of the hybrid embedded system design method serves mainly for optimizing component design process for various expert application domains with strong link to the architectural attributes of such embedded system. The knowledge base layer further optimizes and verifies the memory model design (selection of component and memory model link based on system constraints) and also manages computation resources model. It is reasonable to further extend a knowledge base with additional parameter types or design specifications in relation with the considered implementation platform.

V. Case Study Architecture

A case study embedded system architecture layer model was designed for the needs of experimental verification, including the knowledge base layer, depicted on Fig. 4.

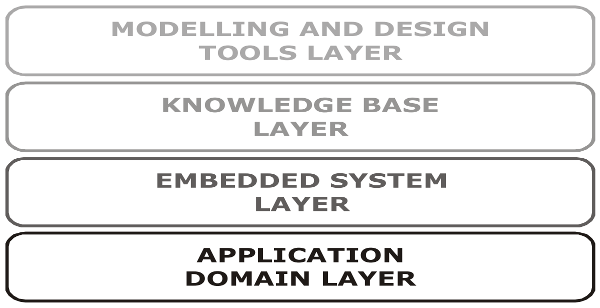

Fig. 4. Layer model of the Hybrid method architecture.

A described hybrid embedded system design method was employed for the individual component design. The design process was successfully verified for every defined component group (critical, hybrid, non-critical). Due to the performance comparison possibility of each design method the case study architecture design was carried out by all three discussed methods, the dependable embedded system design method, the best-effort embedded system design method as well as hybrid embedded system design method.

Since the application domain of the experiment is signal processing, particularly a sound processing, the implementation platform was based on the digital signal processor components. The selected Blackfin family [23] comes from the Analog Devices manufacturer and is considered to be top of the class at the given time. The case study embedded system architecture is depicted on Fig.5.

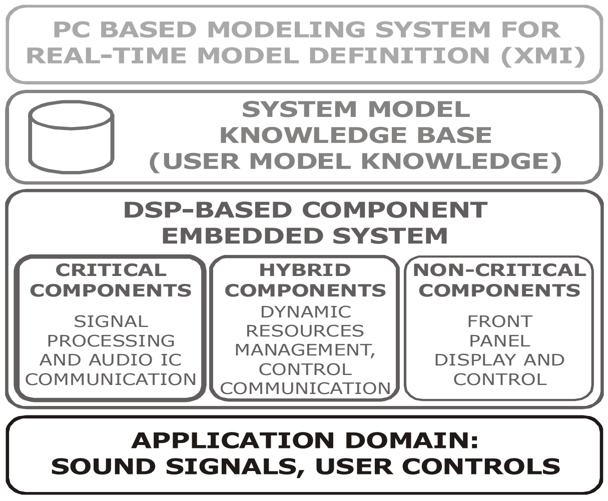
PC BASED MODELING SYSTEM FOR REAL-TIME MODEL DEFINITION (XMI)
SYSTEM MODEL KNOWLEDGE BASE (USER MODEL KNOWLEDGE)
DSP-BASED COMPONENT EMBEDDED SYSTEM
CRITICAL COMPONENTS
SIGNAL PROCESSING AND AUDIO IC COMMUNICATION
HYBRID COMPONENTS
DYNAMIC RESOURCES MANAGEMENT, CONTROL COMMUNICATION
NON-CRITICAL COMPONENTS
FRONT PANEL DISPLAY AND CONTROL
APPLICATION DOMAIN: SOUND SIGNALS, USER CONTROLS

Fig. 5. Layer model of the Hybrid method case study architecture.

The dependable-system-like design showed an increased architectural complexity of implementation throughout the whole design process – it was necessary to employ four parallel DSP processors to achieve successful implementation of the given case study. Due to the very limited Layer 1 memory it was inevitable to distribute all of the designed components into those mentioned four DSP processors which resulted in necessity to implement additional utility component for inter-processor parallel execution and process synchronization management. Another disadvantage was to utilize low-level programming languages due to the critical nature of designed components as well as other hardware constraints (memory size and speed). Achieved component execution response time was however approaching the physical (electrical) capabilities of serviced peripherals. The real-time response of processed sound samples which directly influenced the input-output sound delay was in the range of few single samples, e.g. depending on sampling frequency in range of few tenths of microseconds. Such achieved real-time response times when evaluated from the psychoacoustic perception viewpoint are supernumerary in case of general audio systems use and are only necessary in case of specific measurement equipment. Common acceptable psychoacoustic perception delay times are within the one-millisecond range, discussed more in detail in [3], [24]. Based on all the above results, the dependable embedded system design method appeared unreasonable for such case study embedded system design due to its excessive increase in architecture complexity.

The best-effort method on the other hand was capable of reducing the required DSP processor amount to only one. Unfortunately due to its design nature the resulted architecture could not function properly especially in case of several critical components and showed significant real-time component execution response limitations. The most affected component's response of the case study architecture was in the critical group of sound processing and conversion peripheral servicing components. In case of the best-effort method, use of only Layer 2 memory type was requisite together with the dynamic resources sharing mechanism which resulted in increase of necessary real-time response margin and memory size margin to ensure safe and correct component execution. The component real-time response has therefore risen to the series of several tenths to hundreds milliseconds which was unacceptable from the psychoacoustic sound delay perception point of view. Due to the indicated properties the resulted architecture was rendered non-functional and it was necessary to propose a method which appropriately combined the two discussed design particularities.

The described hybrid embedded system design method successfully reduced the required DSP processor count to one and was furthermore capable of effectively using all provided hardware and memory resources such that all executed components achieved real-time execution response within specified input requirements. Thanks to the component distribution into three distinctive groups, the implementation was well optimized with regards to various component computation resources and real-time response requirements. The critical component group achieved real-time response in series of hundreds of microseconds. The hybrid component group achieved real-time response in series of ones of milliseconds to tenths of milliseconds (depending on each component specific requirement). The non-critical component group due to its negligible real-time response requirement was left non-optimized. The designed case study architecture further provided an increase in execution safety for the critical component group for cases when less critical component failed to execute properly by locking its execution in favor of the critical component group.

The designed case study architecture consists of the following layers (as depicted on Fig.5):

- Application domain within the architecture layer model is considered the external environment. The embedded system itself is usually designed to work directly with the application domain values. Our case study application domain was the sound signal since the application function is to process and modify the sound properties based on the user requirements. Another application domain environment to consider was the user controls employed to modify the model parameters in real-time.
- Direct link with the application domain layer is found on the embedded system layer which is one level above the mentioned application domain layer within the layer model of the hybrid method architecture. The embedded system is designed in regards with required real-time component execution response in a way that the overall architecture functions as effectively and reliably as possible. The components were therefore distributed based on the hybrid method into three real-time response-related groups:
- Critical components – components of the sound conversion peripherals, components of the sound communication peripherals – these required 100% rate of real-time response-correct execution due to the necessity of fully uninterrupted sound signal flow.
- Hybrid components – components of the dynamic resources sharing for computation and memory non-critical component resources, components of communication peripherals, component of auxiliary digital signal processor management.
- Non-critical components – component of local front panel control and display.
- The knowledge base layer in case study was used for user's model design optimization and for the attribute control. The knowledge base layer was deployed within the embedded system together with the user model interpretation. Several qualitative sound signal parameters were gathered and analyzed in real-time within the embedded system. Based on these results, the design optimization rules and knowledge were dynamically defined providing help in user's model real-time design process. Such knowledge base layer implementation facilitated and expedited modeling process and enhanced safety and integrity of the obtained user model by means of employing special rules that allowed only appropriate combinations of components depending on defined particular sound properties.
- The top layer within the hybrid method architecture model represents the modeling tools for creating the overall system model and implementation. The case study architecture top layer proposes using a modeling environment for real-time dynamic user model design which is interpreted at the same time in the embedded system. Further research is already on-going in the area of employing object-oriented modeling and generation tools such as modeling languages (UML) and model description standards (XMI) [25], [26] which are to be used to simplify portability and re-use of the model.

VI. Conclusion

This paper presents embedded systems design methods oriented on input constraints analysis, particularly on analysis and optimization of the rate of real-time response-correct component execution. The core idea of the suggested hybrid design method is the separation of components into three distinctive groups (critical, hybrid, non-critical). Memory model is selected and optimized as well as computation resources model is selected based on the mentioned component grouping. Overall advantage of the described method is the capability of achieving optimized architecture model from the viewpoint of required real-time response, use of available implementation platform resource which is hardly achievable when employing either only dependable or the best-effort embedded system design method.

Acknowledgment

This work was supported by the Slovak Research and Development Agency under the contract No. APVV-0008-10 and by the Scientific Grant Agency of the Ministry of Education - VEGA Grant No. 1/0531/12 Use of Knowledge in Software Processes.

References

[1] S. R. Ball, "Embedded microprocessor systems – Real World Design, " 3rd ed. Burlington, USA: *Elsevier Science*, 2002, ch.1

[2] T. A. Henzinger and J. Sifakis, “The Embedded Systems Design Challenge,” in *Proc. 14th International Symposium on Formal Methods*, Hamilton, CA, 2006, pp.1-15.

[3] O. Železník and Z. Havlice, “Software Architectures for Real-Time Embedded Applications for broadcasting,” in *Proc. 10th International Conference on Information System Implementation and Modeling ISIM'07*, Hradec nad Moravici, Czech republic, 2007, pp. 63-70.

[4] P. S. Roop, Z. Salcic, M. Biglari-Abhari, and A. Bigdeli, “A new reactive processor with architectural support for control dominated embedded systems,” in *Proc. 16th International Conference on VLSI Design VLSID’03*, Washington, USA, 2003, pp. 189 – 194.

[5] L. Carro, F. Wagner, M. Kreutz, and M. Oyamada, “A Design Methodology For Embedded Systems Based On Multiple Processors,” in *Proc. IFIP International Workshop on Distributed and Parallel Embedded Systems,* Deventer, Netherlands, 2000, pp. 33-42.

[6] T. Henzinger and J. Sifakis, “The Discipline of Embedded Systems Design,” *IEEE Trans. Computer,* vol. 40, no. 10, pp. 32-40, Oct. 2007.

[7] *SystemC 2.2, Core SystemC Language and Examples,* Open SystemC Initiative, 2011. Available: http://www.systemc.org/downloads/standards/

[8] *UML 2.0 OCL specification*, Object Management Group, 2003. Available: http://www.omg.org/ocl

[9] J. Schmuller, *Mysl ́me v jazyku UML*, Prague, Czech republic: Grada Publishing, 2001, ch. 1.

[10] D. Bell. (2003). UML basics: An introduction to the Unified Modeling Language. *IBM Global Services, Rationale Software*. [Online]. Available: https://www.ibm.com/developerworks/rational/library/769.html

[11] M. Fowler, *UML Distilled: A Brief Guide to the Standard Object Modeling Language*, 3rd ed. Boston, USA: Addison-Wesley Professional, 2004, ch.1.

[12] F. Balarin et al., “Metropolis: An Integrated Electronic System Design Environment,” *IEEE Trans. Computer,* vol. 36, no. 4, pp. 45-52, Apr. 2003.

[13] J. Eker et al., “Taming Heterogeneity - The Ptolemy Approach,” in *Proc. IEEE*, vol.91, no. 1, pp. 127-144, Jan. 2003.

[14] K. Balasubramanian et al., “Developing Applications Using Model-Driven Design Environments,” *IEEE Trans. Computer*, vol. 39, no. 2, pp. 33-40, Feb. 2006.

[15] J. Sifakis, “A Framework for Component-Based Construction,” in *Proc. Soft. Eng. And Formal Methods SEFM’05,* Koblenz, Germany, 2005, pp. 293-300.

[16] B. P. Lathi, *Signal Processing and Linear Systems*, Carmichael, USA: Berkeley-Cambrige Press, 1998, ch. 2.

[17] O. Železník and Z. Havlice, “MDA Approach in Embedded Systems with Strict Real-Time Response and On-the-fly Modelling Requirements,” *Trans. Acta Electrotechnica et Informatica*, vol. 9, no. 4, pp. 30-36, 2009.

[18] M. Ko, Ch. Shen, and S. Bhattacharaya, “Memory-constrained Block Processing Optimization for Synthesis of DSP Software,” in *Proc. Embedded Computer Systems: Architectures, Modeling and Simulation, IC-SAMOS’06,* Samos, 2006, pp. 137–143.

[19] R. Hametner, A. Zoitl, and M. Semo, “Automation Component Architecture for the Efficient Development of Industrial Automation Systems,” in *Proc. 6th annual IEEE Conference on Automation Science and Engineering*, Toronto, CA, 2010, pp.156-161.

[20] H. Kopetz, *Real-Time Systems: Design Principles for Distributed Embedded Applications*, Norwell, USA: Kluwer Academic Publishers, 1997.

[21] H. Kopetz and G. Bauer, “The Time-Triggered Architecture,” in *Proc. IEEE*, vol.91, p. 112-126, Jan. 2003.

[22] O. Železník and Z. Havlice, “On-the-fly MDA application modelling using Executable and Translatable UML,” presented at the Model Driven Software Engineering Workshop on Transformations and Tools MDSE 2008, Berlin, Germany, 11.-12.12.2008.

[23] *Analog Devices ADSP-BF533 Blackfin Processor Hardware Reference*, Analog Devices Inc., Norwood, USA, 2003.

[24] B. Gold and N. Morgan, *Speech and Audio Signal Processing: Processing and Perception of Speech and Music*, New York, USA: John Wiley and Sons., 1999.

[25] *UML profile for modeling Quality of Service and Fault Tolerance Characteristics And Mechanisms*, Object Management Group, 2004. Available: http://www.omg.org/spec/QFTP/

[26] *XML Metadata Interchange (XMI) – Formally Released Version of XMI*, Object Management Group, 2011. Available: http://www.omg.org/spec/XMI/, 2011

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 modeling 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.