Real time system modeling and analysis

E. Kazanavičius, A. Liutkevičius, V. Dusevičius, V. Kazanavičius

Digital Signal Processing Laboratory, Kaunas University of Technology, Lithuania

E-mail: ekaza@dsplab.ktu.lt

Introduction

The employment of various real time systems have been spread widely last few years. These systems are used in industry, telecommunication networks, ultrasonics and other domains. A Real Time System (RTS) is a digital signal processing (DSP) system which accuracy depends not only on the output, but also the time at which the output is produced.

Embedded systems are very important group of RTS. An Embedded DSP system consists of hardware and software which forms a component of some larger system and which is expected to function without human intervention. A typical embedded system consists of a single-board microcomputer with a software in ROM, which starts running some special purpose application program as soon as it is turned on and will not stop until it is turned off. Embedded real time systems are widely used in mobile phones, domestic appliances, production process control etc. This is very large, rapidly growing application area.

DSP system development process is difficult, involving several stages. This process includes not only software, but hardware development as well. The main stages of the process are hardware and software design, implementation, testing, integration and maintenance [14].

Delivery time to a client is the main criterion for the RTS developer. According to research, an embedded system is out of date after 6 months [17] since it was began to develop. After this period, a DSP system needs to be redesigned to meet the new requirements.

The DSP systems cost is another important criterion as well. The real time system modeling is one of the strategies to reduce the developing time and the cost of the developed system. For example, modeling can be used instead of expensive prototype development. The RTS simulation can be performed by computer aid to test the behavior of the system. Such a method substantially reduces RTS development process cost and time: there is no need to develop expensive prototypes.

MATLAB, SIMULINK, STATEFLOW, DYMOLA and PTOLEMY II are widely used modeling and simulation tools. MATLAB [6], SIMULINK and STATEFLOW are products of Mathworks Inc., which are used for design and modeling of various computer systems. These tools are not initially designed for RTS, but can be used in this domain too. DYMOLA is a modeling tool based on MODELICA modeling language [10]. It enables the analysis of complex systems that incorporate mechanical, hydraulic, electrical, thermal components and control systems. PTOLEMY II is an open project of Berkeley University of California [13]. The embedded systems modeling tool is created during project period. This tool is initially dedicated for development of RTS.

The mentioned modeling systems solve RTS development and modeling problem in some ways. But these tools have several drawbacks. Some of them are mentioned below:

- Restricted modeling possibilities: MATLAB can be used mainly for mathematical modeling, but is not suitable for development of other system models creation. SIMULINK, STATEFLOW and DYMOLA can be used for functional and behaviour modeling only.

- The need to have the additional knowledge about the tool: MATLAB requires the knowledge of a special language; the user introduces the mathematical model of the time on his own, so there is possibility for errors.

- There is no code generation or it is not effective: MATLAB, SIMULINK and STATEFLOW models can be used for the C code generation, but such a code can not be directly ported to specific DSP architecture; DYMOLA can not be used for code generation; PTOLEMY II code generation is in a development stage.

- Modeling can not be performed in a real work space with real input signals: no one of mentioned tools can perform modeling process in a real working environment (potentially PTOLEMY II can be used for this, because it is written in JAVA). Modeling is performed with randomly generated data.

The component based modeling technique presented in this paper solves most of these problems.

Methodology

The component based modeling technique is presented in this paper. This technique uses two technologies: JAVA and XML. JAVA language is selected according to its features:

- Object Oriented;
- Portable;
- JAVA components can be reused;
- The possibility for faster developing (compare with non object oriented languages);
- Popularity: JAVA language is widely used in embedded systems and becomes so popular like C language is;
- Real Time JAVA specification is created: there are a number of RT JAVA versions under development, which could be used for RTS.

JAVA language [2] is very attractive for RTS developers. It accelerates the development process and what is most important it is portable. This means that JAVA programs can be executed on different hardware

platforms without any code changes. The presence of JAVA virtual machine is the only requirement for the selected platform. There are few versions of JAVA language, which are designed for embedded systems [1], [9], [11], [15]. JAVA language is widely used in mobile applications.

Research shows [18] that applications written in JAVA have the same or even higher performance than written in C language. Therefore, JAVA can be recommended for RTS development.

The object oriented technique for RTS development and modeling is presented in this article. This technique includes constructing models of RTS in three levels: functional, behavioural and architectural [17]. The modeling software "KTU_RTSM" developed by our laboratory is based on the proposed modeling technique (see Fig.1).

RTS models are specified using the JAVA and XML languages. Hence, these models can be used as a base for a code generation. Besides, the modeling process can be performed on a specific embedded platform, which has JAVA VM. Such a modeling technique covers the major part of RTS development process.

Component based RTS model

The JAVA and XML are specification languages for the RTS. The JAVA language object orientation [2] is important while constructing RTS models. Each RTS and a computer system consists of various components. These components can be software and hardware of some other kind (e.g. system users). The component performs particular functions and communicates with other components. We propose the component based RTS modeling technique when RTS model is created specifying system components and links between them. The particular component is an object which has its own attributes (parameters) and set of methods, which describe the components behaviour. Objects can be complex, i.e. they can aggregate other objects (components). The modeling process is an interaction between RTS components and observation of that interaction. The JAVA programming language ideally suits for such model creation. It operates with objects and has such definitions as inheritance and aggregation. RTS component performs specific functions, so the component based RTS model creation is functional modeling [17].

RTS can be considered as a complex component which has set of inputs, set of outputs and specific function. Hence, the component based RTS modeling can be performed on desired abstraction level. It can be performed on atomic components level, on complex components level or on the highest system level.

The important feature of JAVA programming language is its compatibility with XML technology [3], [5]. The XML meta-language was selected to specify the structure of modeled RTS components. This technology is very popular among web developers, but can be used in any domain due to such features:

- XML can be used for specifying data of any complexity, amount or type.
- XML data can be read and understood easily.
- It is easy to create XML parsers.
- XML data can be created easily.
- This is a modern technology, which ensures the versatility, maintainability and compatibility of the developed software.

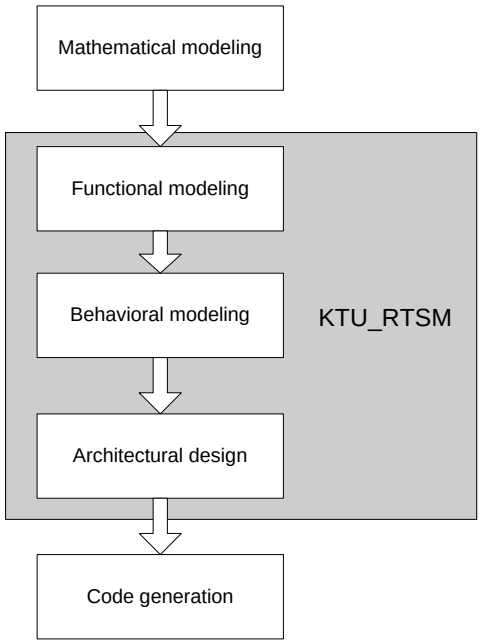

Fig. 1. **KTU_RTSM software functionality**

The JAVA language is used to specify the functions (actions) of particular RTS component while XML is used to specify the internal structure of a component. Such a solution enables to specify new components reusing existing JAVA specifications. In that case it is sufficient to create XML file describing the new component. Very high abstraction level of RTS component is the main feature of XML usage: all components are of the same type with different functions and the internal architecture described in XML files.

RTS component specifying using XML. RTS model components are specified using XML data files. This solution is convenient, because new components can be created or existing components can be modified without any JAVA code changes. Besides, XML description can be easily extended with new features. The main feature of XML usage is possibility to create the desired complexity and abstraction level components without new JAVA object type (class) creations. The following information is described in components XML file:

- Component name – it is the main component identifier. The name is unique: two components with same name can not exist.

- Menu path – this information is used by modeling environment. The path shows where component menu needs to be placed.

- Inputs and outputs quantity – it is the initial number of inputs and outputs. These values can be changed dynamically depending on a particular component.

- Parameters quantity – shows how much parameters (attributes) component has. This value can vary from zero to infinitive number of parameters.

- Parameter name, index and initial value – this information describes the main features of each parameter. The parameter name is used only for XML clarity. The main parameter identifier is an index. The value of a particular parameter can be changed dynamically.

- Component actions (run actions) – it is set of actions (run_actions block), which ensures the functionality of the component. The sequence of actions and JAVA class names, which are used to perform that actions (run_action file) are described in XML file. Component actions operate with data from component inputs or with component parameters.

- Component popup menu actions – this information is used by a modeling environment. It includes the menu action name, JAVA class name, which performs this action, and the parameter index, which is modified by this action.

The XML file example is presented in Fig.2.

As it is mentioned above, the XML technology prevents us from creation of new JAVA class for every new type of component. The XML file is parsed and particular RTS component instance is created according to XML specification.

Simple (atomic) and complex components. Simple or atomic RTS components are the lowest abstraction level of components, which perform basic data processing operations. These components are considered as undivided and seamless. *SimpleComponent* type is used to describe simple components of RTS.

It is a main component describing the class, which involves a graphical component information (used by a modeling environment), component parameters and component actions.

The functionality of a particular component depends on set of its actions. An action is an object, which performs basic manipulations with the component input data or parameters. The component has unlimited number of actions. All classes, which describe actions, are the subclasses of *SimpleComponentAction* class. Hence, a very high abstraction level is achieved.

Actions are not dependent on a particular component – the same action can be used by different components. For example, actions *GetInputs* and *SetOutputs* are used by all components, which have at least one input and output.

Generally speaking, the set of component actions describes its function. The total impact on component data performed by component actions is its function *h(t)*:

$$y(t) = h(x(t)) \qquad (1)$$

where *y(t)* is the component output signal; *x(t)* is the component input signal; *h(t)* is the transformation.

Every component has the set of parameters. Values of these parameters can be changed using the component popup menu. The number of parameters, initial values and the popup menus are described in the XML file.

The high level of abstraction allows creation of RTS components of any complexity and any domain. New components are created by implementing new action classes (or using existing) and creating a new XML file. Hence, the advantage of objects reusing is taken in this case. Reuse is the main feature of object oriented languages, which allows systems development in a more efficient way.

```
<?xml version="1.0" encoding="UTF-8" ?>

<component name="constant">

  <menu_path path="generators"/>

  <inputs quantity="0"/>
  <outputs quantity="1"/>
  <parameters quantity="1">
    <parameter name="constant value"
               index="0" value="1.0"/>
  </parameters>
  <run_actions>
    <run_action file="Constant"/>
    <run_action file="SetOutputs"/>
  </run_actions>
  <menu_actions>
   <menu_action name="Set constant value"
                file="SetDoubleValue"
                param_index="0"/>
  </menu_actions>
</component>
```

Fig. 2**. RTS component XML file**

Complex components are those, which aggregate some part of RTS scheme (model), having the set of inputs and outputs. Any RTS scheme can be saved as a complex component. A complex component is created by the user, which defines the inputs and outputs of the complex component (a complex component can have no inputs or outputs), the component name and menu path. The functionality of the complex component is the same as functionality of a scheme, which is aggregated by the complex component. The complex component can aggregate both simple and complex components. Hence, the complex component is a hierarchical structure with unlimited size and complexity.

This solution makes RTS modeling easier. There is no need to create a big and complex RTS scheme with a lot of components. The scheme is created gradually from complex components, which aggregate the model parts created before. Besides, such a solution allows user to create the desired functionality components reusing existing ones. According to three level model, such a process is an architectural design [17], when algorithms are distributed into hardware.

A complex component is an instance of *ComplexComponent* class. This class inherits *SimpleComponent* class. Hence, the complex component can be considered as a simple component, which additionally has both aggregated components set and links between them set. The complex component has all features of simple component.

Links between components. As mentioned above, each RTS consists of components and links between them. *Link* class instance is used to define the link. The attributes of this class defines the link between one component output and other component input. These attributes are given below:

- The ID of the first component (*obj1_id*) – this is the first component sequence number in a scheme (model).

- The ID of the second component (*obj2_id*) – this is the second component sequence number in the scheme (model).
- The first component output index (*obj1_outIndex*).
- The second component input index (*obj2_inIndex*).

The link has only one direction. It can connect only the output with input, but is impossible between two inputs or outputs. The *Link* object is not a medium for data transfers between components: this object is a rule, which defines how data must be transferred between components. This means that the link does not transfers data from one component to an other, but shows to modeling environment how it must be done.

Data types. One of the tasks was to create an universal communication method between components, which allows using any type of data. The common *Object* class is used for such a purpose. Components operate with *Object* type data, because in JAVA any data can be considered as an object. The interpretation of the processed data is done depending on a particular component type. It means, that a component must convert (cast) the input data into such a type, which it can process. Hence, a component does not know the semantics of the input data and interprets this data as requires the component function.

The behaviour modeling using the component based RTS model. When the RTS model is created using components and links between them, it is possible to perform behaviour modeling of the designed system [17]. This modeling allows observation of the behavior of the RTS during its lifecycle.

A data flow model is a popular way to describe the systems behavior [7]. According to this model, each RTS system can be described as an oriented graph, where nodes (RTS components) perform some data processing and edges correspond to the transfer medium between nodes. A few data flow types exist. The synchronous data flow (SDF) model is one of the best ways to describe DSP systems. The synchronous DSP system is one in which all of the sampling rates in the system are rationally related. The SDF is ideally suited to a large set of DSP applications such as digital communication (QAM, PSK, CDMA) and filtering applications (wavelets, filterbanks, IIR, FIR) [12].

The component based RTS model can be named as SDF [7]. RTS components correspond to SDF nodes and links between components correspond to SDF edges. Components activation schedule is determined once before modeling process. But there is a difference between traditional SDF and the presented RTS model. In the proposed model the schedule is generated randomly, not taking into account the links between nodes. The proper functionality of the model is guaranteed by the internal structure of a component. The component processes input data not directly getting it from other components, but uses buffering. The modeling process is performed iteratively. Input data are fetched into a data buffer during one modeling process iteration. After getting data, all components perform data processing (the current data which are in the buffer) and after this the next iteration is performed. Some components operate with fictive data during first modeling process iterations, but when the system model becomes stable, correct results are obtained. The time period while a system is unstable depends on the size and the level of hierarchy (aggregation) of RTS model. The modeling process is performed using separate *RunSchemeThread* thread. This thread manages the data flow between components by sending data from one component to an other.

A particular component fully performs data processing operations during a single iteration of the modeling process. But in a real system the data processing can last more than a single iteration. The easiest way to construct such types of RTS models is to add delay components to the model. The delay component acts as FIFO (*first in first out*) type memory. Hence, in order to model the situation, when some component performs data processing more than single iteration, this component output needs to be connected to a delay element. In this case, the data processing results will be obtained after *N* iterations at the delay component output. Analogically, more complex components can be created for different usage, e.g. components for synchronization or other types.

The Full RTS model is obtained by introducing time into the model. Some parts of RTS or input data can be analog. Besides, some RTS components can act periodically, depending on time. Hence, the time is the additional parameter, which is used during the modeling process. SDF with the time parameter is called Timed Synchronous Data Flow (TSDF) [12]. The presented component based RTS model has time parameters as well. It is considered, that a single modeling iteration lasts the defined period of time. Hence, RTS components, functions of which depend on the time can periodically perform their actions depending on the pasted time period. According to this, the presented component based RTS model is full and can be used to describe RTS of any purpose and domain.

Modeling of a real time ultrasonic (UT) waveform generator

The UT waveform generator plays an essential role in testing and development of real time NDT systems. The generator can be implemented as a software component or a separate hardware component. Hardware real time generators can be used as replacement to physical environment, thus accelerating overall RT system test and development process and eliminating the need of UT measurement equipment. In this section we will create hard real time UT waveform generator model using the proposed component based RTS modeling technique.

The generator generates UT waveform which consists of a reference echo, reflected by a water-sample A interface and one echo reflected by the sample A-sample B interface as shown in diagram the below (Fig. 3).

The output waveform *y(t)* is generated according to the reference signal *ref(t)* which is the reflection from water-sample A interface. The reflection from the interface sample A-sample B *echo(t)* is obtained in the frequency domain using the frequency dependant reflection coefficients $R_{AB}(\omega)$:

$$echo(t) = FFT^{-1}[FFT[ref(t)] * R_{AB}(\omega)], \qquad (2)$$

where FFT^{-1} is the inverse discrete Fourier transform; *FFT* is the discrete Fourier transform.

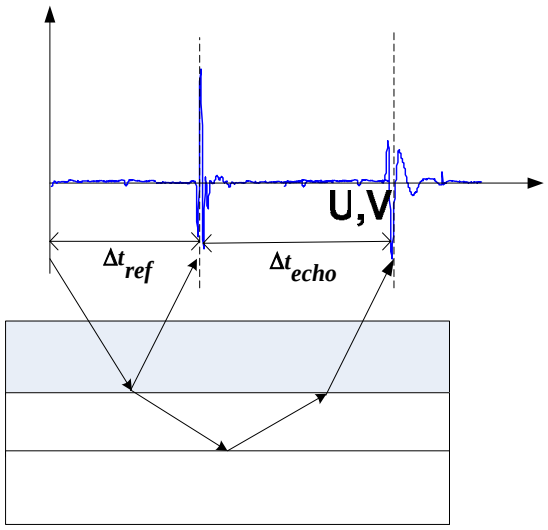

Fig. 3. **Lattice diagram of test setup**

If the time of flight of the reference signal is Δt_{ref} and of the first echo - Δt_{echo}, then the generator output can be written as:

$$y(t) = ref(t - \Delta t_{ref}) + echo(t - \Delta t_{echo}) \qquad (3)$$

Using the proposed modeling technique, UT waveform generator components were described with XML and JAVA languages. The scheme was composed from the described components and is shown in Fig. 4.

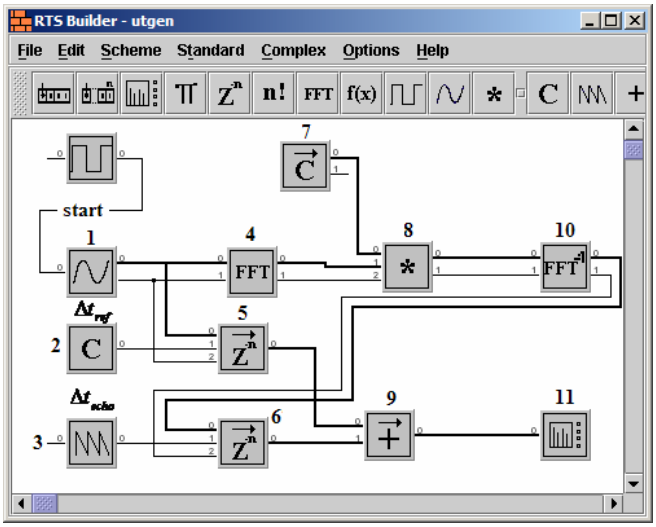

Fig. 4. **UT wave generator model**

The generation of one waveform is initiated by a periodic start signal. The reference signal generation component 1 prepares N samples of the reference signal *ref(t)* and puts them into the internal buffer. After that the discrete FFT is performed by the component 4. The component 8 performs multiplication of the calculated FFT by the preloaded reflection coefficients (from the component 7). After that the inverse discrete FFT is performed on multiplication results by the component 10. The inverse discrete FFT result is the waveform of the sample A-sample B echo, which has to be delayed by Δt_{echo}. The delay function is performed by the component 6 and the delay Δt_{echo} is generated by the component 3 according to the geometry of the specimen.

Similarly, the reference signal is delayed by the value Δt_{ref} which is given by the component 2 and the delay is performed by the component 5.

The delayed reference and echo are summed by the component 9 and visualized using an oscilloscope – the component 11.

In this model we used the frequency dependant reflection coefficients $R_{AB}(\omega)$, measured using the 5MHz 0.75" probe and 100MHz 12-bit ADC:

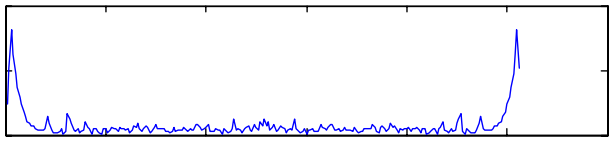

Fig. 5. **Interface A-B reflection coefficients**

During the experiment with the generator, we fed the real experimentally measured reference signal to the generator input (Fig. 6):

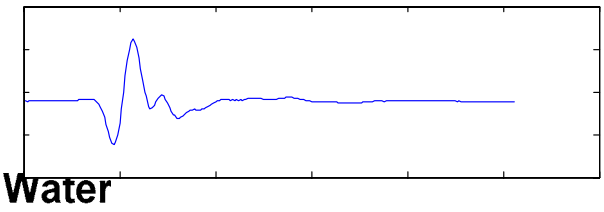

Fig. 6. **The reference signal at generator input**

The generated output echo signal is shown in Fig. 7.

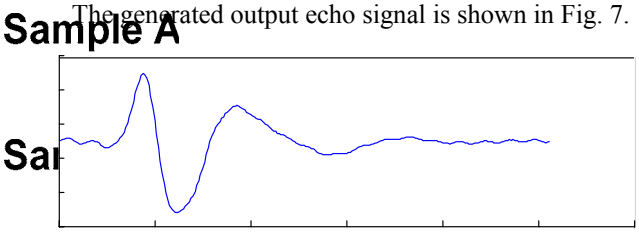

Fig. 7. **The generated signal at generator output**

Experimentally measured in a physical environment (dashed) and simulated (solid) results are shown in Fig. 8.

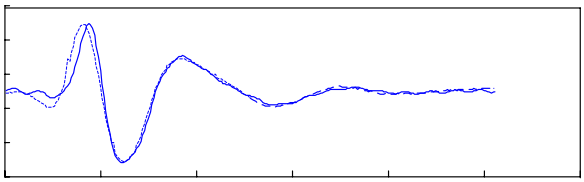

Fig. 8**. Simulated and measured signals together**

The modeled results are very close to the measured. Hence, the experiment proves that the component based modeling technique is correct and can be applied for RTS modeling.

Conclusions

The object oriented technique for RTS development and modeling is presented in this paper. The modeling process includes functional modeling, behavioural modeling and architectural design. The RTS modeling is performed by creating the component based model of RTS. RTS is considered as a complex hierarchical structure, which is composed of various subsystems, called components. The RTS component is specified using JAVA programming language and the internal structure of the component is specified using XML meta-data. The RTS architectural model is created using components and links between them. Components can be complex, i.e. can aggregate other components. Such a solution guarantees the desired abstraction level of the model and allows user to create convenient components by reusing existing components. Since RTS components are described using the JAVA language, the component based model can be used as a basis for a code generation.

The RTS modeling software was created using the proposed modeling technique. This software is suitable to create RTS models of desired complexity and abstraction level. The behaviour of RTS models can be observed as well. Since component has a high abstraction level, this modeling system can be used not only for RTS modeling. The implemented modeling system is suitable for modeling of any type of real world systems, which can be described using mathematical equations. The experiments prove that the proposed modeling technique is correct and modeling results are the same as theoretical.

References

1. **Comp L**. Runtime Abstractions in the Wireless and Handheld Space / L. Comp, T. Dobbing. Intel Technology Journal., 2003.Vol. 7. Issue 1.
2. **Eckel B.** Thinking in Java. 2nd Edition. Prentice Hall. 2000. 1200 p. ISBN 0-13-027363-5.
3. **Hall M.** Core Web Programming, 2nd Edition / M. Hall, L. Brown. Sunsoft Press. 2001. 1440 p. ISBN 0-13-089793-0.
4. **Harold E. R.** XML 1.1 Bible, 3rd Edition. Willey Publishing Inc., 2004. 1022 p. ISBN 0-7645-4986-3.
5. **Harold E. R.** Processing XML with Java. Addison-Wesley, 2002. 1100 p. ISBN 0201771861.
6. **Hunt B. R.** Guide to MATLAB: For Beginners and Experienced Users / B. R. Hunt, J. Rosenberg, R. L. Lipsman. Cambridge University Press. 2001. 348 p. ISBN 052100859X.
7. **Lee E. A.** Synchronous data flow / E. A. Lee, D. G. Messerschmitt. Proceedings of the IEEE. 1987. Vol. 75. No. 9. P. 1235-1245.
8. **Marven C.** A simple approach to digital signal processing /C. Marven, G. Ewers. A Wiley-Interscience publication. 1996. 248 p. ISBN 0-471-15243-9.
9. **Moertiyoso N.** Designing Wireless Enterprise Applications on Mobile Devices /N. Moertiyoso, K. Choong Yow // First Iinternational Conference On Information Technology & Applications (ICITA 2002). Bathurst, Australia. 2002. ICITA2002 ISBN 1-86467-114-9.
10. **Otter M.** Hybrid Modeling in Modelica Based on the Synchronous Data Flow Principle / M. Otter, H. Elmqvist, S. E. Mattsson. 10th IEEE International Symposium on Computer Aided Control System Design (jointly with the 1999 Conference on Control Applications). Hawaii, USA. 1999. ISBN 0-7803-5449-4.
11. **Paal P.** Java 2 Platform Micro Edition [online]. 2001, [last visited 2004-05-16]. Address:http://www.hut.fi/~opaal/netsec/j2me.pdf.
12. **Pino J. Z.** Cosimulating Synchronous DSP Applications with Analog RF Circuits / J.Z.Pino, K.Kalbasi // 32nd Asilomar Conference on Signals, Systems and Computers. Monterey, California. 1998.
13. Ptolemy II: Heterogeneous Concurrent Modeling and Design in Java [online] / J. Davis, M. Goel, C. Hylands, et. al. Memorandum UCB/ERL M99/44, EECS. University of California, Berkeley, CA, USA. 1999. [last visited 2004-05-16]. Address: http://ptolemy.eecs.berkeley.edu/publications/papers/99/HMAD/
14. **Sommerville I.** Software Engineering, 6th Edition. Addison-Wesley. 2001. 677 p. ISBN 0-201-39815-X.
15. Sun Microsystems, Inc. J2ME Building Blocks for Mobile Devices [online]. [last visited 2004-05-16]. Address: http://java.sun.com/products/cldc/wp/KVMwp.pdf
16. World Wide Web Consortium. Extensible Markup Language (XML) 1.0 (Third Edition) [online]. 2004, [last visited 2004-05-16]. Address: http://www.w3.org/TR/REC-xml/
17. **Kazanavičius E.** Signalų apdorojimo sistemos. Technologija, 2004. 196 p. ISBN 9955-09-639-X.
18. Java and C Performance Evaluation for DSP Applications / E. Kazanavičius, V. Dusevičius, A. Liutkevičius, R. Žukaitis. Conference at GSPx 2004, Embedded Applications: Software & Hardware. Santa Clara, CA USA. Sept. 27-30, 2004.

E. Kazanavičius, A. Liutkevičius, V. Dusevičius, V. Kazanavičius

Realaus laiko sistemų modeliavimas ir analizė

Reziumė

Pateiktas realaus laiko sistemų objektiškai orientuotas komponentinis modelis. Jis sudarytas naudojant JAVA ir XML kalbas realaus laiko sistemų komponentams specifikuoti. Modelis ir jo pagrindu sudaryta sistema naudojami įterptinių sistemų programoms sintetinti tiesiogiai iš specifikacijos. Sistema analizuojama ir modeliuojama komponentiniais modeliais, tai leidžia aprašyti bet kokias sistemas ir bet kokio abstrakcijos lygio. Atlikti realaus laiko ultragarsinių signalų generatoriaus modeliavimo eksperimentai.

Pateikta spaudai 2004 05 31