Proc. Estonian Acad. Sci. Eng., 2005, **11**, 1, 18–30

Challenges for real-time systems engineering. Part 2: Towards time-aware technology

Leo Motus[a], Robertus A. Vingerhoeds[b] and Merik Meriste[c]

[a] Department of Computer Control, Tallinn University of Technology, Ehitajate tee 5, 19086 Tallinn, Estonia; leo.motus@dcc.ttu.ee
[b] Ecole Nationale d'Ingénieurs, Av. d'Azereix, 65016 Tarbes, France; eaai@wanadoo.fr
[c] Institute of Technology, University of Tartu, 50090 Tartu, Estonia; merik.meriste@ut.ee

Received 26 March 2004

Abstract. This part of the paper discusses evolution trends of the theory and technology in time-aware interaction-centred models of computation and in time-aware multiagent systems that foster the emergence of a multidisciplinary environment, capable to support analysis of design decisions at the early development stages of time-critical software-intensive systems.

Key words: real-time systems, time-critical systems, proactive components, interactive computing.

1. THEORETICAL AND TECHNOLOGICAL NEEDS OF SOFTWARE-INTENSIVE SYSTEMS

Software-intensive systems form the major part of contemporary computer applications. This class contains embedded and real-time systems, proactive systems, a remarkable part of interactive problem solvers and decision-support systems, etc. All such systems have pretty similar theoretical and technological needs and expectations that differ from those characterizing data processing and conventional information processing systems. In the first part of this paper [1] several application areas of software-intensive systems were surveyed, with the focus on new requirements to the theory and technology arising from those applications.

Based on this survey one can distinguish three sources of system designer's worries:

– increased complexity of systems with all its implications;
– incoherence of the applied theory of computation and the actual computation needed in those systems;

– cumulative approximation effects of a variety of theories and algorithms integrated into systems.

Some emerging methods that may reduce the worrying influence of the above-listed factors were also pointed out in [1].

Rapidly increasing complexity of systems can remarkably better be controlled if systems were built from autonomous and proactive components. Eventually this will lead the today wide-spread component-based systems to multi-agent systems. Another emerging method for coping with the complexity is to introduce time-awareness to the systems and to their components. This enables to substitute many complicated causal relations with appropriate time constraints that have approximately similar impact on system behaviour [2,3]. It is interesting to note that both of these methods have been thoroughly tested in the history of human society. Another reason for introducing time-awareness to systems, in addition to direct reduction of the complexity, is the potential need for time-selective communication in a system that contains autonomous and proactive components. Especially in the cases where a computer-based component has to communicate with a non-computer component of the system from the natural or artificial environment.

Incoherence between the available and required theory of computation has been detected since systems started to perform more demanding tasks, e.g. in safety- and time-critical applications, besides becoming more and more complex. The conventional belief of a computer scientist that holds in conventional applications – correct prescriptive description fully determines the system behaviour – does not hold in the majority of software-intensive applications. Many researchers have been working intensively to cope with the emerging behaviour that is generated dynamically during the operation of new systems [2,4–7]. In spite of many interesting theoretical results, the commercially available tools for system development are still not quite coherent with the required theoretical basis. It means that on the average the theory for new models of computation is not mature and is at the moment lagging behind the system building practice. For instance, think about the long history of practical time-constraint stream processing and the not too advanced status of the corresponding theory, or about many empirical (theoretically not thoroughly studied) issues in the foundations of the UML.

Theoretically and practically the least advanced domain is related to *systematic study of phenomena occurring at systems integration from components, e.g. impact of the cumulative effects of approximations in various components on the systems behaviour*, and joint effects of multiple collaborating theories. For instance, the conventional theory for proving stability of a control algorithm is only of limited use when the algorithm is implemented on a multiprogrammed computer, because the time unit as implemented in a computer is of random length and therefore is inconsistent with the assumptions of the conventional stability theory. Another sample problem from this domain is related to the effects of small incoherence in the ontology of interacting components of a system. Ontology is usually considered as a static notion. In order to make any two components to collaborate, they should have a common ontology. If not, then at least one of the components should be able

to modify its ontology in order to understand its partner. In proactive component-based software-intensive systems one actually needs the notion of interactive ontology that is not sufficiently studied yet. Interactive ontology is related to capabilities for adapting to, or learning from perceived experience.

From the practitioner's point of view it would be ideal to have a unified system development framework that caters for the above listed worries, and integrates tools for handling all the different involved theories and available practical knowledge into an easily usable design environment. Putting together the evolution trends that exist today – e.g. those supported by the Object Management Group (OMG) consortium (http://www.omg.org), Foundation for Intelligent Physical Agents (FIPA, http://www.fipa.org), and many individual researchers – the outlines of such an ideal framework become predictable. The development of systems starts from preliminary system engineering study that continues with elaboration of the preliminary concepts in a model space. Today a realistic alternative is to apply MDA-based technologies [8] as UML and its profiles (e.g. real-time UML and Agent UML), or to use the concept of UML model processors (that semi-automatically enables to apply different theories for formal analysis of different aspects of the design [9]). Run-time monitoring and diagnosis-related functions are to be added to the future system during its design stage, in addition to periodic formal analysis of the design by using model processors. Run-time monitoring and diagnosis has an important role in keeping the emergent behaviour within the acceptable limits. The completed design is then transformed into code by code generators (or in more sophisticated cases by agent composers, synthesizers, etc.), verified and tested.

The rest of this paper focuses on surveying the evolution of new models of computation (that form the basis for the elaboration of methods potentially used in new UML model processors), development and application of cognitive engineering methods and elaboration of methods for building and analysing time-aware multi-agent systems.

2. TIME-AWARE INTERACTION-CENTRED MODELS OF COMPUTATION

A short introduction to the research of interaction-centred models of computation was given in the first part of this paper. This part focuses on the survey of the evolution of time-aware concepts in modelling interaction-centred computation. In other words, this is a survey of attempts to re-introduce the explicit notion of time into computer science with the final goal of getting better support to behavioural analysis of embedded real-time systems, time-aware multi-agent applications and the rest of software-intensive systems. Pragmatically, the designers are looking for models that possess the capabilities:

– to handle multiple data-streams simultaneously (act as a multi-stream interaction machine [4]);

– to capture time constraints imposed not only upon performance, activation and process execution deadlines, but also upon inter-process interactions and validity intervals of data and events;
– to reason about the coherence of the imposed time constraints and to verify that the system requirements, specification, design, and implementation satisfy the imposed time constraints.

Characteristic to the time-critical software-intensive systems is that they have to match phenomena that occur in different environments – in a computer, engine, chemical processes, mechatronic device, etc. Different time-counting systems may be used in each of those environments, whereas the computer has to be aware of all those time-counting systems. Introduction of proactive components increases pressure for multiple, simultaneous time-counting systems and time concepts [10,11]. This is the reason why time model to be used in software-intensive systems must be more sophisticated than the one used in conventional computer science.

Metric time quietly started to infiltrate into computing systems about three decades ago. Consider, for instance, timed Petri nets [12], a variety of temporal logics [13,14], and timed process algebras [15] that were introduced and mostly also successfully applied. For different reasons, any of the listed methods did not hold one or more of the above listed capabilities. For instance, Petri nets are not able to describe multi-stream processing since Petri nets have been proved to be equivalent to Turing machines. Besides, the used time model in timed Petri nets was too simplified to capture all the required time constraints. Temporal logics have two typical (and somewhat related) limitations:
– most of the introduced temporal logics have the expressive power that is equivalent or below that of the first-order predicate logic and therefore cannot, in principle, handle multi-stream processing that needs higher-order predicate logic [2,16,17];
– they cannot capture all the required time constraints (e.g. those imposed on inter-process interactions and on validity intervals of data and events) due to oversimplified time model used.

The application domain of timed-process algebras was limited mainly because of the oversimplified time model, and in some cases because of the applied dense (continuous) time presentation [15]. The use of continuous time is the dream of modellers who just describe systems, but could be a nightmare for synthesizers of new systems because of major approximations required when implementing the designs on digital computers, and also because of many potential paradoxes related to causal reasoning and detecting the quantitative order events.

Almost simultaneously with Milner's interaction-centred model of computation CCS [18], completely independently and with different purpose and different arguments Quirk and Gilbert published the first time-aware interaction-centred model of computation [5]. This model did not result from smooth evolution of conventional scheduling theory, temporal logic, timed process algebra related research, timed Petri nets, or any other timed theory of computation. It just stated that each component in a system has a right for its own time-counting system,

and this possibility should be considered when designing inter-component communication. The major innovative ideas introduced by Quirk and Gilbert were:

– a remarkably more sophisticated time model that enables each computing process to have its own time-counting system (and supports a multitude of time concepts, see for philosophical background [10]);
– explicit description of a non-terminating program with a set of repeatedly activated terminating programs and with explicit identification of the activation instants and time constraints; so far, conventional computer science considered non-terminating programs as not verifiable (there is no point in verifying properties of a non-terminating algorithm).

Pragmatically speaking, Quirk and Gilbert in [5] pointed to the stream-processing essence of a non-terminating program and also pointed to the necessity and to a method of verifying non-terminating programs. The inner structure of a non-terminating program can be represented by a set of interacting and countable number of times re-activated terminating programs [5,19], or by a set of persistent Turing machines [16], or by a multistream interaction machine [4]. Unlike persistent Turing machine and multistream interaction machine (as defined by Wegner) and interactive ASM [6,20] that are not aware of time, the alternative suggested in [5,19] has explicit time awareness. Obviously the search for time-aware interaction-centred model of computation should depart from the ideas of Quirk and Gilbert [5].

One of the key issues when ensuring time awareness of systems and their models is the selection of concepts for time models introduced to software-intensive systems. Time models as used in real-time systems were discussed in 1993 in [21] and did not attract wide attention. The major breakthrough in using sophisticated time models for building software systems was invoked by OMG decisions to fix a time model for RT CORBA and UML profile for scheduling, performance and time [9,22]. Similarities between OMG-adopted time models and those published in 1993 are discussed in [11].

Recent progress in the practice of time-aware interaction-centred models of computation is related to UML. It was demonstrated in [23] that UML model presents multi-stream interaction-centred computations. Before elaboration of UML profile for scheduling, performance and time, together with the concept of UML model processors [9], experiments were made to introduce time to pre-UML object-oriented models [24] and to test an approach that later became known as UML model processors.

3. EXAMPLES OF PRACTICAL APPLICATION OF NEW MODELS OF COMPUTATION

As discussed in [25], several ideas of time-aware interaction-centred models of computation have been put in place and experimentally tested in the project LIMITS. This project developed and evaluated the basic ideas of a new generation of real-time software engineering tools. LIMITS was based on a

highly innovative methodology (the Q-methodology [2]), whose capacity to model the real-time behaviour of software is mathematically proved – its description and analysis power is equivalent to a weak second-order predicate calculus [17]. As claimed independently in [16], the necessity for higher-order predicate calculus and the use of forced (true) concurrency is characteristic to interaction-centred models of computation. Sophisticated time model used as an essential part of the formalism makes it possible to handle, in principle, all the temporal aspects of industrial applications of information technologies, including timing analysis of interactions [26].

In order to avoid the common curse of formal methods – too high requirements on the mathematical knowledge of their users and bad scalability – the Q-methodology focuses on universal properties whose existence conditions can be proved beforehand. The LIMITS hides most of the formal aspects of the methodology from the system designer and in many cases the timing analysis is reduced to simple checking the suitability of system parameters or their combinations. Such an approach enables one to decompose the verification of the properties of a system into verification in the large (interactions of components, overall behaviour of the system) and verification in the small (specific properties of components that enable matching the ontology of components and systems). Verification in the large can be started at very early stages of system development, immediately after the user requirements and the preliminary system architecture have been fixed – even before the algorithms are finally selected. Such an early start of formal analysis improves economic feasibility of a project. As an additional value, LIMITS generates an executable prototype of the system that is based on the formally checked preliminary architecture and user requirements. The animation of this prototype improves the end-user's understanding of the operational properties of the future system and facilitates the end-user's participation in the software process.

Considering the rather narrowly dedicated character of LIMITS (timing correctness of interactions is its main goal), it should be used in combination with more generally oriented development tools. Therefore LIMITS was seamlessly integrated with, at that time upcoming, Object Modelling Technique [24]. Later on LIMITS was used in close cooperation with two different UML tools. This probably was the first experiment of semi-automatic navigating from an object-oriented design environment to a formal analysis environment, and returning with inserted corrections and modifications to the object-oriented environment to continue the development. Similar idea has been applied later [9] in the specification of UML model processors to navigate out of the UML environment for schedulability and performance analysis and to return with the modified model to the UML environment.

Tools like LIMITS become absolutely necessary for developing applications where temporal coherence of imposed time constraints and behavioural characteristics of the components is as important as the logical exactitude of the treatment of functional tasks. In other words, LIMITS-like tools can remarkably

reduce development cost and related risks of potential accidents (by setting implicit limits on the emergent behaviour) when developing safety- and time-critical applications. Some aspects of such an application are discussed in [27], where the analysis engine of LIMITS is used for run-time monitoring subsystem in an industrial diagnosis tool BRIDGE.

Behavioural and timing analysis of interactions in LIMITS takes place in three steps that can be iterated depending on the interim results of the analysis:

1) analysis of individual building elements of the system (processes and channels); all the values characterizing those elements (e.g. activation instants, execution duration, intervals for interactions, validity intervals for variable values, etc.) are given as interval estimates;
2) analysis of interacting pairs of processes (separate theorems have been proved for each type of channels that connect two processes); three basic types of interactions are separately analysed: synchronous, semi-synchronous and asynchronous;
3) analysis of the group behaviour of interacting processes or that of the whole system; analytical min-max performance estimates can be computed, potential static deadlocks caused by time-selective inter-process communication and by requirements to validity intervals of interchanged data are detected and achievability of required synchronization precision in clusters is checked.

The UML is *de facto* rapidly growing towards an international standard and is already standardized within the OMG consortium. However, a designer of software-intensive systems, working for industry, would need several improvements in UML-related methodologies and tools before they meet designer's pragmatic requirements. For instance, better support to considering location-related properties and to acquiring and storing information about temporal characteristics that are required for timing analysis of interactions is needed. The general-purpose UML approach and Agent UML profile, similarly to its profile on schedulability, performance and time, should be seamlessly integrated with formal analysing methodologies (e.g. by applying UML model processors).

The conclusion is that today real-time UML and MDA offer enough flexibility and mechanisms to successfully move towards application of interaction-based models of computation in industrial systems. A step is now needed to be made at automation, control theory, systems science and computer science level to take these ideas on board.

4. NEED FOR COGNITIVE ENGINEERING

The evolution of conventional system engineering, i.e. methodology and tools for building artificial systems from passive components, is more or less on track in spite of comparatively late acceptance of the time-sensitivity concept. At the same time the effect of proactive components on systems behaviour has not yet been seriously studied by the mainstream system engineering. A pretty strong research community that builds applications based on agents and multi-agent

systems exists. However, they tend to study systems, built entirely and only from proactive components (e.g. distributed artificial intelligence systems) and focus essentially on issues of artificial intelligence.

It is true that distributed AI and multi-agent systems provide good arguments for the interaction-centred model of computing. Still, majority of industrial applications of software-intensive systems tend to comprise passive, active, reactive and proactive components. Active and proactive components can be found in a computer system as well as in its natural and artificial environment including human operators and supervisors. Such a non-homogeneous component base of a system poses many new problems in its behaviour analysis that, so far, have been studied neither in multi-agent systems nor in common software-intensive technical applications. At the moment the integral effect of those problems is approximately covered by the term "emergent behaviour".

Emergent behaviour is caused, in addition to incomplete knowledge about causal relations, by adaptation, learning and self-structuring capabilities of the components. The latter capabilities are based on the perceptive behaviour of the system (or its components) that leads to self-assessment capability (or to using automatic assessment by a special monitor) of its behaviour with respect to its goal function. First awkward steps have been made to study and describe formally the time awareness of the perceptive behaviour [28] in the context of proactive system design. Run-time assessment of system behaviour is used as a special case and is still considered too expensive for routine use. Adaptation and learning have been studied mostly as stand-alone phenomena, separately from problems of collecting the required information and also separately from questions related to interactive ontology. Adaptation and learning are commonly used off-line since run-time adaptation may insert too intensive emergent behaviour and thus create too high safety risks.

Looking at today's situation, there is a clear trend to move from human-centred systems to human-supervised systems [29]. This has become possible by introducing proactive components into artificial systems and by increasing the autonomy of the component by decision-making in artificial systems. In order to successfully build and operate these next generation systems one has to consider three aspects of the systems. First, one has to continue improving smartness and proactivity of artificial systems as discussed in the previous sections. Two other aspects are more human-related – to improve the design methodologies and human interfaces for system operators by paying more attention to human limitations and capabilities. The natural (although often implicit) goal, when designing artificial systems, is that the designed system should augment the capability of a human being, improving its capability to stay in command of the situation whenever so desired.

Therefore the future evolution of system and software engineering and their relation to system and software architecture should cater for introduction of those "human" aspects to existing approaches and technologies. The human aspects are well represented and studied in the context of social sciences, e.g. in cognitive

psychology. Some engineering aspects of social sciences have been studied in connection with artificial intelligence, artificial life, cybernetics and other disciplines. A new discipline has emerged, called cognitive engineering. The majority of cognitive engineering research has been focused on human-machine collaboration [30,31]. Leveson [32] claims that the key to successful building of complex systems is improved integration of system engineering, software engineering and cognitive engineering (including research in interactive ontology).

Tennenhouse [29] suggests that proactive computing has been invoked by the same reasons that led to the emergence of cognitive engineering. Indeed, behaviour of artificial proactive components often resembles closely some behavioural aspects of biological creatures/entities, e.g. co-constructing autonomous agents [33]. The increasing number of proactive components in artificial systems and increasing autonomy of components (e.g. freedom for making their own decisions based on the information acquired from their environment) leads to a new application domain of cognitive engineering in artificial systems. Many concepts and notions used traditionally for analysing and describing human behaviour can be applied to studying individual and groupwise behaviour of artificial proactive components (e.g. agents). The conventional problems of matching characteristic features and similarity of goals that are important for efficient collaboration in human communities can also be found in artificial world of self-organizing proactive components.

Notions like negotiations, matching ontology, competition and cooperation, earlier applied for studying communities of biological creatures, are now well established in the community of designers of artificial (multiagent) systems. A new, not yet thoroughly studied problem is the evolution of the ontology of the components during the negotiation (or perception) process. Usually ontology is studied (e.g. in philosophy and linguistics) as a static property. In artificial systems, a proactive component should preferably have interactive ontology. That is, a component should be able to modify its ontology dynamically, if necessary – as a new aspect of automatic adaptation and learning. Those are some of the issues that are to be studied for improving smartness and proactivity of artificial systems during their operation.

As an example, consider the application of cognitive engineering methods applied in the project BRIDGE (ESPRIT 22154). The aim was to create a design environment for diagnosis systems for large-scale complex systems [25]. The theoretical foundation of the project was built around a case-based reasoning system, modified to meet the requirements of the time-critical environment. Specific requirements of the application were, for instance, deterministic time required for reasoning. Most of the variable values in the system had validity time intervals, in some cases additional tests had to be made (within fixed time intervals) on the diagnosed object. Whenever a substantial change in the case base occurred, the time-deterministic reasoning system had to be readjusted.

Some common types of knowledge and actions that were automatically checked in the BRIDGE are illustrated in Table 1. It is clear that different techniques are to be applied to different types of knowledge and actions.

Table 1. Different knowledge needs different processing methods

No.	Knowledge or actions	Checks and analysis performed
1	Knowledge base of the application	Functional integrity of the base, integrity of data and relations, time coherence of variable values, imposed constraints and restrictions
2	Case base used for diagnosis	Coverage of the activation conditions of different parts of the application, coverage of the expected outcome in abnormal conditions, assessing maximum reasoning time after each extension of the case base
3	Correctness and quality of diagnosis	Influence of activation conditions on reachability of necessary diagnosis for this condition, influence of uncertainty in activation conditions on the correctness of the diagnosis, time integrity of conditions and the diagnosis
4	Interactions of components of application and diagnosis systems	Automatic check and adjustment of interfaces to guarantee functional integrity of communicated information, run-time monitoring of timing correctness of interactions

BRIDGE made a step forward in cognitive engineering by applying run-time checking, correcting and warning procedures that a human normally learns and applies for fulfilling its tasks. In cases when automatic correction of a mishap is not possible, the diagnosis system informs the human operator about the potential problems that might have occurred during the performance of the overall system and in the knowledge base and in some cases provides recommendations for corrective actions.

5. CONCLUSIONS

Although in practice of software-intensive systems and automation the use of formal verification methods is often “forgotten”, its power and advantages in verifying the correct behaviour and assessing the quality of service of the system is well understood. In many cases the practical use of formal methods is hindered by the excessive cost involved (in terms of additional processing power and required mathematical experience of developers). In some cases low confidence in proofs (because of overly sophisticated and not quite pertinent to the situation theory) has some influence on the applicability of formal methods.

The latter comment is true in the case of timing analysis of interactions (and any timing analysis at early stages of system development) and by handling time-sensitive proactive behaviour. There is a real need for deeper study of timing and proactivity issues – may be by co-constructing autonomous agents [33] – to provide tools for engineers to evaluate the impact of their design decisions in a structured manner, also in the case of new emerging industrial computer applications. Recent statistics indicates that more than half of all the errors,

detected during a complete software life-cycle, stem from the early stages: user requirements, imposed constraints and restrictions and specification. The situation was almost similar at the end of 1960s. From the system engineering point of view it is true that many inconsistencies and errors actually stem from early stages of the life-cycle of the system, and are due to the integral effect of many minor approximations applied in related theories, algorithms and models. Just think of a stable control algorithm that becomes instable due to excessive jitter of a time unit in a computer application.

It seems that by improving interdisciplinary collaboration in system engineering one might avoid many incoherent or inconsistent requirements and thus easily correct logical and temporal features in software specification that may lead to potential errors. The irony is that even the most advanced software engineering approaches (e.g. UML and MDA) are entirely dependent on the quality of pre-software research – user requirements to the system, system specification plus requirements and constraints to software. An encouraging fact is that UML models have been used for describing non-software entities; still the coherence of various UML descriptions is usually reasoned semi-formally or even informally.

When turning attention to automatic software code generation, after diligent refinement of software models the importance of interdisciplinary collaboration becomes even clearer. The correct interaction between software models, the models of control strategies developed by the control engineer, the systems architecture developed by the system engineer and the behaviour of automatically generated software, running on the target processors, are essential for the correct behaviour of the system. Static analysis of the logical and timely behaviour of the system will allow to assess and correct errors in early stages of development, assuming that there is no emergent behaviour. The majority of new computer applications, however, exhibit a noticeable share of emergent behaviour, caused by potential proactivity of the components and by incompletely known properties of the latter. If the autonomy of components allows for dynamic restructuring of the system, then the static analysis should be complemented by dynamic run-time analysis. Dynamic run-time analysis of the emergent behaviour has not yet attracted sufficient attention of the researchers.

System engineering for software-intensive systems has some distinctive features with respect to other engineering disciplines. People have not yet accepted that software has “natural” limits as well as other components built from “real” materials. Software is not infinitely flexible and malleable. This paper suggests that a reasonably sophisticated time model together with interaction-centred models of computation provide a better understanding of inherent limitations of software-intensive systems and reduces the complexity of system design. Such models encourage an integrated view on a system as a collaborating entity comprising natural world components, (proactive) computing components and humans. Computations are considered as processes invoked by interactions with the non-computer world. The explicit interaction between computing system and its

environment sets natural limits on absolute freedom of thought that often has been believed to be the basic characteristics of software in stand-alone computers in a virtual world.

ACKNOWLEDGEMENTS

The authors gratefully acknowledge comments made by many colleagues. Partial financial support from the Estonian Science Foundation (grant No. 4860), and the Estonian Ministry of Education (projects Nos. 0142509s03 and 0182565s03) is appreciated.

REFERENCES

1. Motus, L., Vingerhoeds, R. A. and Meriste, M. Challenges for real-time systems engineering. Part 1: State of the art. *Proc. Estonian Acad. Sci. Eng*., 2005, **11**, 3–17.
2. Motus, L. and Rodd, M. G. *Timing Analysis of Real-time Software*. Pergamon/Elsevier, 1994.
3. Meriste, M. and Motus, L. On models for time-sensitive interactive computing. *Lecture Notes in Comput. Sci*., 2002, **2329**, 156–165.
4. Wegner, P. Why interaction is more powerful than algorithms. *Comm. ACM*, 1997, **40**, 80–91.
5. Quirk, W. and Gilbert, R. *The Formal Specification of the Requirements of Complex Real-time Systems*. AERE, Harwell, Rep. No. 8602, 1977.
6. Gurevich, Y. Evolving algebras. In *Proc. 13th IFIP Congress*, 1994, **1**, 423–427.
7. Milner, R. *Communicating and Mobile Systems: The PI-calculus*. Cambridge Univ. Pr., 1999.
8. Mellor, S. J., Kendall, S., Uhl, A. and Weise, D. *MDA Distilled*. Addison-Wesley, Boston, 2004.
9. Object Management Group. *UML Profile for Schedulability, Performance, and Time: Specification*. OMG document ptc/2002-03-02, Needham, 2002.
10. Denbigh, K. G. *Three Concepts of Time.* Springer Verlag, Berlin, 1981.
11. Motus, L. Modeling metric time. In *UML for Real: Design of Embedded Real-time Systems* (Selic, B., Lavagno, L. and Martin, G., eds.). Kluwer, Norwell, 2003, 205–220.
12. Sifakis, J. Use of Petri nets for performance evaluation. *Acta Cybern.*, 1979, **4**, 185–202.
13. Manna, Z. and Pnueli, A. *The Temporal Logic of Reactive and Concurrent Systems*. Springer Verlag, New York, 1992.
14. Ostroff, J. S. *Temporal Logic for Real-time Systems.* Research Studies Pr., Taunton; J. Wiley, New York, 1989.
15. Caspi, P. and Halbwachs, N. A functional model for describing and reasoning about time behaviour of computing systems. *Acta Inform.*, 1986, **22**, 595–627.
16. Wegner, P. Interactive foundations of computing. *Theor. Comput. Sci.*, 1998, **192**, 315–351.
17. Lorents, P., Motus, L. and Tekko, J. A language and a calculus for distributed computer control systems description and analysis. In *Software for Computer Control, Selected Papers from the Fourth IFAC/IFIP Symposium*. Oxford. Pergamon/Elsevier, 1986, 159–166.
18. Milner, R. A. A calculus of communicating systems. *Lecture Notes in Comput. Sci.*, 1980, **92**.
19. Motus, L. Timing problems and their handling at system integration. In *Artificial Intelligence in Industrial Decision Making, Control and Automation* (Tsafestas, S. G. and Verbruggen, H. B., eds.). Kluwer, 1995, 67–88.
20. Börger, E. The origins and the development of the ASM method for high level system design and analysis. *J. Universal Comput. Sci.*, 2002, **8**, 2–74.
21. Motus, L. Time concepts in real-time programming. *Control Eng. Pract.*, 1993, **1**, 21–33.

22. Object Management Group. *Enhanced View on Time. Version 1.1*. OMG document formal/02/05/07, 2002.
23. Goldin, D., Keil, D. and Wegner, P. An interactive viewpoint on the role of UML. In *Unified Modeling Language: Systems Analysis, Design, and Development Issues* (Siau, K and Halpin, T., eds.). Idea Group Publ., Hershey, PA, 2001, 250–264.
24. Motus, L. and Naks, T. Formal timing analysis of OMT designs using LIMITS. *Comput. Syst. Sci. Eng.*, 1998, **13**, 161–170.
25. Vingerhoeds, R. A. *Génie d'automatisation et systèmes intélligents temps réels, Proposition d'une méthodologie de conception.* Dossier d'habilitation a diriger des rechèrches, Institut National Polytechnique de Toulouse, 2002.
26. Selic, B. and Motus, L. Using models in real-time software design. *IEEE Control Syst. Mag.*, 2003, **23**, 31–42.
27. Naks, T. and Motus, L. Handling timing in a time-critical reasoning system – a case study. *Ann. Rev. Control*, 2001, **25**, 157–168.
28. Motus, L., Meriste, M., Kelder, T., Helekivi, J. and Kimlaychuk, V. A test-bed for time-sensitive agents – some involved problems. In *9th IEEE International Conference on Emerging Technologies and Factory Automation*. Lisbon, 2003, 645–651.
29. Tennenhouse, D. Proactive computing. *Comm. ACM*, 2000, **43**, 43–50.
30. Vicente, K. J. *The Human Factor: Revolutionizing the Way People Live with Technology*. Knopf Canada, Toronto, 2003.
31. Crowder, R., Bracewell, R., Hughes, G., Kerr, M., Knott, D., Moss, M., Clegg, C., Hall, W., Wallace, K. and Waterson, P. A future vision for the engineering design environment: a future sociotechnical scenario. In *Proc. 14th International Conference on Engineering Design* (Folkeson, A., Gralen, K., Norell, M. and Sellgren, U., eds.). Stockholm, 2003.
32. Leveson, N. G. Software engineering: stretching the limits of complexity. *Comm. ACM*, 1997, **40**, 129–131.
33. Kaufmann, S. A. *Investigations*. Oxford University Press, 2000.

Sardsüsteemide arendustehnoloogia kitsaskohad. 2. osa: Ajatundliku tehnoloogia areng

Leo Mõtus, Robertus A. Vingerhoeds ja Merik Meriste

Artiklis on peamine tähelepanu pööratud ajatundlikele interaktsioonikeskse-tele arvutusmudelitele ja proaktiivsete komponentide kasutuselevõtust tingitud muudatustele süsteemide loomiseks kasutatavates tehnoloogiates. Näidete abil on illustreeritud uusi ideid süsteemide arendamise keskkonna loomiseks, nende realiseerimise protsessi ning mõningaid sellega seotud raskusi.