Available online at www.sciencedirect.com
SCIENCE DIRECT®
Electronic Notes in Theoretical Computer Science
ELSEVIER
Electronic Notes in Theoretical Computer Science 141 (2005) 69–95
www.elsevier.com/locate/entcs

Time-awareness and Proactivity in Models of Interactive Computation

Leo Motus[1]

Tallinn Technical University
19086 Tallinn, Estonia

Merik Meriste

Tartu University Institute of Technology
50411 Tartu, Estonia

Walter Dosch

Institute of Software Technology
University of Luebeck
Luebec, Germany

Abstract

The paper discusses explicit properties and the requirements that are to be verified, imposed upon software-intensive systems by their environment and by their users. Those systems are time-critical, may contain autonomous components, and may exhibit proactive behaviour. It is suggested that the analysis and verification of properties in software-intensive systems requires time-aware model of interactive computation. The authors of this paper claim that hitherto used time interpretation in computer science is too simplified, and several simultaneously maintained independent time counting systems is a necessary precondition for timing analysis of interactions. A feature space for comparing the existing approaches to interactive computing is suggested, and a potential candidate for time-aware model of interactive computation is discussed.

Keywords: proactive and autonomous computing, time-awareness, time-sensitive interactions, many simultaneous metric times, feature space for taxonomy of models of interactive computation.

[1] Email: Leo.Motus@dcc.ttu.ee

1571-0661/$ – see front matter
doi:10.1016/j.entcs.2005.05.017

1 Introduction

The practical design of contemporary computing systems has moved away from the conventional methods based on the Church-Turing algorithm theory. From the pragmatic point of view, the new systems are increasingly built from components (reused or specially developed), and the increasing number of new computing systems are directly interacting with their (non-computer) environment, as well as with the other computing systems. From the philosophical point of view, the new practice has amplified the impact of incomplete information as regards to the properties of components, and the environment. Even in the rare cases when the complete knowledge about natural and/or artificial environments exists, one cannot make use of it because of the limited computing power does not match with the requirements imposed by the environment. Think, for example, of a control of a telomerization reaction where the actual productive duration of the reaction is shorter than the time required for resolving the kinetic equations that determine the recommended duration of the reaction under the given conditions.

The use of components has partly shifted the designers' attention from algorithms to the interaction of algorithms (and their collections). This is because any component may comprise more than one algorithm, whereas precise description of algorithms used in a component and component's inner structure are very seldom known to the designer. Quite often components with different inner structure are equivalent with respect to their external behaviour and can substitute each other in a system. The shift of interest from algorithms to interaction of algorithms has invoked research into interaction-centred models of computation (see, for instance, [22,45,47]).

The direct interaction of a computing system with its environment implies violation of many traditional assumptions widely used in conventional algorithm theory based computation. Some examples of those violations are:

- non-terminating (i.e. on-going) computation has become a rule, not an exception (as in data processing); in the other words, the leading computing paradigm has shifted from string processing to stream processing;
- non-interference assumption, that is mandatory for verifiability of algorithmically parallel computations, is not applicable in forced concurrent mode of computation; forced concurrency is imposed upon the computing system by its environment [24,26] and results usually from stream processing;
- context-awareness of computation has become an issue since computing systems started to interact directly with their environment; in many cases the context-awareness can be reduced to time-awareness and location-awareness of the computation, and/or of the interaction of components; in the case

of learning and/or adapting systems, one should also consider computing and environmental history as one of the features that defines the dynamic context.

Traditionally these issues have been studied for building theoretical foundations for embedded real-time systems, ubiquitous computing systems, and universal plug-and-play systems that are often characterised by the presence of mobile components, dynamically changing topology of inter-component interactions, strong interference from the environment, and pretty strict dependability requirements from their users. The evolution of practical computing systems has shifted the context-awareness problem to one of the central positions in the formal description of computation (models of computation).

1.1 Components, proactivity, agents, and emergent behaviour

Contemporary computing systems are built from components – control and monitoring systems in cars, communication systems, many essential features of transport and banking systems, and medical devices, and computational models are just a few examples. All those computing devices and systems are built from autonomous components (quite often from COTS (commercial, off the shelf)), and are essentially software-intensive. Basic functionality of a software-intensive device is determined by its software. Software-intensive systems differ from the other engineering systems in that their functionality can be easily modified, they are clearly more capable for *explicit proactive behaviour*, and rely on *dynamically evolving control structure* more often as compared to the non-software-intensive systems.

By proactivity the authors of this paper mean component's ability to anticipate the evolution of its environment, to choose the goal-directed activities that lead to better satisfaction of the component's goal, and in the case of a well-designed system to better satisfaction of the system's goal. Usually the proactivity increases components' capability for autonomous operation. The notion of proactive behaviour stems from the natural world and was first applied to artefacts from the artificial world by distributed artificial intelligence, artificial life, and computer control communities. Majority of software-intensive systems operate across the border of natural and artificial worlds – e.g. computer control systems for technical devices and technological processes, medical devices, systems for habitat monitoring, autonomous mobile robots, interactive problem-solving systems – and contain quite often AI based components. Considering the major trend in software design – from object-oriented design to (potentially autonomous) components based design – it becomes only natural to apply proactive components explicitly for designing

software-intensive systems.

The rapidly increasing use of components with autonomous and proactive behaviour is a very clear trend in today's computer applications. A component with autonomous and proactive behaviour matches many definitions of an agent, for instance, [35]. Agents, and systems built from agents (multi-agent systems) is a successful research and application area by itself. However, computationally the software-intensive systems (and devices) that apply proactive components and multi-agent systems face very similar theoretical problems. In a larger scale, this can be considered as an example of cross-fertilization of the knowledge obtained in biology and/or molecular biology with that of the emerging science of the artificial [37].

Such an approach has been named *kenetic engineering*. The name was coined by J. Ferber in the context of distributed artificial intelligence research, and denotes the process of development artificial systems by applying interacting autonomous and proactive components [9]. Similarity with genetic engineering, as defined for natural biological systems, is intentional and emphasises certain cohesion between the building principles of proactive artificial systems as compared to those of biological systems. Multi-agent systems rely essentially on behavioural features that cannot be specified in conventional algorithmic computing, but are inevitably present in real-time, autonomous, and/or proactive computing systems.

Examples of such features are persistency of computation, direct interaction between a computing system and its environment, time-awareness of behaviour, dynamically evolving structure of interactions, and remarkable share of emergent behaviour. These properties cannot be completely specified in advance, during the user requirements specification and design stage. The appearance of those properties depends on the particular context and history of events in operation of the computing system itself, as well as on the context and history of events in the system's environment. Attempts to handle and analyse the above-mentioned features within the paradigm of algorithmic computing have led to theoretical difficulties [3,18,22,23,45,47].

The emergent behaviour has been studied mostly in natural systems. P.W. Anderson stated in [1] that "the behaviour of a complex system can not be understood in terms of a simple extrapolation of the properties of its components, elements, and entities". Anderson's paper was published a couple of years before R. Milner published [22], one of the first papers in the second wave of interest in interaction-centred models of computation. With the increase of the number of components (and their proactivity) in computer systems, and the number of software-intensive systems in practical applications, the role of emergent behaviour in artificial systems has become an important practical

problem. This is additional argument for search of new computing paradigms since the conventional paradigm (based on Church-Turing algorithm theory) practically neglects the emergent behaviour of the designed computing systems.

The evolution of computer science is gradually reaching the understanding formulated by proponents of interactive computing as follows: *"Interactive systems such as modelled by UML represent a new paradigm in computation that inherently cannot be modelled using traditional, or algorithmic, tools. At the heart of the new computing paradigm is the notion that a system's job is not to transform a single static input into an output, but rather to provide an ongoing service"* [11].

1.2 Context-awareness of software-intensive systems

The development and elaboration of the new computing paradigm (interaction-centred model of computation) is progressing well and rapidly gaining popularity. However, so far the context-awareness of computations has gained insufficient attention. Context-awareness has become an issue since computing systems started to interact directly with their environment – especially in the cases when a computing system is to modify the behaviour of the environment, or is to modify its own functionality in response to changes in the state of the environment. Real-time, embedded systems and monitoring and/or diagnostic systems are typical examples of context-aware computing systems.

The precise definition of a context-awareness depends on a particular class of applications. Keeping in mind the contemporary computer applications – e.g. control, monitoring, surveillance, communication, decision-making systems that are built from autonomous, proactive, and mobile components, are heavily distributed, and often rely on heterogeneous ad hoc networks – one can clearly see the necessity for considering time-awareness and location-awareness of the computing. In more sophisticated cases – involving machine-learning, adaptation, and self-organisation – the environmental perceptivity by components of the computing system becomes an important element of the context. To be able to analyse the behaviour of a context-aware computing system, one should consider the environment as part of the analysable system. The environment is also described as a collection of interacting, potentially proactive and autonomous components of a larger entity.

This paper focuses on the time-awareness property of computing systems and postpones the discussions on location-awareness, and environmental perceptivity for the following papers. This decision was taken not only because time-constraint computing systems have been studied for some time already

but mostly because the required time-awareness in contemporary computing applications is philosophically a truly challenging problem, see for example [29,36].

Time has always been present in computing in the form of topological ordering of operations. The performance and scheduling issues for execution time and activation instants of key algorithms became important with the appearance of multiprogramming and multiprocessors, and the related scheduling theories introduced metric time into computing.

The application domain of a single metric time was extended by introducing temporal logics for describing and analysing some of the properties of programs. Many domains of modelling also apply a single metric time (e.g. computational economy). Typically, in those applications the metric time applies either the fully reversible time concept, or the strictly increasing time concept. Some temporal logics use the notion of "branching time" that is not a separate time concept, but rather a method for representing, and reasoning about, alternative evolution paths of the history.

The philosophers distinguish three concepts of a metric time (see, for instance [6]), that usually are not used simultaneously:

- strictly increasing metric time (e.g. as applied in biology)
- fully reversible metric time (e.g. as applied in theoretical physics)
- relative metric time with moving origin (e.g. as applied in psychology).

One of the few examples where all the three concepts of metric time are to be applied simultaneously in order to be able to capture all the required features, is the timing correctness analysis of inter-component interactions in real-time systems, as first suggested in [25], and refined in [29].

Traditionally, a researcher describes the studied phenomena, and a system's designer describes the system that is being developed from his/her point of view, assuming full control of the situation. Hence, the time of the designer and/or of the researcher is sufficient to capture the dynamics of the phenomena or the system. However, in a system composed of proactive autonomous components, each of the components may have its own independent time counting system. Hence, one additional time dimension for the whole system cannot solve the time-awareness problem. This is even more so because, in many cases, the researcher and/or the designer are not in full control of the situation – meaning that the autonomous components cannot always be synchronised with, or reduced to the time of the system designer.

Many practical applications of computing systems accept autonomous and proactive components as building blocks of the computing system, allow direct interaction of those components with the environment, and require that the

computing systems are distributed. Consequently, the researchers have to admit the existence of several independent time counting systems that are to be maintained simultaneously in a complex formed by the (distributed) computing system and its environment. Of course, one can argue whether all those time counting systems are sufficiently influential to be considered explicitly. The existing experience with the embedded real-time systems indicates that in order to analyse the properties of forced concurrent operation it is essential to maintain several independent time counting systems simultaneously so as to verify correct timing of inter-component interactions [26,29].

Whenever the authors mention time-aware interaction-centred model of computation further in this text, it is assumed that the corresponding time-model comprises several independent metric times with simultaneous existence of multiple concepts of each time.

The existing software engineering practice, and the corresponding tools used for developing component-based, and/or agent-based, software support simple time-models that do not foster timing analysis of inter-component (and inter-agent) interactions. In fact, the widely used time-models cater for performance and scheduling features and neglect timing of interactions.

The authors of this paper suggest that the further research should focus on potential ways of designing, assembling, and analysis of computing systems based on autonomous, proactive components. In a large number of applications it is important that the new theories and methods focus explicitly on description and analysis of in-component and inter-component interactions, and specifically support the verifiable satisfaction of context-awareness requirements imposed upon the computing systems.

Two complementary research goals can be pointed out:

- how to build a system that guarantees the required behaviours, enables and assesses the emergent behaviours, and minimises the unwanted behaviours (the research domain of "conventional" interaction-centred computing),
- how to build a system that – in addition to what was said in the previous paragraph – satisfies the imposed context-awareness requirements imposed on individual components, on groups of components, and on interactions of components and on groups of components (the research domain of context-aware interaction-centred computing).

Whereas the first goal can be achieved separately from the second, the second goal cannot be achieved separately from the first goal. This means that for designing and building a computing system, and for analysing its context-awareness related properties, one needs a theory for context-aware interactive computation, and the corresponding model of computation. A natural hy-

pothesis is that the theory for context-aware interactive computing should integrate "seamlessly" the theory for "conventional" interactive computing.

1.3 About this paper

This paper initiates a study of problems that result from attempts to integrate the theories for conventional and context-aware interactive computing. Section 2 of this paper compares the properties of embedded real-time systems with those of time-constraint multi-agent systems and suggests a common paradigm for reasoning about the computational properties of those context-aware interactive computing systems. Section 3 surveys some of the ideas and concepts that have led to explicit formulation of the essence of interactive computation, and eventually to several prototypes of interaction-centred models of computation. Section 4 suggests a feature space that enables to compare computing systems that operate in different applications and have to satisfy a wide spectrum of requirements – e.g. data processing, information processing, computer control, on-line monitoring, decision-support, and multi-agent systems. Section 5 discusses superficially a potential approach to developing a time-aware, interaction-centred model of computation by extending the Q-model with the timed-stream processing methods.

2 Reasoning about component- and agent-based systems

The search for efficient information representation and encapsulation methods that would lead to natural software structuring, has been a driving force for software engineering. The evolution of information encapsulation methods started from modular programming, followed by object-oriented programming and design, and eventually reached the era of component-based software. A component is usually, but not necessarily always, a collection of objects that has limited autonomy, i.e. a component can exist, and to certain extent operate in a stand-alone mode. For its full-scale operation a component usually requires a specific supporting infrastructure.

The practice of object-oriented programming has always (intuitively) followed the paradigm of interactive computing. Still, an object has always had only partial control over its own methods and data structures. An autonomic object (or rather a set of such objects) with full control over its own methods forms a pragmatic basis for implementing agents. In reality, an implemented agent needs a dynamic support infrastructure for its full-scale autonomic operation. Here the full-scale autonomic operation means the social behaviour of an agent – capability to interact with the other agents at its own choice,

and to select actions that ensure satisfaction of time- and other constraints imposed upon its autonomous behaviour by its environment [10].

In the conventional approach to agents, the attention has been usually focused on issues related to agents' intelligence, such as reasoning, beliefs, intentions, desires, negotiations with other agents, and others. In the other words, the research of multi-agent systems has been mostly agent-centred, even organisational aspects of a system have been described and implemented by means of "mental" states of agents. Computational and systems engineering issues in distributed artificial intelligence studies have received comparatively little attention.

In the domain of real-time systems (and embedded systems) the research focus has been on control, monitoring, and communication issues with a strong emphasis on systems engineering aspects. Information encapsulation is of primary importance, and component-based design is widely used. However, computational aspects have gained slightly less attention – note the analogy with the agents. The artificial intelligence methods are increasingly used in real-time embedded systems, in many cases some components of a real-time system are agents. A very clear trend in practical applications is the merger of two domains – multi-agents and real-time embedded systems.

As the result, the infrastructure of multi-agent systems is to be enhanced with a sophisticated time-model, in addition to multiple, potentially independent time-models in individual agents. At the same time, a real-time system is extended by autonomous, proactive components. Consequently, real-time systems are to be considered as loosely coupled collections of interacting autonomous agents (and other components) with time-critical constraints on agents' (and other components') behaviour and on their interactions.

The agents (components of a real-time system) and their interaction patterns may change dynamically during integration, testing, and also during normal operation of the system – for instance, because of autonomous, proactive behaviour of the agents, or because of mobility of agents. Controlled dynamic reconfiguration of components has always been desirable in conventional real-time systems, but has deliberately been avoided to increase the behavioural determinism. In the other words, fixed structure has been applied only to be able to predict, with reasonable confidence, the behaviour of the future system already during its design. The component-based design and steadily increasing autonomy and pro-activeness of components have enabled automatic reconfiguration ability to real-time systems, and increased the share of emergent behaviour in real-time systems to the level that requires reconsidering the methods for behavioural analysis.

A typical agent-based real-time system operates in a time-sensitive envi-

ronment and has a major additional property, as compared with a conventional real-time system – the complete list of interacting agents and the structure of their interactions cannot be finally fixed at the design stage. This property invokes at least two new research topics. First, the agent-based architecture itself is evolving in time and different aspects of the evolving architecture need monitoring (and may be partial control) in order to guarantee the required service and to avoid unwanted emergent behaviours. Second, research of real-time systems, composed from autonomous agents with imposed time and location constraints on agent's individual behaviour, on interaction of agents, and on the overall system's behaviour – needs a qualitatively new model of computations (context-aware interaction-centred model of computation).

In a nutshell, we need a model of computation for a computing system that typically:

- is distributed in a dynamically reconfigurable heterogeneous computer network that comprises wired networks, wireless networks, and multi-hop *ad hoc* networks with potentially mobile nodes
- interacts immediately with the non-computer components of the artificial and/or natural environment
- contains (smart) components with environmental perceptivity, that are capable for autonomous and proactive behaviour
- provides persistently on-going service that supports algorithmic concurrency, as well as forced (or sometimes, true) concurrency in a sense defined in [26,45]
- has a logical structure that can be described as a collection of loosely coupled, context-aware, interacting agents that satisfy all the behavioural requirements and constraints imposed by the environment and by their interaction partners.

3 New concepts of computing and models of computation

The emergence of new concepts of computing

Ubiquitous (and pervasive) computing is based on the expansion of the principles applied in real-time systems and plug-and-play experiments. Computationally new concepts have emerged from the domain of ubiquitous computing in relation with autonomic computing [16] and proactive computing [39]. Those two new concepts are related in a sense that the existence of autonomic components in a system is a precondition to introducing proactivity features to the system. Autonomic and proactive computing has been

compared in [41]. Agents and agent-based systems form a generic example of autonomous and proactive computing.

A system that exhibits proactive behaviour must have autonomous and proactive components. Each of the autonomous components may have its own independent time counting system. Those of the components that exhibit proactivity must have a perceptive subcomponent that collaborates with the goal processing subcomponent. The situation is further complicated if several proactive components form a coalition in order to reach a better perceptivity. A more detailed discussion of proactivity needs a separate publication.

Considering the time issue, each autonomous component may have its own time counting system and each of those time counting systems may apply its own metrics. Strictly speaking, the time instants and intervals defined in different time counting systems (time models) can only be compared within known uncertainty limits. Hence, one time dimension for the whole computing system – that so far has been the conventional approach in computer science and software engineering – cannot solve the time awareness problem. Time in agents has usually been considered in concordance with the traditions of computer science, i.e. time is an additional dimension of a state space – meaning that a single time variable is introduced for the whole system. Examples of traditional time models as used in computer science are discussed and surveyed in [20,50].

The primary reason for the necessity of many metric times for describing the joint operation of a computing system and its environment is the autonomy of some components. In many cases the autonomy of a component also means that the computing system and its designer do not, or can not have access to complete knowledge about the inner operation of the autonomous component. For instance, the components interact with each other directly via dynamically created communication links (e.g. because of the system's emergent behaviour not prohibited earlier by the designer), or the components react to events in the environment (that cannot be controlled by the designer), or the components react to exceptions in the computing system (not foreseen by the designer).

The formal computational aspects of autonomous and proactive computing have not yet been thoroughly studied. Intuitively it is believed that a suitable underlying model of computation should be that of interactive computation. For many applications the model of interactive computation should be extended with time-awareness. Unfortunately, the appropriate and widely accepted formalism for such a model is not yet available. Further in this paper an attempt is made to move towards time-aware model of interactive computation.

Evolution of computing models

A model of computation provides a concise (and, in principle, approximately matching) description of what happens in a computing system [3,46]. Sufficiently precise and widely accepted description of computing has been provided by the concept of Turing machine. However, many of the properties of today's computing systems can not be handled by the concept of Turing machine (and the algorithm theory that stems from the Church-Turing thesis). Turing's seminal paper [40] introduced, apart from "automatic-machine" that later became known as the Turing machine, also "choice-machine", "oracle-machine", and "unorganised machine" (a-, c-, o-, and u-machines respectively). The three latter machines were forgotten for some time as not sufficiently influential for the mainstream computing. At the end of the 1960-es H. A. Simon, in his essays about the science of the artificial (see, for example the third edition of those essays [37]), emphasised the importance of computation over the border of two different environments. Such a computation corresponds well to the ideas of c- and o-machines. An interaction between a computing machine and some other entity that resides in a potentially different environment is considered as essential property of c- and o-machines.

At the end of the 1970-es Simon's essays were followed by a Calculus of Communicating Systems [22] that emphasised the importance of interactions in determining the observable behaviour of Turing machines. Almost at the same time the first method for describing time-aware, forced concurrent, distributed computing system with explicit interactions [34]was published, and was unfortunately left unnoticed for many years.

Interactive computing has gained popularity since the 1990-es. The gradually changing role of practical computing, together with the improved, and more liberal understanding of the essence of computation has led to development of concepts, theories and models of interactive computation, such as [23,42,43,47]. The recent research papers have radically relaxed the strict and traditional restrictions of the Church-Turing algorithm theory. Some of these concepts assume radical generalisation of computability in the Church-Turing sense, for instance [14] defines computability logic where computation is described as game played by machine against the environment. Some others suggest a systematic step-by-step approach (e.g. [12]) that eventually also leads to non-trivial generalisation of Church-Turing thesis.

Starting from CCS [22], quantitative time has been abstracted away from interactive computing. Milner's π-calculus [23] manages to handle system's behaviour without metric time. Wegner's research group formulated basic principles of interactive computing [42,43,45,46] in the 1990-es, and suggested that a multi-stream interaction machine represents the most sophisticated

interactive computation, again neglecting explicit quantitative time. The authors of this paper are convinced that the share of implemented interactive computing systems that need a time-aware (or context-aware) model of interactive computation to reason about their properties is rapidly increasing.

Related philosophical aspects

The evolution of computing has persistently increased the role of uncertainty in the computation. In the framework of Church-Turing algorithm theory the assumption of complete knowledge of the causal relations is essential, and is further amplified by forbidding interference from the environment, or from the other algorithms, during execution of an algorithm. In today's computing systems such assumptions and taboos are remarkably softened, and in many cases have disappeared completely.

Let us consider an example of embedded real-time systems, or multi-agent systems – the case when one interacting partner operates in an artificial world (e.g. a computing system with completely known causal relations) and the other partner operates in the natural world (as a rule, with incompletely known causal relations). The interaction should be on-going rather than terminating, quite often the partner from the natural world is the controlling (or at least equal) partner. The latter property may often cause interference with the execution of an algorithm in a computing system. The interacting partner in the computing system is usually specially designed, or selected to work with a particular component from the natural (or artificial) world. It is reasonable to expect that the dynamic properties of interacting partners match. In practice the dynamic properties of interacting partners can be matched if one of the partners imposes time constraints upon the other's behaviour [29].

The above-described interference situation corresponds to violation of the fundamental assumption in Science – the assumption about the existence of a stationary axiomatic basis of the applied theory (or algorithm) as long as the theory is being used (or the algorithm is being executed). This type of violation can be easily avoided by introducing time constraints that explicitly define the domain where the assumption for stationary axiomatic basis holds. Another useful role of time constraints is approximate presentation of unknown (or incompletely known, or physically not accessible knowledge about) causal relations.

Imagine, for instance, a computer controlled telomerization reaction, where the increase of the useful product slows down after certain concentration has been reached. The reasonable point for stopping the reaction can, in principle, be computed based on a system of kinetic equations for that reaction. However, it takes more time to compute the stopping point than it takes for

the reaction to reach that point. In practice, one can substitute the lengthy computation with an estimated duration of the productive phase of the reaction.

In spite of the rapid increase of time-critical and/or time- and location-aware computer applications, the role of time is still considered by the majority of researchers in a simplified manner – as a single variable, common for all mathematical functions used in the system. This practice stems from mathematics and is still the ruling belief in computer science. Hypothetically this belief is based on the assumption that a neutral observer (e.g. a researcher or a designer of the system) can have complete knowledge about the studied phenomenon, and can observe all the details of that phenomenon, or of the designed system. This assumption, in its turn, results obviously from the not quite correct interpretation of the philosophical foundations of the Newtonian (one single observer) and Einsteinian (several independent observers) theories about the universe.

The necessity of many metric times for describing and analysing the properties of a computing system as stated at the end of section 2 of this paper, is in perfect concordance with the realistic approach taken in the quantum theory. In the quantum theory the universe is considered as a collection of interacting autonomous particles that are all equally important. Even the observers, if they exist, are not more important, nor more capable, than the other particles. In such a universe it becomes natural that each particle may have its own time counting system that may be independent from that of the other particles. This statement follows from the ideas suggested by Simon [37] and Wegner [44].

Examples of approaches towards models of interactive computing

A rather subjectively grouped list of publications on approaches related to the evolution of interactive computing paradigm, and to extending it with time-awareness, follows:

- *State machines* that focus on the state transition view: an input effects on update of the state and on output c-machine [40]; self-reproducing automata [32]; abstract state machine [13]; input/output automata [9]; attributed automata [21]; interaction machine [43];
- *Process algebras*: represented, for instance, by CCS [22] and π-calculus [23]; cost calculus (a process algebra of bounded rational agents for interactive problem solving) [47];
- *Stream-based approach*: time-aware history transformers [5]; compositional refinement of interactive systems [4], [7];

- *Logical framework based* models: represented by weak second order predicate calculus with time [18]; temporal logic [20]; logic of rational agents [48]; computational logic [14];
- *Miscellaneous approaches*: represented by Quirk's report [34], the Q-model [26]; and by agent-group-role model [10].

The above listed publications, together with the intriguing properties of new, practical computing systems forms the basis for selecting the dimensions of the feature space for taxonomy of models of computation. Taxonomy in a carefully selected feature space is to support the comparison of various methods and approaches in order to move towards time-aware model of interactive computing. In this paper we suggest generic taxonomy that implicitly includes all the above listed publications. Development of the detailed taxonomy with explicit positioning of different methods in the feature space needs a separate effort and is not the goal of this paper.

4 Feature space for taxonomy of computations

For systematic progress in developing the time-aware interaction-centred model it would be desirable to categorise the variety of models of computation according to their characteristic features. Examples of feature spaces used earlier by other researchers have been surveyed and discussed in [3,33,45]. The feature space should scatter the different models of computation in the space, and collect similar models into clusters in the space. The earlier used dimensions of the feature spaces applied for taxonomy could not explicitly emphasise the specific properties of context-aware, proactive computing systems. Therefore we suggest the following three dimensional approximation of the feature space – *action, interaction, and time-awareness* (see Figure 1). Further we demonstrate that this feature space clearly distinguishes the conventional models of computation based on the Church-Turing algorithm theory, models of interactive computation, and models for context-aware, interactive computing.

The dimensions of the feature space are interpreted as follows:

- *Action* denotes the execution of a structural unit of a computing system that includes consumption of the input signal (message) and results in producing an output signal (or a message) of that unit that may influence the behaviour of the other structural units, or the environment; a structural unit may be an algorithm, or a component, or an agent
- *Interaction* denotes the process of transferring a signal (message) from the producer of that signal (message) to the consumer; the producer and the consumer are usually structural units of the computing system, or its en-

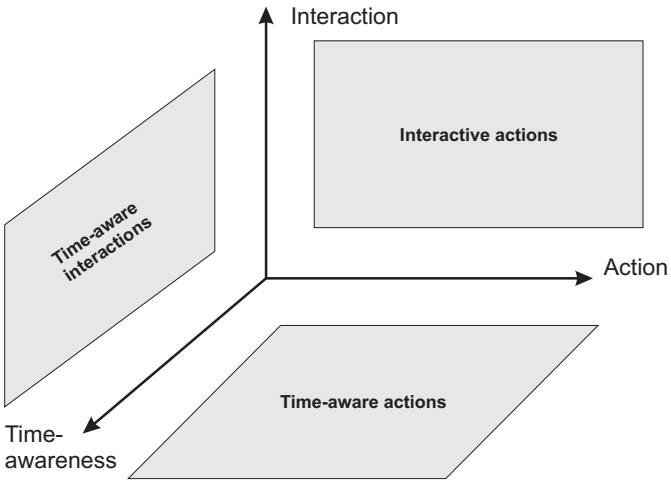

Fig. 1. The feature space and its projections.

vironment; in some cases the message may be deliberately modified in the transfer process (e.g. for reliability, safety or security reasons)

- *Time-awareness* feature characterises time model used in the system – starting from a single topological (i.e. non-metric) time, then a topological time and one metric time, and the most complex case with a topological and several metric times; in some cases it is necessary also to distinguish between the users of a single time concept (e.g. reversible metric time), or several simultaneous time concepts (e.g. reversible, strictly increasing and relative times with the same metrics).

Projections of the feature space onto two-dimensional planes (see Figure 1) – i.e. the planes of *interactive actions*, *time-aware actions*, and *time-aware interactions* – describe rather precisely the already existing research directions (e.g. interactive computing, time-constraint computing, and timing analysis of interactions). Intuitively, the three-dimensional feature space forms a good starting point for building taxonomy of existing approaches to, and for discussing potential new directions in developing, models of computation – in order to explicitly describe and analyse properties that lead to better satisfaction of the requirements imposed upon the computing systems by their environment and their users.

For each feature a metrics will be introduced by markers that define classes of models based on qualitative properties that are of interest for distinguishing models of computation. The number of markers can be increased if a more detailed categorisation is required. The categories considered in this paper are not disjoint, typically the category that is further from the origin of the coordinates includes properties of categories that are closer to the origin of

coordinates. In this paper we will consider the following classes as a starting point.

The *action* dimension is partitioned by the following markers:

- A1 – *actions completely prescribed by algorithms*
  The structural units that perform actions comprise fixed algorithms that are causally related, and the environment may not influence the algorithms and their relations during the action; in control theory such situation is called programmed control.
- A2 – *actions may be influenced by environment*
  Behaviour of some structural units of the system is influenced or controlled by the environment directly during the action, indirectly by the previous actions; in control theory such situation is called feedback control.
- A3 – *proactive actions, as in A2 but also influenced by the goal of the structural unit and by the perceived situation*
  Behaviour of some structural units of the system is proactive and autonomous, meaning that the unit can choose an action from a set of actions that best serves the component in a given situation (smart and selfish components with dynamically changing behaviour).
- A4 – *adaptive actions, as in A3, but the unit may modify of the set of its pre-fixed actions and can dynamically improve its perception abilities*
  In addition to proactivity, some components have capability to learn and adapt their behaviour and goals according to changing conditions (systems with high share of emergent behaviour and hard to predict dynamic behaviour).

The *interaction* dimension is partitioned by the following markers:

- I1 – *algorithmically predefined interactions*
  Interaction between the structural units is strictly predefined by the algorithms (causal reasons) and cannot be modified dynamically; conventional parallel processing, as a case of algorithmic computation, belongs to this category.
- I2 – *dynamic interactions*
  The case of systems that can behave differently in different situations, and where interactions between the structural units actually determine the behaviour of the system (e.g. different algorithms may produce equivalent behaviour of a system); this is a case of interactive computation; stream processing and forced parallel processing are accepted in such systems.

- I3 – *time-constraint dynamic interactions*
  This category comprises models of systems essentially based on time-constraint interactive computation, including constraints imposed upon the occurrence instants of interactions and on the validity of information exchanged during those interactions.

The *time-awareness* dimension is partitioned by the following markers:

- T1 – *a single topological time*
  Sometimes topological time is also called logical time. Time is established by ordering events (that are important to us) that occur in a system without paying attention to the explicit relation between the instant of occurrence an event and the corresponding instant of a metric time. Counting events of executing instructions is a good example – earlier executed instruction has a smaller time label than later executed instruction, independently of the actual value of the instruction counter. Such a time is easily usable in sequential programs, whereas in parallel programs and in programs with distributed processing the ordering of events becomes complicated.
- T2 – *a single metric time*
  This is category for models and systems where, in addition to topological time, one has a metric time (a counter of strictly periodic ticks that can be synchronised with astronomical time as used by humans). The counter of periodic ticks is usually strictly increasing. Based on this counter, the system designer may build timers that enable to use reversible time (i.e. occasionally to back in history, to redo things) or relative time for reasoning about properties in this system. Strictly increasing, reversible, and relative times (as measured by the different timers that apply the same metrics) are called time concepts [6].
- T3 – *multiple metric times*
  This category contains models and systems that accept autonomous and proactive components as building blocks and allow direct interaction of those components with the environment. Consequently, the researchers have to admit the existence of several independent time counting systems that are to be maintained simultaneously in a complex formed by the (distributed) computing system and its environment. By independent time counting systems the authors mean different metric times (i.e. periodic tick counters with different metrics) – see also section 1.2 of this paper. For each of those metric times the system may maintain several simultaneous concepts of time.

The markers that were introduced above are used for partitioning of the fea-

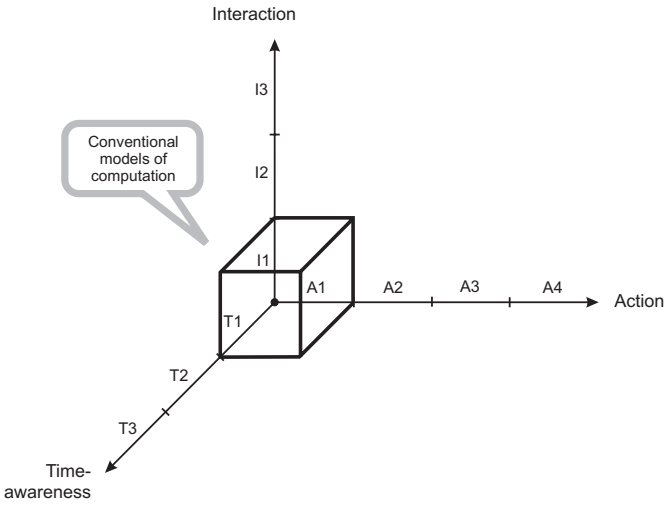

Fig. 2. The feature space with markers.

ture space in Figure 2. As an illustration, the category comprising models of computation that describe computing in terms of Church-Turing algorithm theory is displayed in the vicinity of the origin of coordinates. This category is in the subspace of algorithmically described actions (A1) with algorithmically predefined interactions (I1), and under single topological time (T1). Please note that models using temporal logics do not belong to this subspace, since many of them operate with metric time.

The definitive taxonomy of models of computation in such a feature space is, to the best of our knowledge, not yet available. Categorisation of models of computation in a different feature space has been discussed in [33,50]. Preliminary results suggest that taxonomy based on the features defined in this paper is a valuable tool for planning research in formal description and analysis of the emerging paradigms for computing. A generalised view of relative placement of conventional models of computation, interaction machines and time-aware interaction machines is displayed in Figure 3. More detailed taxonomy that separates particular formalisms – e.g. a variety of methods for handling timing properties – can be developed, provided that a refined marker system is applied.

The suggested feature space stems from the expected properties and requirements of the rapidly spreading new classes of computer applications – such as ubiquitous computing that includes autonomic and proactive components, computing systems with dynamic *ad hoc* architecture, multi-agent systems, time- and location aware computing systems etc. The feature space enables to distinguish between the objectives, capabilities, and scope of the existing models of computation, as well as that of respective tools and re-

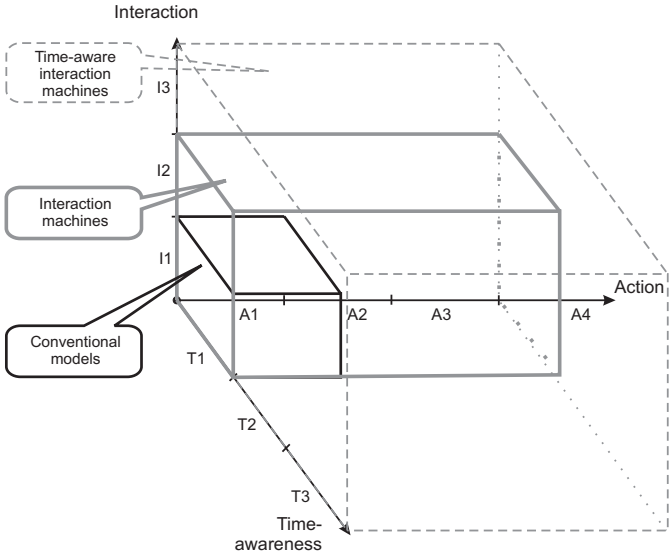

Fig. 3. Sample taxonomy.

sulting products – in practice it is a question of selecting the appropriate set of markers, too few markers give a generalised picture, too detailed markers provide too complex picture.

Time-aware computing is often considered to be something bizarre, not really belonging to the family of decent computing methods. The authors of this paper were encouraged by the first preliminary categorisation of the models in this feature space (see figure 3) – it can be seen that time-awareness is to be considered as a natural extension of algorithmic computing. Our experience with a step-wise development of time-aware models for interactive computing in the KRATT environment [30,31] for time-aware multi-agent systems supports the impression suggested by taxonomy.

Taxonomy in Figure 3 fixes relative positions of conventional models for algorithmic computing, models for interactive computing, and models for time-aware interactive computing. On such a generic level the taxonomy is of little practical use, but if the same taxonomy be used to position more specific products – e.g. Persistent Turing Machines [12], Abstract State Machines [13], π-calculus [23], the Q-model [26] – some useful hints might be extracted for guiding the further research into models for time-aware, proactive, interactive computing.

The following section discusses the use of the Q-model for describing computing systems with dynamically changing configuration and interaction topology, and with time-aware on-going computation in its components.

5 Towards time-aware computing – Q-model and streams

This section introduces preliminary steps taken to formalise the ideas about time-aware interactive computing. The approach is based on an attempt to apply streams as used in the mainstream of computer science, expand streams with time-awareness, and describe the Q-model in terms of the time-aware streams.

In spite of a good progress in researching interactive computing – starting from Milner's CCS [22], followed by a series of publications from Wegner's group [12,43,45,46], and by many others – metric time has been abstracted away. As explained earlier in this paper, many today's applications of computing systems heavily depend on time- and location-awareness. In fact such applications have exited since the 1960-es, but the related research has been outside of the mainstream of computer science.

Approximately at the same time with Milner's CCS, a report by Quirk and Gilbert [34] was published on real-time systems. The report was based on interactive computing concept that was extended by a truly sophisticated time model. The basic result of this publication was further elaborated under the name of Q-model (see, for instance [26]), mapped into a weak second-order predicate logic with time in [18], linked with object-oriented software development environment [27], and suggested for building a real-time UML model-processor for timing analysis of interactions.

Approximately at the same time with Milner's CCS, a report by Quirk and Gilbert [34] was published on real-time systems. The report was based on interactive computing concept that was extended by a truly sophisticated time model. The basic result of this publication was further elaborated under the name of Q-model (see, for instance [26]), mapped into a weak second-order predicate logic with time in [18], linked with object-oriented software development environment [27], and suggested for building a real-time UML model-processor for timing analysis of interactions.

In the following we describe the Q-model superficially, discuss how it satisfies the requirements of the new computing systems (listed at the end of section 2 of this paper), and provide some hints of how the Q-model can be represented by streams.

The Q-model is specifically developed for distributed real-time systems and defines a system as a collection of loosely coupled components (called "processes" because of Q-model tradition) that interact via one-to-one connected and one-way "channels". A component may be implemented as a common process p that is a straightforward mapping from domain of definition to

value range of the process whereas the mapping is repeated many (up to the countable number of) times, i.e.

$$p : \mathcal{T}(p) \times dom\ p \rightarrow\ val\ p\ , \text{where}$$

$\mathcal{T}(p)$ is a well-ordered time-set that determines the time instants when the mapping is executed. A component may also be implemented as a selector process [26] – not considered in this paper due to its complexity. The mapping may be defined across the border-line of different environments, thus covering the case when a component in a computing system interacts with a non-computer component in the artificial or natural environment. The defined mapping can behave autonomously since the time-set that determines when to execute and when not to execute belongs to the mapping. Also the proactivity can easily be built on the potentially existing inner memory of the mapping, plus many additional decision-making mechanisms in the case of selector process.

The time-set may describe periodic, quasi-periodic, and spontaneous execution of the component (depending on the definition of the time-set's elements). The time-set may also be used for modelling dynamic reconfiguration of the computer network, and temporary or permanent disappearance of some components. Naturally, we rely on discrete time since continuous time causes implementation problems.

Such a mapping specifies a stream that allows additional flexibility as compared to conventional stream processing – each component (the corresponding stream) may have a different execution pattern (time-set), the streams need not be executed at regular intervals (that may have good as well as bad consequences).

The interaction of a component with another component generates a specific interaction stream (see figure 4).

The formation of the interaction stream is controlled by the *consumer* p_j component (the one that receives a message) while the other partner in the interaction is the *producer* p_i (the one that provides data for the communicated message). The mechanism that forms the message required by the consumer form the data produced by the producer and transfers the message is called in the Q-model a *channel*. A *channel* is a mapping:

$$\sigma_{ij} : val\ p_i \times \mathcal{T}(p_i) \times \mathcal{T}(p_j) \rightarrow proj_{val\ p_i}\ dom\ p_j\ ,$$

where the length of the message (i.e. depth of the consumer required memory, or the length of an element in the interaction stream) is determined by a channel function

$$K(\sigma_{ij}, t) \subset \mathcal{T}(p_i),\ \text{and t} \in \mathcal{T}(\text{p}_\text{j}).$$

Please note that different time-sets may represent different time counting

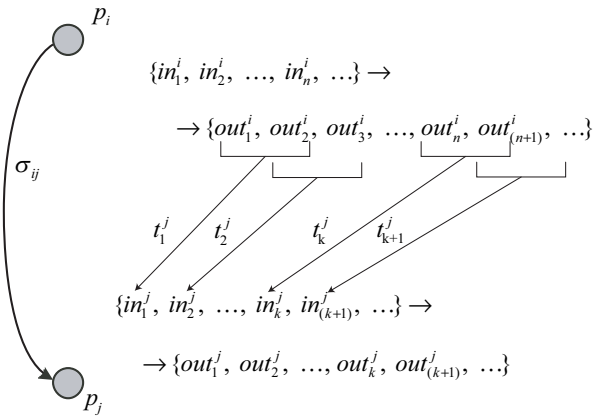

Fig. 4. Illustration of a time-selective interaction.

systems (different metric times), and the channel function defines a timer for relative time on the producer time-set $\mathcal{T}(p_i)$. The origin of this relative time is moving with the progress of the consumer process and is defined by the instant $t \in \mathcal{T}(p_j)$ when the consumer is activated. Depending on the relationship between the producer and consumer time-sets, one may need three basic types of channels – synchronous when $\mathcal{T}(p_i) \equiv \mathcal{T}(p_j)$, semi-synchronous when $\mathcal{T}(p_i) \rightarrow \mathcal{T}(p_j)$, and asynchronous when the time-sets are independent of each other. The interaction stream, formed by a channel, is to be time-labelled in the defined relative time so that the time-constraints imposed upon the interaction by the consumer process can be verifiably satisfied – even when considering the potential uncertainty introduced by matching times with different metrics.

6 Conclusions

The paper focuses on properties, development methods, analysis methods, and tools for software-intensive systems directly interacting with their environment. Many such systems are built from autonomous components that may exhibit proactive behaviour. Software-intensive systems differ from the other engineering systems in that they are clearly more capable for *explicit proactive behaviour* and rely on *dynamic control structure* more often as compared to the non-software-intensive systems in the artificial world (see also [37]). This paper states that applications of software-intensive systems require properties that cannot be studied by conventional mainstream methods of computer science, and suggests that a new time-aware model of interactive computation is to be developed.

Large part of the paper explains specific role of time in new computing

systems that directly interact with their environment and may contain autonomous and proactive components. This has led to emphasising the need for introducing a truly sophisticated time model into the mentioned computing systems and into respective models of computation. In a system composed of autonomous and proactive components, each of the components may have its own independent time counting system. Hence, one additional time dimension for the whole system cannot solve the time-awareness problem. Formal timing analysis of interactions in such systems presumes the use of more than one metric time plus simultaneous use of three time concepts (strictly increasing, fully reversible and relative with moving origin), such time model has not been widely used in computer science so far.

Many promising attempts have been published to formalise models of interactive computation, and to introduce time-awareness to computing. This paper suggests a new feature space that would facilitate comparison of the existing approaches, and select the proper starting point for developing time-aware model of interactive computing. The feature space is suitable for building taxonomy of the existing models. However taxonomy has not been built in this paper. Instead, a superficial discussion of the ongoing research on merging the Q-model approach with processing methods of timed streams is presented. The taxonomy and the formal study of relationships between different approaches to (time-aware) interactive computation are still to be done.

This material published in this paper is based on interim results of an ongoing larger project carried out in the Estonian Centre of Excellence for Dependable Computing (CDC) – a long term joint venture of Tallinn University of Technology, Tartu University Institute of Technology, and recently joined University of Luebeck.

Acknowledgement

This research has been partially financed by Estonian Science Foundation (ETF) grant no. 4860, and by a grants no. 014 2509s03 and no. 018 2565s03 from the Estonian Ministry of Education. The authors sincerely appreciate the questions and recommendations of the reviewers that inspired major improvements in the manuscripts.

References

[1] Anderson P. W., *More is different*, Science, **177**, No. 4047, (1972), 393–396.

[2] Bigus, J. P., D. A. Schlosnagle, J. R. Pilgrim, W. N. Mills and Y. Diao, *ABLE: A Toolkit for Building Multi-agent Autonomic Systems*, IBM Systems Journal, **41** (2002), 350–371.

[3] Blass, A. and Y. Gurevich, *Algorithms: A quest for absolute definitions*, Bulletin of European Assoc. for Theoretical Computer Science, **81** (2003), 195–225.

[4] Broy, M., *Compositional Refinement of Interactive Systems*, Journal of the ACM, **44** (1997), 850–891.

[5] Caspi, P. and N. Halbwachs, *A Functional Model for Describing and Reasoning about Time Behaviour of Computing Systems*, Acta Informatica, **22** (1986), 595–627.

[6] Denbigh K. G. "Three concepts of time", Springer Verlag, 1981.

[7] Dosch, W. and A. Stümpel, *Introducing Control States into Communication Based Specifications of Interactive Components*. In: H.R. Arabnia, H. Reza (eds.): Proceedings of the International Conference of Software Engineering Research and Practice (SERP'04), Volume II. Las Vegas, Nevada, June 21-24, 2004. Athens, GA: CSREA Press, 2004, 875–881.

[8] Eberbach, E., D. Goldin and P. Wegner, *Turing's Ideas and Models of Computation*, book chapter in "Alan Turing: Life and Legacy of a Great Thinker", ed. Christof Teuscher, Springer, 2004.

[9] Ferber, J., "Multi-agent systems. An Introduction to Distributed Artificial Intelligence", Addison-Wesley, Harley (UK), 1999.

[10] Ferber, J., O. Gutknecht and F. Michel, *From Agents to Organizations: An Organizational View of Multi-agent Systems*, P. Giorgini, J.P. Müller, J. Odell (Eds.): AOSE 2003, LNCS **2935** (2004), 214–230.

[11] Goldin, D., D. Keil, and P. Wegner, *An Interactive Viewpoint on the Role of UML*, Ch.15. in Unified Modeling Language: Systems Analysis, Design, and Development Issues, K. Siau and T. Halpin (Eds)., Hershey, PA: Idea Group Publishing, 2001, 250–264.

[12] Goldin D., S. Smolka, P. Attie and E. Sonderegger, *Turing Machines, Transition Systems, and Interaction*, Information and Computation Journal, **194**, No. 2 (2004), 101–128.

[13] Gurevich, Y., *Evolving algebras 1993: Lipari guide*. In Borger, Ed., Specification and validation methods, 1995, 231–243.

[14] Japaridze, G., *Introduction to computability logic*, Annals of Pure and Applied Logic, **123** (2003), 1–99.

[15] Jennings, N. R., *An Agent-based Approach for Building Complex Software Systems*, Communications of the ACM, **44**, No. 4 (2001), 35–41.

[16] Kephart, J. O. and D. M. Chess, *The Vision of Autonomic Computing*, Computer, **36**, No. 1 (2003), 41–50.

[17] Lamport, L., *The Temporal Logic of Actions*, ACM Transactions on Programming Languages and Systems, **16** (1994), 872–923.

[18] Lorents, P., L. Motus, and J. Tekko, *A Language and a Calculus for Distributed Computer Control Systems Description and Analysis*, Proc. on Software for Computer Control, Pergamon/Elsevier (1986) 159–166.

[19] Lynch, N. A. and M. R. Tuttle, *An introduction to input/output automata*, CWI Quarterly 2(3) (1989), 219–246.

[20] Manna, Z. and A. Pnueli, "The temporal logic of Reactive and Concurrent systems: Specifications", Springer Verlag, 1991.

[21] Meriste, M. and J. Penjam, *Attributed Models of Computing*, Proc. of the Estonian Academy of Sciences. Engineering, **1** (1995), 139–157.

[22] Milner, R. A., "A Calculus of Communicating Systems", LNCS, **92** (1980), 171p.

[23] Milner, R., "Communicating and Mobile Systems: The π-calculus", Cambridge University Press, 1999.

[24] Motus, L. and P. Lorents, *Specification of distributed real-time systems*, International Conference Control'85, Conf. Publ. No. 252, IEE, London, UK, **2** (1985), 569–574.

[25] Motus L., *Time concepts in real-time programming*, Control Engineering Practice, **1**, No. 1, (1993), 21–33.

[26] Motus, L. and M. G. Rodd, "Timing Analysis of Real-time Software", Elsevier, 1994.

[27] Motus, L. and T. Naks, *Formal timing analysis of OMT designs using LIMITS*, Computer Systems Science and Engineering, **13**, No. 3 (1998), 161–170.

[28] Motus, L., M. Meriste, T. Kelder and J. Helekivi, *An Architecture for a Multi-agent System Test-bed*, Proceedings of the 15th IFAC World Congress, vol. L, Elsevier Science Publ., (2002), 6 pp.

[29] Motus, L., *Modeling metric time*. In B. Selic, L. Lavagno, G. Martin (Eds), UML for Real: Design of Embedded Real-time Systems, Kluwer Academic Publ., Norwell (2003), 205–220.

[30] Motus, L., M. Meriste, T. Kelder, J. Helekivi and V. Kimlaychuk, *A test-bed for time-sensitive agents – some involved problems*, 9th IEEE Intern Conf. on Emerging Technologies and Factory Automation, Portugal, **2** (2003), 645–651.

[31] Motus, L., M. Meriste, T. Kelder and J. Helekivi, *Agent-based Templates for Implementing Proactive Real-time Systems*, Proc. International Conference on Computing, Communications and Control Technologies, Austin, Texas, Vol. 1, (2004), 199–204.

[32] Von Neumann, J., "Theory of Self-Reproducing Automata", Univ. of Illinois Press, 1966.

[33] Van Parunak, H., S. Brueckner, M. Fleischer and J. Odell, *A Preliminary Taxonomy of Multi-Agent Interactions*, 2nd Int.Conf. on Autonomous Agents and Multi-agent Systems (2003), 1090–1091.

[34] Quirk, W. J. and R. Gilbert, "The formal specification of the requirements of complex real-time systems", AERE, Harwell, rep. No. 8602, 1977.

[35] Russell, S. J. and P. Norvig, "Artificial Intelligence: A Modern Approach", Englewood Cliffs, NJ: Prentice Hall, 1995.

[36] Selic, B. and L. Motus, *Modeling of Real-time Software with UML*, IEEE Control Systems Magazine, **23**, No. 3 (2003), 31–42.

[37] Simon H. A., "The Science of the Artificial", MIT Press, 1996.

[38] Stümpel, A., "Stream Based Design of Distributed Systems through Refinement", Logos Verlag Berlin, 2003.

[39] Tennenhouse, D. L., *Proactive Computing*, Communications of the ACM, **43**, No. 5 (2000), 43–50.

[40] Turing, A., *On Computable Numbers, with an Application to the Entscheidungsproblem*, Proc. London Math. Society, **42:2** (1936), 230–265; A correction, ibid, **43** (1937), 544–546.

[41] Want, R., T. Pering and D. Tennenhouse, *Comparing Autonomic and Proactive computing*, IBM Systems Journal, **42** (2003) 129–135.

[42] Wegner, P., *Interaction as a Basis for Empirical Computer Science*, ACM Computing Surveys, **27**, No. 1 (1995), 45–48.

[43] Wegner, P., *Why Interaction is More Powerful than Algorithms*, Comm. of ACM, **40**, No. 5 (1997), 80–91.

[44] Wegner, P., *Towards empirical Computer Science*, Monist., **82**, No. 1 (1998), 58–108.

[45] Wegner, P., *Interactive Foundations of Computing*, Theoretical Computer Science, **192** (1998), 315–351.

[46] Wegner, P. and D. Goldin, *Coinductive Models of Finite Computing Agents*, Electronic Notes in Theoretical Computer Science, **19** (1999).

[47] Wegner, P. and E. Eberbach, *New models of Computation*, Computer, **47**, No. 1 (2004), 4–9.

[48] Wooldridge, M., "Reasoning about rational agents", MIT Press, 2000.

[49] Wooldridge, M., *On the Sources of Complexity in Agent Design*, Applied Artificial Intelligence, **14** (2000), 623–644.

[50] Yu, S., *The Time Dimension of Computational Models*, Tech. Report No. 549. Univ. of Western Ontario, Dep. of Comp. Sci., 2000.