SCIENTIFIC PAPERS, UNIVERSITY OF LATVIA, 2010. Vol. 756
COMPUTER SCIENCE AND INFORMATION TECHNOLOGIES 193–203 P.

The Concept of Automated Process Control

Ivo Oditis[1], **Janis Bicevskis**[2]
[1] Bank of Latvia, K. Valdemara 2a, Riga, Latvia
ivo.oditis@lais.lv
[2] University of Latvia, Raina bulv. 19, Riga, Latvia
Janis.Bicevskis@lu.lv

This paper describes research on control of heterogeneous information systems, which run as parallel interlinked processes. A formalized process control description language is proposed. It is a domain-specific language which provides opportunity to automate process execution control mechanism. The language separates two types of processes: base and supervisory processes. Supervisory processes require specific language elements for the control and synchronization of base processes. Also, the first concept of automated control mechanism is introduced. The proposed mechanism and process control definition language is developed as part of smart technology framework aiming at autonomous system concept developed by IBM.

Keywords: business process control, domain-specific languages.

Introduction

For many years computer scientists spent most of their work on research of software development technologies, while less effort was spent to make the use of the already developed software more convenient. In part this problem can be explained by software developers' concerns about software sales leaving software usage problems into users' hands. The complexity of the whole system is increased when one company or organization acquires software from more than one vendor and software is introduced with significant time span. This way a complex heterogeneous system environment is formed.

There are at least two groups among system users: those who are end users or users of the system's business functionality and system administrators whose responsibilities include system security and technical configuration of system and its environment. By its complexity the area of business system administration and control is comparable to network administration. There are numerous tools for network administration and monitoring in the market; however, the authors of this paper could find no acceptable solutions for heterogeneous system administration and their process execution control. It can be explained by the diversity of the systems and the nonstandard nature of legacy system communication.

Process control is a well-known problem. There have been many attempts to solve it in software history [1]. In the era of mainframes, process management was partly delegated to the operating system and Job Control Language. As a significant tool of this area, the SDL or Specification and Description Language must be mentioned. SDL is a specification language targeted at unambiguous specification and description of the behavior of reactive and distributed systems. Originally focused on telecommunication systems, its current areas of application include process control and real-time applications in general; however, it requires a very detailed process description and is not suitable for high-level business process description. Therefore, authors introduce a new and simple domain-specific language for business process control.

In the first chapter, problems are identified and brief solutions are explained. The second chapter describes the architecture of the process control mechanism and the domain-specific language used for process control description.

1 Description of the Problem

1.1 Usage of Heterogeneous Systems

This publication is aimed to describe the usage of heterogeneous software in large companies, where many different software platforms are used. The formation of heterogeneous environment in long running large companies is unavoidable, if the necessary software is acquired gradually and the size and functions of the companies are changing over time.

The most serious problems are caused by distributed environment where many systems are running simultaneously on different platforms and communicating with each other. Typically, operators and administrators of each of these systems have to have specific management skills. As a rule, service staff have to follow if processes carried out by the systems are done correctly; and there is no process control built in systems. If one process is carried out by two or more independent systems, each system can control execution as far as it is in its scope, but the whole process typically is controlled manually. Therefore, the usage of systems depends on the qualification of the supporting staff and precise execution of operations by the staff.

Automated process control is proposed to solve the problems described above and to reduce the dependency of system usage on the subjective factor of the supporting staff. In brief, the proposed solution contains two components: a description of controlled processes and a mechanism for controlling process execution according to process description.

1.2 Smart Technology and Autonomous Systems

The proposed process control automation solution is based on the ideas of *smart technology* [2]. The idea of *smart technology* is to create software similar to a live organism, which can react adequately to unpredictable changes of living environment. Ideally software built according to the principles of smart technologies could adequately react to the changes of the external environment (changes of infrastructure, network throughput etc) as well as to the internal environment. Smart technologies provide a framework for software development. Using a common framework, smart technology could be included in systems without significant increase of software complexity.

The concept of smart technology includes external environment testing [3, 4], intelligent version updating [5], self-testing [6] and others. The concept of smart technology has similar goals as the concept of autonomous systems developed by IBM in 2001 [7, 8, 9]. Both concepts aim at improving software intellect by adding a set of nonfunctional advantages – ability to adapt to external situation, self-renewing, self-optimizing and other advantages. The autonomous systems are built as universal and independent form properties of a specific system. As a rule, they function outside of a specific system and cooperate on the level of application interface.

The first results of smart technology implementations are available. There are two types of smart technology software developed and introduced in currently used systems: intelligent version updating software and software for external environment testing. The first is used in budget planning and the discharge control system FIBU. It is used in more than 400 state-owned organizations in the Republic of Latvia. There are more than 2 000 users of this software. The external environment testing software is used by the same system, FIBU, to solve the problem of many operating systems and other application versions. The same external environment testing software is planned to be used in the Bank of Latvia to manage numerous independently developed, but interrelated systems. The first results of the use of smart technology demonstrate their practical usefulness [10, 11].

This research is on smart technology as well: automated operation control of heterogeneous systems.

1.3 Process Control Automation as Part of Smart Technology Framework

The suggested automated process control concept can be identified as another extension of smart technology. It works over heterogeneous systems and semi-automates system process control.

The solution introduces two types of processes:

- *base processes* – simple processes containing no sub-processes;
- *supervisory processes* that contain, control and synchronize base processes.

Processes implemented by computer systems are mutable by their nature. Process modifications can be caused by changes in system infrastructure, by changes in process priorities or changes in organization structure. The suggested process control concept contains two components: the process control mechanism and process control description. The process control mechanism is running according to easily adjustable process description. It is useful to implement process control description by introducing a domain-specific language with elements particular for the control of base and supervisory processes.

We briefly describe the process control mechanism and process control definition language. Detailed implementation of the language and control mechanism is subject of further research.

Processes from a payment clearing system will be used in descriptions of the process control mechanism and process control definition language. Payment clearing systems provide a large volume of retail payment exchange and settlement (clearing) among banks (system participants). Typically, a clearing process is organized in four steps: systems receive retail payment batches from participants, calculate each participant's position (difference of payments' totals sent and received by participant), settle positions in the system where participant accounts are kept and deliver payments to participants, payment receivers. This is called a "clearing cycle". Systems with one or more clearing cycles per day exist. A system with one clearing cycle will be used in our examples.

Using the clearing system, base and main processes can be identified. Receiving payments from one clearing participant can be described as a base process, and the whole clearing cycle can be identified as a supervisory process containing and controlling a set of base processes (each participant's payment collection).

2 Automated Process Control

Automated process or system operation control mechanisms check if the described system processes are running according to the process descriptions. The process sequence and operation timing is checked. If a discrepancy between the description and ongoing processes is detected, the control mechanism sends information to system support stuff. Two types of information can be identified: timely warnings (the system tries to identify potential problems) and information on the detected errors.

The control mechanism's main task is to continuously verify whether the process flow is correct, incoming and outgoing data is coherent, all process steps are done and whether all of the steps are done timely. The control mechanism does not test the system under control nor does it test the quality of data produced by the system.

Another important component of the process control mechanism is process trace recorder. Process traces could be useful not only to identify possible causes when a problem has occurred, but also could provide substantial statistical information on a typical system workload and bottlenecks. Analysis of system traces could provide early warnings about changes in process execution times.

Three collaboration types are possible between the control mechanism and systems under control:

- the system under control is interpreted as a black box from the control mechanism perspective, and all information on the process flow is taken from system external interfaces;
- the system under control sends information to the control mechanism on process execution;
- the system under control requests information from the control mechanism on process execution.

2.1 The Architecture of the Process Control Mechanism

A significant number of systems are distributed over more than one server. Thus, one of the most important requirements for the architecture of the system control mechanism is the possibility to control widely distributed systems.

The control mechanism offered by the authors contains two main components: *Central Hub* and *Agents*. Agents are software modules which trace different events of the systems implementing the process under control. For example, a new file or file modification could be one type of event handled by agents. When agent detects event related to the process under control, it sends event notification to the central in order to check if the process is running according to process description and if the timing of the process is accurate.

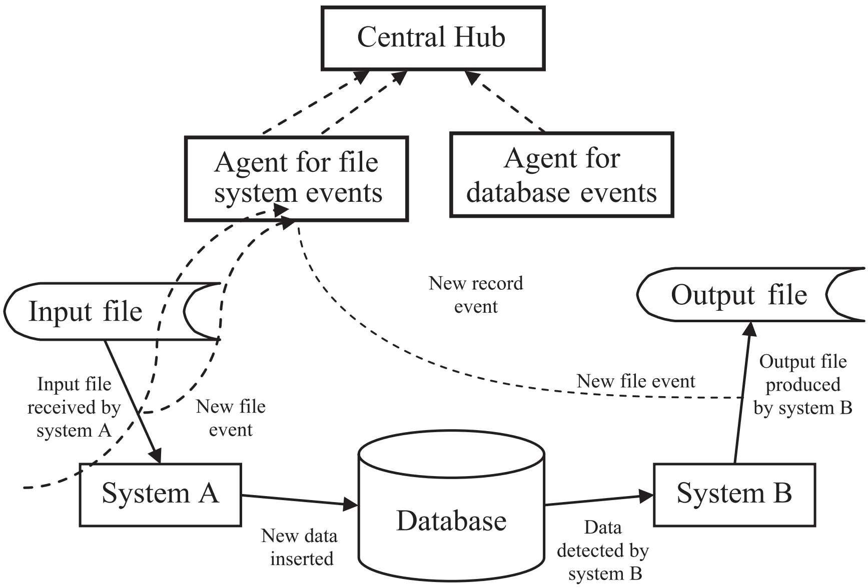

Fig. 1. The control mechanism contains two types of components: *Central Hub* and *Agents*

For instance (Fig. 1), one process could be provided by two systems: A and B. *System A* takes an input file, processes its contents and inserts new data in a database. *System B* takes the data from the database and produces another output file. *Central Hub* is controlling the whole process by using two event agents: one agent provides file system events, another – database events. When input file is received, the file system event agent receives "new file" event and passes it to *Central Hub*. After *System A* inserts data into database, "new record" events are handled by the second agent and sent to *Central Hub* for processing. When *System B* creates an output file one more "new file" event is handled by the first agent and passed to *Central Hub*.

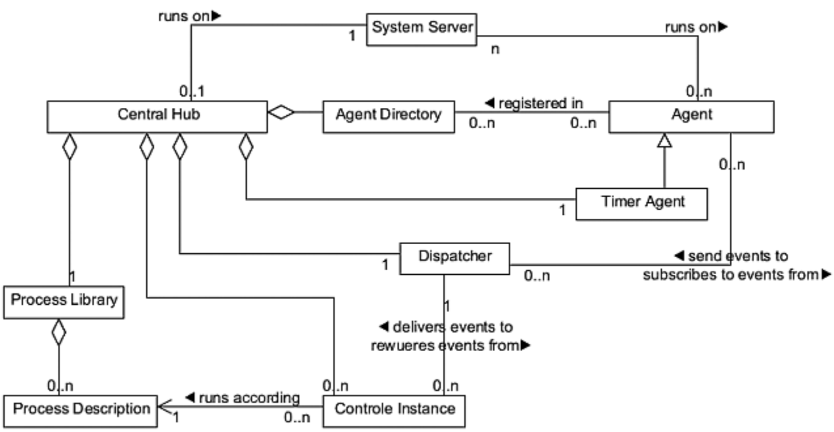

Fig. 2. A UML class diagram of the process control mechanism components

Central Hub (Fig. 2 represents components of the process control mechanism) contains five modules: *Process Library*, *Controller Instances*, event *Dispatcher*, *Agent Directory* and *Timer Agent*. *Process Library* contains all of *Process Description* the *Central Hub* has to look after. When a new process is started, *Central Hub* takes the *Process Description* from *Process Library* and creates new *Controller Instance* to control the process flow. *Controller Instance* analyses *Process Description* and receives events from *Dispatcher* the process could generate. Data *Dispatcher* subscribes to the appropriate type of event from appropriate *Agent* according to the required event types and *Agent Directory*. When *Agent* handles the requested event, it sends it to the *Central Hub's Dispatcher* where *Controller Instance* which requested the event is identified. *Control Instance* processes each event and checks process state according to *Process Description*. If events are fired in inappropriate order, *Controller Instance* sends error messages or warnings to the person in charge according to *Process Description* notification rules. There is a specific type of *Agent*, the *Timer Agent*, hosted in *Central Hub*. It provides timer events to *Controller Instances*. *Control Hub* is hosted on one server; however, there may be more than one *Agent* hosted inside a network on many servers. Many servers may host many *Agents*, but only one of each type. Thus, the control mechanism can be applied to a widely distributed system.

2.2 The Process Control Description Language ProCDeL

The domain-specific language ProCDeL is introduced by authors for description of controlled processes. The language was developed with two main criteria in mind:

- it must be easy to use by various types of users (from system administrators to skillful end users);
- the language should be used for rather complex process description.

The first criterion sets a requirement for the process description language to have both graphical and textual notation so that processes could be represented as graphs or scripts.

The concept of the process control definition language is similar to BiLingva [12], where a typical state chart diagram (contains state and connection elements) is supplemented with action elements. The process control definition language ProCDeL contains three types of elements: states, events (connections in state charts) and flow control elements (actions in BiLingva). Process flow control elements allow to describe parallel process execution, loops and control over other processes.

2.2.1 An Example of ProCDeL Usage

Let us demonstrate the language elements by example. Fig. 3 describes an electronic clearing payment system. The process starts with event *Clearing day opened*. It must be done no later than at 8:30 in the morning. After the day has been opened, the system starts to process incoming client payment files. Those are processed in parallel. At 14:00 file reception must be stopped and payments may be settled. After it is done, all payments are delivered to recipients.

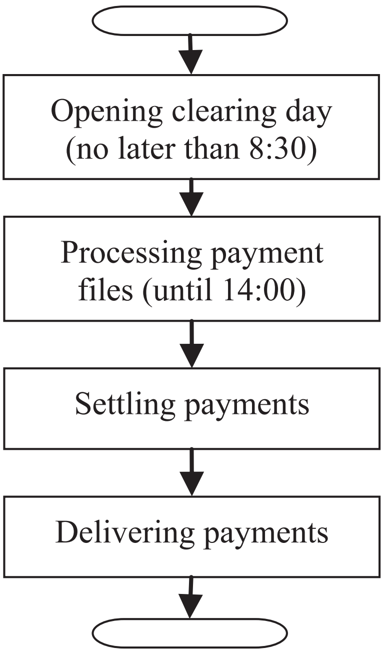

Fig. 3. The clearing process workflow

This process can be described in the process description language as a graph (Fig. 4) or as a textual version. The first event that the control mechanism has to detect is the beginning of the clearing day. This can be done by checking the database for new day event.

Next step in the clearing process workflow (Fig. 3) is file processing. There is another process description named *ReceiveIncomingFiles* made according to this workflow step. It describes file processing for one clearing participant. All clearing participants must be identified before process *ReceiveIncomingFiles* is started. It is done by flow control operation *ControlData (Banks, DBData.Procedure, [CLEAR_GET_PARTICIPANTS]).* After all participants are found, next control flow operation is executed to load file processing processes for each participant.

When all files are received, file reception must be closed. It is identified by state *FILE_RECEPTION_CLOSED* in the process description graph. This state can be reached when all file processing processes are finished. There is one more control added to this state – time 14:00. It means that the control mechanism must check if the state is reached by 14:00.

There are two more events and states following *FILE_RECEPTION_CLOSED*. The first event detects when all of payments are settled. This event corresponds to workflow step *Settling payments*. On this event, the process moves to the next state *PAYMENTS_SETTLED*. There is no time control for this state. The last event in this process description is the event that identifies the end of payment delivery. This event leads the process into the ending state with time control 14:30.

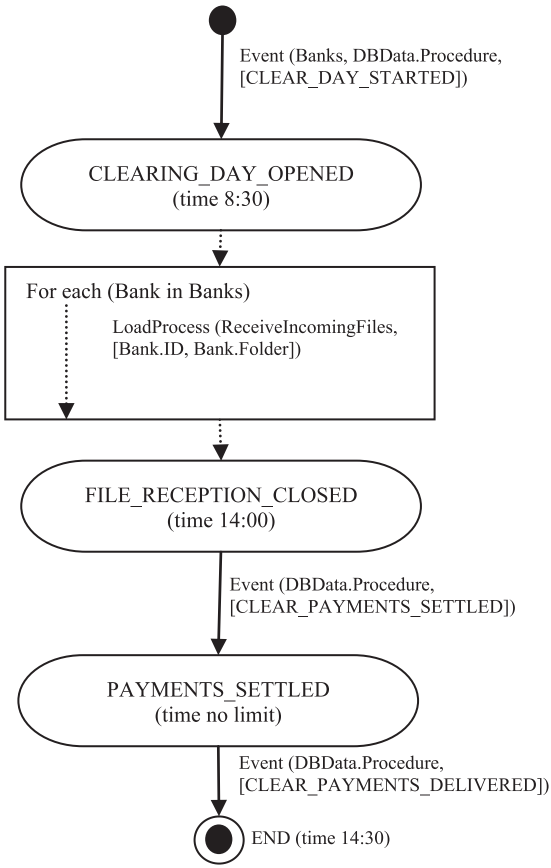

Fig. 4. The clearing process described in the process definition language

All of the processes mentioned above can be described in a textual form using the same language.

Clearing process description in a textual form in ProCDeL

```
process PaymentDay{
    event (Banks, DBData.Procedure,
          [CLEAR_DAY_STARTED]);
    state CLEARING_DAY_OPENED (time 8:30);
    forEach (Bank in Banks) {
          loadProcess (ReceiveIncomingFiles, [Bank.ID,
                                              Bank.Folder]);
    }
    state FILE_RECEPTION_CLOSED (time 14:00);
    event (DBData.Procedure, [CLEAR_PAYMENTS_SETTLED]);
    state PAYMENTS_SETTLED (time no limit);
```

```
    event (DBData.Procedure,
          [CLEAR_PAYMENTS_DELIVERED]);
    state END (time 14:30);
}
```

The example shows just one kind of event (DBData.Procedure – event occurs if the database procedure returns any data); however, there are no limitations to event types in the process description language. As many events as event agents implemented in the control mechanism can be used: for instance, file system event agents, database event agents, e-mail agents etc.

Each event's occurrence returns results that can be used in other events. For instance (Fig. 3), first event in the process *PaymentDay* is type of *DBData.Process* and it calls database procedure *CLEAR_DAY_STARTED*. It returns all of clearing participants and those are loaded in the variable *Banks*. Later the variable *Banks* may be used in other events or control flow statements as an argument. The variable *Banks* was used in the process *PaymentDay* to define *forEach* statement (loop over all list items).

Discussions are still ongoing on,how to describe reporting issues in the process description language. When the control mechanism detects improper process execution according to the process definition, it must send some alarms to the person who is in charge. There could be a rather simple process control with just one type of alarm (for instance, error messages) and one recipient. However, many complex processes running over more than one system could have errors, warnings and notifications with various recipients. Thus, the process description language must have rather flexible control flow expressions to add different types of notifications. These problems will be solved in future developments of the language.

2.2.2 Elements of ProCDeL

Three types of elements are utilized in the process description language: states, events and flow control elements (Fig. 5).

Process description (Fig. 5) has three attributes: process name, schedule for when process may be running and the number of process instances allowed to be running in parallel.

The language introduces three types of process states: *Beginning*, *Ending* and *Intermediate State*, the last two of which are time-controlled states. It means that time control can be done by reaching these states. Time control allows for two types of limits: absolute time (for example, state must be reached by 12:45) and relative distance from other states. The distance may be set in seconds, minutes, hours and days, depending on process specifics. Intermediate states may be identified by unique ids used to specify the acceptable distance between states.

States are connected by *Events*. Each event has event type, arguments and optional event id. Event id may be used in other events or control flow elements to refer to the results returned by the event.

Last group of the elements is *Flow Control* elements. *Cycle* allows to define iterations in the list of items returned as a result of some event. The body of *Cycle* may contain other *States*, *Events* or even *Flow Control* elements. *Load Control* is provided for loading sub-process controls. Those sub-process controls may run synchronously or asynchronously.

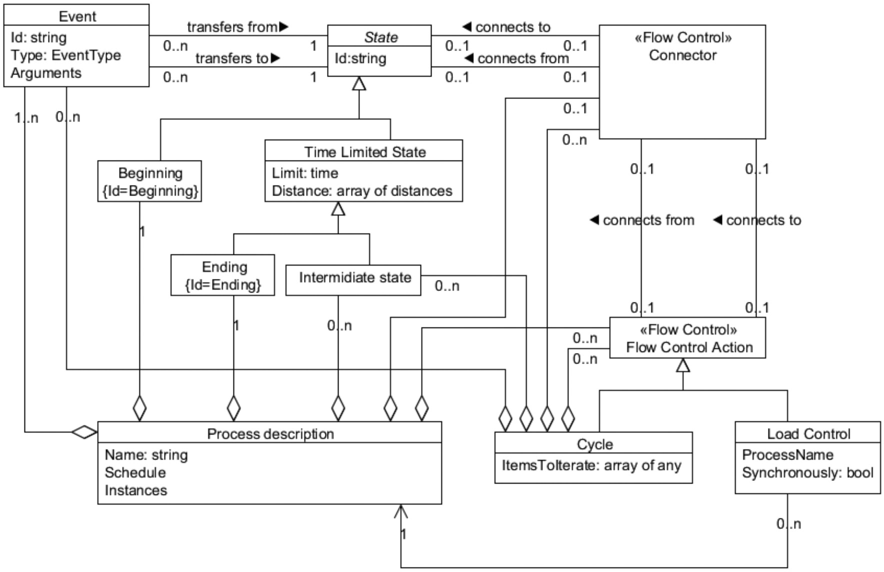

Fig. 5. Elements of the process description language

Conclusion

A new component of a smart technology framework – process control – is being researched. From the process description perspective, the ProCDeL language is ready for the first prototype of the process control mechanism implementation. However, the language must be supplemented with error and warning elements to offer broad potential of information distribution on incorrect and correct process flows. For instance, there can be a process state with two time limits in one process description: when the first limit is reached, the system operators are warned about a possible problem and, when the second limit is reached, error messages are sent.

After the language is supplemented, the first prototype of the process control mechanism will be developed. Most likely this step will make some further changes in the language to make it more usable. The process control mechanism itself is a wide avenue of future research as it is distributed in real time systems. The authors have identified two groups of problems in the process control mechanism:

- problems concerning correct interpretation of event flow by *Central Hub* of the control mechanism;
- technical problems to implement the control mechanism as a reliable distributed real-time system.

The first group of problems is concerned with the algorithms of process control. For instance, there must be an algorithm of how to identify right process control instance if two of the instances from the control instance pool have subscribed for *NewFile* event from the same network resource and one event has arrived. One of the solutions is to move both of processes one step further and keep in mind that one of them could be rolled back. There are other problems concerning event interpretation in this problem group.

The other problem group of mechanism implementation contains more technical problems: heart beat mechanism implementation to determine if all the agents are up and running, time synchronization and tracking of the order of nearly simultaneous events, and other technical problems. Most of these problems are not unique for the process control mechanism and there are solutions in the distributed server system world.

This paper only introduces the concept of automated process control. Further research on concept prototype implementation will be done.

References

1. J. A. Bergstra, P. Klint. The discrete time TOOLBUS – a software coordination architecture. *Science of Computer Programming*, 31, 1998, pp. 205–229.
2. Z. Bičevska, J. Bičevskis. Smart Technologies in Software Life Cycle. In: J. Münch, P. Abrahamsson (eds.), *Product-Focused Software Process Improvement.* 8th International Conference, PROFES 2007, Riga, Latvia, July 2–4, 2007, LNCS, vol. 4589. Berlin/Heidelberg: Springer-Verlag, 2007, pp. 262–272.
3. K. Rauhvargers, J. Bicevskis. Environment Testing Enabled Software – a Step Towards Execution Context Awareness. In: H.-M. Haav, A. Kalja (eds.), *Databases and Information Systems, Selected Papers from the 8th International Baltic Conference*, vol. 187. IOS Press, 2009, pp. 169–179.
4. K. Rauhvargers. On the Implementation of a Meta-Data Driven Self Testing Model. In: T. Hruška, L. Madeyski, M. Ochodek (eds.), *Software Engineering Techniques in Progress*, Brno, Czech Republic, 2008, pp. 153–166.
5. Z. Bičevska, J. Bičevskis. Applying Smart Technologies in Software Development: Automated Version Updating. In: *Scientific Papers of the University of Latvia, Computer Science and Information Technologies*, vol .733, 2008, pp. 24–37. ISSN 1407-2157.
6. E. Diebelis, V. Takeris, J. Bičevskis. Self-testing – new approach to software quality assurance. In: *Proceedings of the 13th East-European Conference on Advances in Databases and Information Systems (ADBIS 2009).* Riga, Latvia, 7–10 September, 2009, pp. 62–77.
7. A. G. Ganek, T. A. Corbi. The dawning of the autonomic computing era. *IBM Systems Journal*, vol. 42, no. 1, 2003, pp. 5–18.
8. R. Sterritt, D. Bustard. Towards an autonomic computing environment. *Proceedings of the 14th International Workshop on Database and Expert Systems Applications (DEXA 2003)*, 2003, pp. 694–698.
9. S. Lightstone. Foundations of Autonomic Computing Development. *Proceedings of the 4th IEEE International Workshop on Engineering of Autonomic and Autonomous Systems*, 2007, pp. 163–171.
10. Z. Bicevska. Applying Smart Technologies: Evaluation of Effectiveness. *Conference Proceedings of the 2nd International Multi-Conference on Engineering and Technological Innovation (IMETI 2009),* Orlando, Florida, USA, July 10–13, 2009.
11. Z. Bičevska, J. Bičevskis. Applying Self-Testing: Advantages and Limitations. In: H.-M. Haav, A. Kalja (eds.), *Databases and Information Systems, Selected Papers from the 8th International Baltic Conference,* vol. 187, IOS Press, 2009, pp. 192–202.
12. J. Ceriņa-Bērziņa, J. Bičevskis, Ģ. Karnītis. Information Systems Development Based on Visual Domain-Specific Language BiLingva. Accepted for publication in the 4th IFIP TC2 Central and East European Conference on Software Engineering Techniques (CEE-SET 2009), Krakow, Poland, October 12–14, 2009.