MASITS – A Tool for Multi-Agent Based Intelligent Tutoring System Development

Chapter · August 2009

DOI: 10.1007/978-3-642-00487-2_52 · Source: DBLP

2 authors:

Egons Lavendelis
Riga Technical University
34 PUBLICATIONS **148** CITATIONS

SEE PROFILE

Janis Grundspenkis
Riga Technical University
136 PUBLICATIONS **867** CITATIONS

SEE PROFILE

MASITS - A Tool for Multi-Agent Based Intelligent Tutoring System Development

Egons Lavendelis and Janis Grundspenkis

Department of Systems Theory and Design
Riga Technical University
Kalku 1, LV-1658, Riga, Latvia
egons.lavendelis@cs.rtu.lv, janis.grundspenkis@cs.rtu.lv

Abstract. Intelligent Tutoring Systems (ITS) have some specific characteristics that must be taken into consideration during the development. However, there are no specific tools for agent based ITS development. This paper proposes such tool named MASITS for multi-agent based ITS development. The tool supports the whole life-cycle of ITS development. It provides an environment to create models needed in all phases of the development. During the analysis phase a goal diagram and a use case diagram is used. The design is divided into two stages, namely external and internal design. The tool provides code generation from the diagrams created during the design. Source code of JADE agents, behaviours and ontology are generated.

Keywords: Agent Oriented Software Engineering, Agent Development Tool, Intelligent Tutoring System

1 Introduction

Extensive research in the agent oriented software engineering field is ongoing. Several agent oriented software engineering methods and methodologies have been proposed, for example, Gaia [1], Prometheus [2], PASSI [3], MaSE [4], and Tropos [5]. However, only few of them provide a CASE tool to be used in agent oriented software development. A few examples of existing tools are the following: agentTool [4], Prometheus Design Tool [6] and Goal Net Designer [7]. Agent implementation environments like JADE [8], JACK [9], MADE [7] exist at their own. One of the main tasks that an agent development tool has to accomplish is to generate implementation code from the design. Prometheus Design Tool generates JACK code and Goal Net Designer generates MADE agents. However, many other tools fail to provide sufficient code generation.

Intelligent agents and multi-agent systems are widely used in Intelligent Tutoring System (ITS) development, for an overview see [10]. Additionally a few multi-agent architectures for ITS development have been built [11, 12, 13]. At the same time, specific tools for multi-agent based ITS development do not exist. However, ITSs have some specific characteristics that must be taken into consideration during the development. Firstly, ITSs are hardly integrated into organisation and they have very few actors. So, organisational and actor modelling

can hardly be used and requirements come only from the system's goals and functionality. Thus, methodologies and tools that use organisational modelling as one of the main techniques are not applicable to ITS. Secondly, ITSs consist of known set of agents and have a well established architecture [10, 13, 14], that should be taken into consideration during the design. However, agent (or agent type or role) definition is major activity in the above mentioned tools. Prometheus Design Tool provides agent types' definition by grouping functionality. AgentTool provides role definition and derives agents and multi-agent architecture from previously defined roles. Thus, a specific ITS development tool can have advantages over general purpose tools by supporting appropriate activities.

Additionally, CASE tool usage during the agent based system development process has the following advantages: (1) the tool enhances diagram drawing by providing the appropriate elements; (2) relationships among different diagrams can be created, which are used for consistency checking and crosscheck; (3) some diagrams can be partly generated from previously created diagrams; (4) source code for agents and other system's elements can be generated from the design. Thus, there are significant advantages in design of ITSs using a specific tool. This paper contains description of a MASITS (Multi-Agent System based Intelligent Tutoring System) tool for ITS development.

All diagrams used in MASITS tool are described in details in Section 3. Section 4 includes an overview of interdiagram links used for consistency checks and crosscheck. Finally, Section 5 gives conclusions and outlines the future work.

2 The MASITS tool

The MASITS tool has been built for multi-agent based intelligent tutoring system development. The tool supports full ITS development life cycle from requirements analysis to implementation. The purpose of the MASITS tool prescribes that the main characteristics of ITSs are taken into consideration. Appropriate requirements analysis techniques have been chosen. ITSs are hardly integrated into any organization and there are very few actors involved. Thus, requirements analysis has to be done using techniques that focus on system's goals and functionality, but not organizational or role analysis. Similarly, during the design phase the results of agent based ITS research are included. The MASITS tool is intended to design multi-agent systems where agents communicate by sending simple messages without any complicated protocols. Besides, the set of agents that build up an ITS is known [10]. The set of agents can be adjusted to meet specific needs, but there is no need for any agent definition activities during the design. Additionally, ITSs can be built using a holonic agent architecture [13]. Thus, a support for holons and their hierarchy is included in the MASITS tool.

Interface of the MASITS tool consists of the following main parts, as it is shown in Figure 1. Main menu (1) contains all functions of the tool. The most frequently used features are included in the main toolbar, too (2). All diagrams created during the development of the system are included in tabs (3). Each tab of the diagram contains a drawing toolbar (4) and a page for diagram drawing (5).

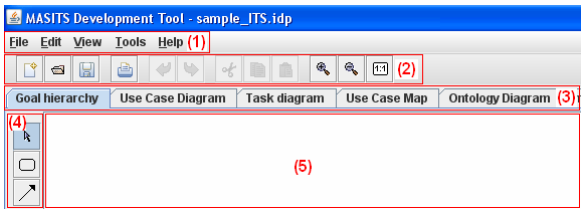

Fig. 1. Main parts of the interface

The first phase that MASITS tool supports is requirements analysis. It is done using goal modelling and use case modelling. The goal modelling is done first, because goals are used in use case creation. The second phase is design that is divided into two stages, namely, external and internal design of agents. During the external design agent functionality and interactions among agents are specified. During the internal design the internal structure of agents is specified, i.e., it is defined how agents achieve functionality specified during the external design.

Of course, design tools which can be used to specify systems that are implemented on different platforms have wider usage. However, transformation from design concepts to agent implementation platform concepts is almost unique for each combination of design concept set and agent platform [14]. Thus, we have chosen to use a set of JADE agent development framework's concepts [8] already during the design phase. JADE was chosen, because it provides simple way to create Java agents and organise their interactions. Moreover, Java classes of JADE agents can be easily generated from the design elements. Agent communication in JADE is organised using predicates from the domain ontology. Agents are Java classes and their actions are defined as behaviours. So, main implementation elements are ontology, agent and behaviour classes. Additionally to these elements, a batch file to start the system is needed. The batch file includes deployment details of the system, which are specified in the deployment diagram. The batch file is generated automatically from the deployment diagram. So, the MASITS tool supports implementation and deployment phases, too.

3 Diagrams used in MASITS tool

The ITS development using MASITS tool consists of creation of a set of diagrams. The following diagrams are included: a goal diagram, a system level use case diagram, a task-agent diagram, a use case map, an interaction diagram, a ontology diagram, an agent's internal view and a holon hierarchy diagram. All diagrams included in the MASITS tool and dependencies among them are shown in Figure 2. Sequential creation of diagrams is denoted with an arrow, crosscheck is denoted with a dashed line.

3.1. Goal diagram

The goal diagram depicts goals and hierarchical relationships among them. Two types of relationships are distinguished: (1) AND decomposition (simple line),

meaning that all subgoals have to be achieved to achieve the higher level goal. (2) OR decomposition (black circle and connecting lines), meaning that at least one of the subgoals has to be achieved to achieve the higher level goal. An example of a goal diagram is shown in Figure 3.

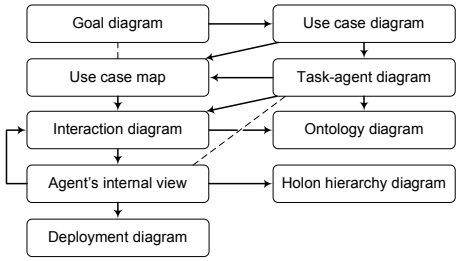

Fig. 2. Diagrams included in MASITS tool and dependencies among them

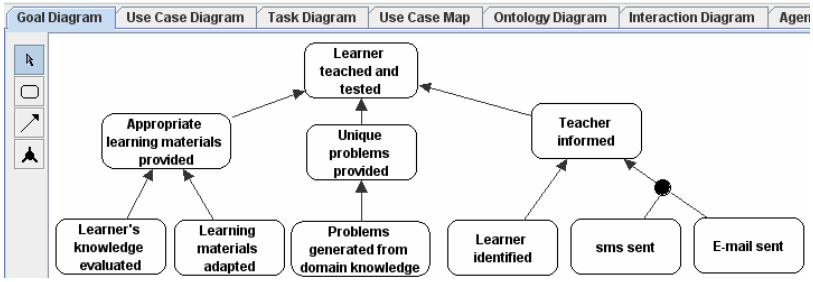

Fig. 3. Goal diagram

3.2. Use case diagram

The second diagram included in MASITS tool is well known use case diagram. It includes system level use cases and actors interacting with the system. At first, all actors are identified and included in the diagram. Then use cases and their descriptions are created corresponding to actions that have to be done to accomplish the lower level goals of the goal diagram. Interdiagram links to goals that are supported by use cases are created. A well known UML use case notation is used [16]. An example of a use case diagram is shown in Figure 4.

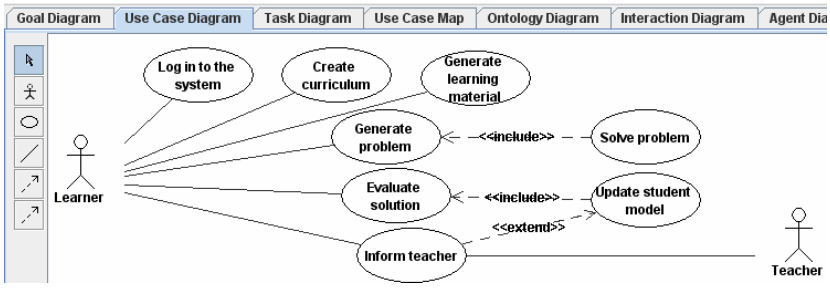

Fig. 4. Use case diagram

3.3. Task-agent diagram

The task-agent diagram is a hierarchy of tasks that the system has to accomplish. The diagram contains information about task allocation to agents, too. Name of agent that is responsible for each task is added to the task's node.

The first step of task-agent diagram creation is task decomposition resulting in a task hierarchy. The task-agent diagram consists of one or more task hierarchies. The task hierarchy is created by defining tasks corresponding to use case scenario steps, i.e., a task is created for each step of the use case scenario. Task hierarchies are created by linking up tasks that can be assigned to the same agents not by corresponding use cases or goals. So, the structure of task hierarchy is different from the goal hierarchy. After finishing task decomposition, tasks are allocated to agents using basic principles of ITS architecture [10, 13, 17]. An example of a task-agent diagram is shown in Figure 5.

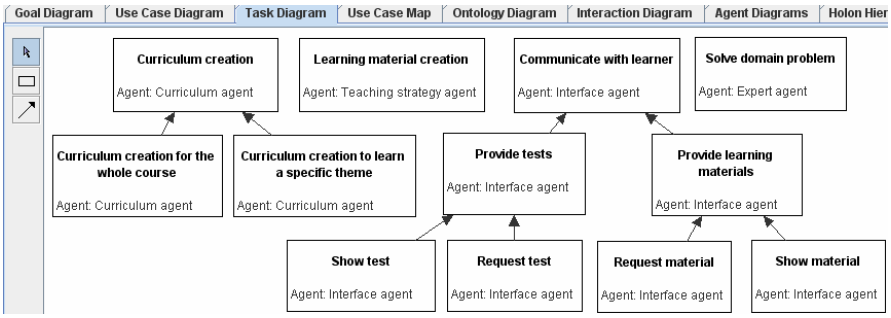

Fig. 5. Task-agent diagram

3.4. Use case map

In object-oriented approach use case maps are used to model the control passing path during the execution of use case (for details, see [18]). In multi-agent design use case maps include agents, their tasks and message paths among them. A use case map represents the use case scenario explicitly showing interactions among agents. Each link between two tasks can be considered as a message between corresponding agents. Thus, after creating the use case map interactions among agents can be easily specified just adding message content.

Use case map creation is the first step of agent interaction design. Interaction is designed for each pair of interacting agents. Use case maps are created for pairs of agents whose interaction is too complicated to be specified directly in interaction diagram. An example of a use case map is shown in Figure 6.

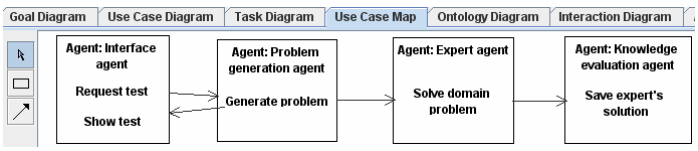

Fig. 6. Use case map for use case "Generate problem"

3.5. Ontology diagram

The ontology diagram is a model of the problem domain. Concepts of the problem domain and predicates used in agent interaction are described. The ontology diagram is a modified version of the class diagram. In fact, the ontology diagram is a class hierarchy. Two superclasses are used: (1) Concept – subclasses of this class are domain concepts; (2) Predicate – subclasses of this class are predicates used in agent interactions as message contents.

All other classes are subclasses of the above-mentioned two superclasses. Each class has attributes which have name, cardinality and type. Attribute's type may be a primitive type, Java class or a concept defined in the ontology diagram. Predicates are not allowed as attribute types. Initial ontology diagram usually is created during interaction design by adding predicates used in agent interaction specification. Concepts needed to define these predicates are added during interaction design, too. During agents' internal design the ontology diagram is refined by adding concepts and predicates used in agents' internal view. An example of the ontology diagram is shown in Figure 7.

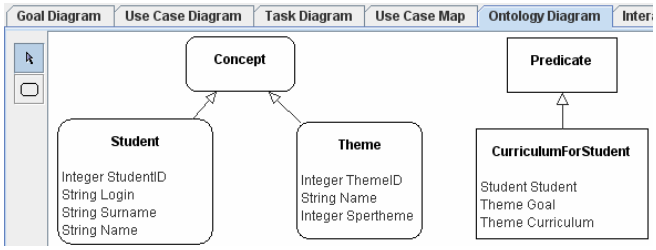

Fig. 7. Ontology diagram

3.6. Interaction diagram

The interaction diagram specifies interactions among agents. The MASITS tool allows specifying interactions by messages sent among agents. Protocols are not included, because the tool is designed for agent based ITS development and our research has shown that complicated protocols are not widely used in this domain.

The diagram consists of two main elements, namely, agents and messages. Agents are denoted as rounded rectangles. Messages are denoted as labelled links with full arrows between agent vertexes. Labels of message links are names of predicates. The diagram includes interaction between an interface agent and a user, too. Thus, a few additional elements are added: a user (denoted as an actor), events monitored by an agent (dashed line) and methods of interface called by interface agent (line with simple arrow).

The main interaction diagram is created to specify interactions among higher level agents. Additionally, the interaction diagram is created for each holon. Interaction diagrams for holons may include sets of typical agents [17] and directory facilitator agents. These elements are introduced to allow specifying open holons. An example of an interaction diagram is shown in Figure 8.

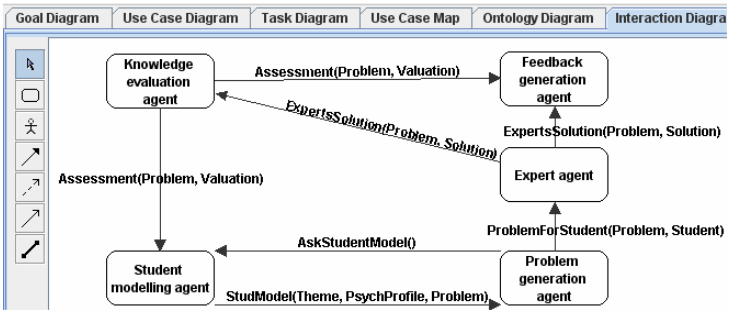

Fig. 7. Fragment of the interaction diagram

3.7. Agent's internal view

The agent's internal view specifies design of agent's internal structure. An internal view is created for each agent. It consists of the following elements:

- An agent diagram, including agent's perceptions, messages sent by an agent, agent's actions and links among these elements.
- Perception (message and event) processing rules.
- Startup rules describing actions that are performed by an agent during the startup.
- A list of agent's beliefs. Agents' beliefs are specified as pairs of belief's type and name: <Type>, <Name>. Type can be either primitive type used in Java language, Java class or ontology class (predicate or concept) from ontology diagram.
- A list of agent's actions specifying implementation details. Actions are described in table containing the following columns: name of the action, name of corresponding behaviour class, type of the action (one-shot, cyclic, timed, etc.), inner (implemented as inner class in agent's class) or outer (implemented as a class in the same package as agent) behaviour class.

The agent diagram includes the following elements: (1) all messages sent and received by agent. The agent diagram of holon's head includes messages designed in higher level interaction diagram and also messages sent to and received from body agents of the holon; (2) agent's actions and interactions among them. So, agent's plans are modelled; (3) perceived events from the environment; (4) agent's actions to user interface.

The agent diagram specifies an agent as its actions and interactions among them. Actions mainly are initiated reacting on received messages or perceptions. Additionally actions initiated by time can be used (denoted by a ticker). Each social agent has at least two actions – message sending and receiving. These actions are not depicted in the agent diagram. Instead, received and sent messages have message contacts. Agent can have four types of contacts: message receiving, message sending, event receiving and action to external environment contacts. Each message, event or action is added to its own contact.

Perception processing and startup rules are designed as IF-THEN rules: IF <Condition> THEN <Action> ELSE <Action>. The following templates can be used as condition: Received (check if particular predicate is received), Compare and True. Additionally, Boolean operators (AND, OR, NOT) can be applied to conditions. Templates Action, Belief Set (sets value of one of the agent's beliefs), Action conjunction (sequence of actions) and IF-THEN rule can be used as action.

It is advisable to perform internal design of each agent iteratively. Each iteration can include design elements corresponding to either a set of perceptions or a set of actions. For details and an example of agent's internal view, see [17].

3.8. Holon hierarchy diagram

The holon hierarchy diagram shows holons that exist in the system and a hierarchy among them. This diagram is created automatically. Holon nodes are added to the hierarchy when the user creates a holon to implement an agent. The only purpose of the diagram is to summarize the structure of the multi-agent system. An example of the holon hierarchy diagram is shown in Figure 8.

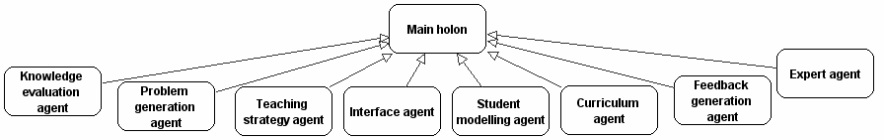

Fig. 8. The holon hierarchy diagram

3.9. Deployment diagram

The deployment diagram specifies how the designed agents are used in the system. The diagram consists of containers that are used to run the system and agent instances that are launched during the system startup. Each container is a JADE container that is started during the startup of the system. One of the containers is the main JADE container, included in all deployment diagrams. Interactions among containers are specified, too. Agents are defined in the JADE containers. Each agent has a name and agent class defined in the agent diagram. In fact, agents designed in the interaction diagrams and agent internal views are agent classes that can have multiple instances. Concrete instances are specified in the deployment diagram.

4 Interdiagram links

One of the main advantages of MASITS tool is the introduced concept of interdiagram links that are links among elements of two different diagrams. Firstly, interdiagram links are used to ensure the diagram consistency. Majority of them are created if an element is used as a part of another element. For example, an interdiagram link is created from predicate vertex to communication links, where the predicate is used as message content.

MASITS tool allows editing each element only in one place. All other occurrences of the element are changed automatically to ensure consistency. For example, agent's name can be changed only in the interaction diagram. If it is changed, the tool changes the name of the agent in the agent diagram, the holon hierarchy diagram, the task diagram and other interaction diagrams. Elements can be deleted in the same way as changed. However, if an element is deleted, significant parts of other models can be affected. Two choices are available. The tool just deletes dependant elements if the deleted element has the same semantics as the dependant elements. Deletion of the element is restricted if the dependant part is a significant part of other diagrams with different semantics. Due to the space limitations, further details about consistency checking links are omitted.

Secondly, interdiagram links are used to specify semantic dependencies between elements of different diagrams, too. Such links are created by the user and are used during the crosscheck among diagrams. The MASITS tool contains interface to define such interdiagram links and to see defined links and unlinked elements. The unlinked elements can be used to check, if all elements of one diagram are supported by elements in another diagram. Such approach is used in the MASITS tool four times. Firstly, links between use cases and goals are created. Unlinked goals show which goals need use cases to be created. Secondly, links between goals and tasks are created to show which tasks support which goals. This kind of links is used during the crosscheck between goal and task diagrams. Any unsupported goal indicates that defined tasks are not sufficient to achieve system's goals. Thirdly, paths from use case maps are linked to messages in the main interaction diagram to ensure that every link from paths in use case maps have corresponding messages in the main interaction diagrams. Finally, each agent must have actions to realize all tasks assigned to it in the task diagram. Thus, interdiagram links are created among tasks and agents' actions.

5 Conclusions and future work

A specific tool for multi-agent based ITS development is proposed. The main advantages of the proposed tool are the following. The tool supports the whole life cycle of ITS development, thus there is no need for any additional tool. The very popular agent development environment (JADE) is used for implementation. The tool provides consistency checking and important crosschecks during the development of system, helping to find possible errors. Finally, code generation is done automatically from the diagrams created during the design phase.

Regardless of the MASITS tool is developed for specific purpose, it can be used to develop other multi-agent systems, especially those that have similar characteristics with multi-agent based ITSs.

Our future work is to develop a full case study of ITS development using the MASITS tool. After finishing the case study it will be possible to evaluate the tool in more details. However, the abovementioned advantages make the tool to be a promising one. The main direction of our research is to formulate a full lifecycle methodology for agent based ITS development supported by the MASITS tool.

References

1. Wooldridge, M., Jennings, N.R., Kinny, D.: The Gaia methodology for agent-oriented analysis and design. Journal of Autonomous Agents and Multi-Agent Systems. (2000)
2. Padgham, L., Winikoff, M.: Prometheus: A Methodology for Developing Intelligent Agents, Proceedings of the Third International Workshop on AgentOriented Software Engineering, at AAMAS 2002. (2002)
3. Burrafato, P., Cossentino, M.: Designing a multi-agent solution for a bookstore with the PASSI methodology. Fourth International Bi-Conference Workshop on Agent-Oriented Information Systems at CAiSE'02. (2002)
4. DeLoach, S.: Analysis and Design Using MaSE and agentTool. Proceedings of the 12th Midwest Artificial Intelligence and Cognitive Science Conference, pp.1-7. (2001)
5. Giunchiglia, F., Mylopoulos, J., Perini, A.: The Tropos Software Development Methodology: Processes, Models and Diagrams. Proceedings of the First International Joint Conference on Autonomous Agents and Multiagent Systems. pp. 35-36. (2002)
6. Padgham, L., Thangarajah, J., Winikoff, M.: Tool Support for Agent Development Using the Prometheus Methodology. Fifth International Conference on Quality Software (QSIC'05), pp. 383-388. (2005)
7. Yu H., Shen Z., Miao C. Intelligent Software Agent Design Tool Using Goal Net Methodology. Proceedings of the 2007 IEEE/WIC/ACM International Conference on Intelligent Agent Technology. pp. 43-46. (2007)
8. JADE Home Page. http://jade.tilab.com/ (Last visited: 10.06.07).
9. Howden, N., Rönnquist, R., Hodgson, A., Lucas, A.: JACK Intelligent Agents - Summary of an Agent Infrastructure. Proceedings of the Fifth International Conference on Autonomous Agents (2001)
10. Grundspenkis, J., Anohina, A.: Agents in Intelligent Tutoring Systems: State of the Art. Scientific Proceedings of Riga Technical University "Computer Science. Applied Computer Systems", 5th series, Vol.22, Riga. pp.110-121. (2005)
11. Capuano, N., et al. A Multi-Agent Architecture for Intelligent Tutoring. Proceedings of the International Conference on Advances in Infrastructure for Electronic Business, Science, and Education on the Internet, SSGRR 2000, (2000).
12. Webber C., Pesty S. A two-level multi-agent architecture for a distance learning environment. In: ITS 2002/Workshop on Architectures and Methodologies for Building Agent-based Learning Environments, E. de Barros Costa, pp.26-38 (2002)
13. Lavendelis, E., Grundspenkis, J.: Open Holonic Multi-Agent Architecture for Intelligent Tutoring System Development. Proceedings of IADIS International Conference "Intelligent Systems and Agents 2008", pp. 100-108. (2008)
14. Smith, A.: Intelligent Tutoring Systems: personal notes. 1998. http://www.cs. mdx.ac.uk/staffpages/serengul/ table.of.contents.htm (Last visited 18.04.2005)
15. Massonet, P., Deville, Y., Neve, C.: From AOSE methodology to agent implementation. Proceedings of the First International Joint Conference on Autonomous Agents and Multiagent Systems, Bologna, Italy. pp. 27 – 34. (2002)
16. OMG UML Superstructure 2.1.2. Available: http://www.omg.org/docs/formal/07-11-02.pdf (Last visited: 03.10.2008).
17. Lavendelis, E., Grundspenkis, J.: Design of Multi-Agent Based Intelligent Tutoring Systems. Scientific Proceedings of Riga Technical University "Computer Science. Applied Computer Systems", RTU Publishing, Riga, (2009) (accepted for publishing).
18. Buhr, R.J.A., Elammari, M., Gray, T., Mankovski, S.: Applying Use Case Maps to Multi-Agent Systems: A Feature Interaction Example. Proceedings of the Thirty-First Hawaii International Conference on System Sciences, pp. 171-179. (1998)