ISBN: 978-972-8924-87-4 © 2009 IADIS

MASITS – A MULTI-AGENT BASED INTELLIGENT TUTORING SYSTEM DEVELOPMENT METHODOLOGY

Egons Lavendelis, Janis Grundspenkis
Riga Technical University, Department of Systems Theory and Design
Kalku 1, LV-1658, Riga, Latvia

ABSTRACT

During the last decades intelligent agents integrated in multi-agent systems are accepted as a suitable technology to implement mechanisms of Intelligent Tutoring Systems (ITS), so facilitating ITS development. Still, no adequate methodological support is available for agent based ITS development. Regardless of the fact that general agent-oriented software engineering methodologies are the most suitable for this purpose, ITSs have specific characteristics that need to be taken into consideration during the development process. Additionally, ITSs have their own field of research with important knowledge gained, that is useful during the ITS development. The paper presents the specific ITS development methodology named MASITS to support agent based ITS development. The methodology integrates results of ITS research into the most appropriate techniques used in agent-oriented software engineering methodologies. The paper presents also the corresponding tool for support of the methodology and gives a brief description of case study of the methodology and the tool.

KEYWORDS

Agent-Oriented Software Engineering Methodology, Intelligent Tutoring System, Multi-Agent System Application

1. INTRODUCTION

During the last decades a large number of Intelligent Tutoring System (ITS) prototypes have been built, for example, FLUTE (Devedzic et al., 2000), ABITS (Capuano et al., 2000), Passive Voice Tutor (Virvou and Tsiriga, 2001), AGT (Matsuda & VanLehn, 2005), Slide Tutor (Crowley & Medvedeva, 2005) and Ines (Hospers et al., 2003). However, the ITS development is still a complex process, because functionality of ITSs is intelligent and complicated. At the same time sufficient methodological support to manage complexity during the ITS development is not available. Agents and Multi-Agent Systems (MAS) have become one of the most popular technologies to implement ITSs during the last years. Well known examples of agent based ITSs are the ITS for enhancing e-learning (Gascuena & Fernández-Caballero, 2005), ABITS and Ines. Agents are so popular, because they are autonomous, reactive, proactive, and they are capable to plan, reason and make decisions (Grundspenkis & Anohina, 2005). Additionally, MASs are open. These characteristics of agents help to implement intelligent mechanisms into ITSs. Agents allow decomposition of large monolithic modules of ITSs. Consequently, the architecture of agent based ITS may consist of smaller entities which are easier to develop then large-scale modules. Thus, agents help to reduce complexity of ITS development process. However, development of agent based ITS in an ad-hoc manner without usage of any methodology can be done only as a research and not as an industrial software development. This is one of the main obstacles for agent based ITSs to become more widespread.

To facilitate adoption of Agent-Oriented Software Engineering (AOSE) by software developers a large number of agent-oriented software engineering methodologies have been developed during the last decades. The well known examples of AOSE methodologies are Gaia (Zambonelli et al., 2005), Prometheus (Winikoff & Padgham, 2004), MaSE (De Loach, 2001), Tropos (Giunchiglia et al., 2001), MAS-CommonKADS (Iglesias C. et al., 1998) and RAP (Taveter & Wagner, 2005). Two types of methodologies can be distinguished. First, general purpose agent-oriented methodologies aim to support development of any agent-oriented software. Gaia is one of the most popular examples of this type. Second, specific purpose methodologies for some specific agent-oriented software are proposed, for example, ADELFE methodology

for adaptive MAS development (Picard & Gleizes, 2004). General purpose methodologies aim to support as many different agent systems as possible by providing high level techniques that support different agents. Contrary, specific purpose methodologies provide particular techniques that can be used more effective to develop systems with specific characteristics. In case if the merely distinguishing characteristic is the chosen agent development platform, only detailed design, implementation and deployment phases can be adopted to the platform. Such approach is used in the Prometheus methodology. Additionally, a methodology for systems in specific domain can include appropriate techniques for analysis and early design. Thus, specific methodologies can facilitate development of respective agent-oriented systems. At the same time our study of large number of publications reveals that a specific methodology for agent based ITS does not exist. This fact motivated an effort to develop such methodology called MASITS which is described in this paper.

The remainder of the paper is organized as follows. In Section 2 distinguishing characteristics of ITS and their influence on the development process are given. Section 3 includes description of the MASITS methodology. Section 4 describes the MASITS tool that supports ITS development using the MASITS methodology. Section 5 is dedicated to a case study of the MASITS methodology and tool. Section 6 concludes the paper.

2. SPECIFIC CHARACTERISTICS OF AGENT-BASED INTELLIGENT TUTORING SYSTEM DEVELOPMENT

The ITSs aim is simulation of a human tutor during the learning process. To do it, ITSs have to generate curriculum for particular learner to learn the course, realize different teaching strategies, generate tasks, problems and questions to evaluate learner's knowledge as well as give feedback to explain learner's mistakes and offer appropriate remedial actions. All these activities have to be done in adaptive manner to implement one-to-one tutoring and meet needs of different learners. To carry out these activities ITSs use three types of knowledge: domain knowledge, knowledge about a learner and pedagogical knowledge (Grundspenkis & Anohina, 2005). The ITS architecture consists of three modules corresponding to the knowledge types used: an expert module, a tutoring module and a student diagnosis module. Additionally a communication module is used to communicate with a learner. Grundspenkis & Anohina (2005) offer a set of agents that are used to implement all modules of ITS: an interface and animated pedagogical agent for communication module; one or more teaching strategy agents, a curriculum agent, a feedback and explanation agent for pedagogical module; a psychological agent, a knowledge evaluation agent, a cognitive diagnosis agent and an interaction registering agent for student diagnosis module; and one or more expert agents for expert module. Complementary Lavendelis and Grundspenkis (2008) offer the holonic multi-agent architecture for agent based ITS implementation. Thus, different ITS researches have provided significant knowledge that can be used during the development process. However, if general purpose methodology is used it is hard to integrate the results of ITS research into the development process.

Agent based ITSs are agent-oriented systems with specific characteristics that should be taken into consideration during the development process. The main characteristics are the following. Firstly, ITSs are weakly integrated into organization and have very few types of users (learners, teachers and administrators if latter one needed). Thus, any organizational analysis techniques are not applicable during ITS requirements analysis. Agents and their tasks are treated as requirements in many AOSE methodologies, but it is reasonable only if agents are created to represent someone. It is not the case considering the ITS. So, ITS requirements analysis has to be based on the functional requirements and not organizational structure or agents. Secondly, a set of agents, used for implementation of ITS, is well known. Therefore, instead of agent definition during the design the already existing set of agents may be used and modified if needed. Thirdly, majority of agents, that are used to implement an ITS, are reactive agents, whose development must be as simple as possible. At the same time some agents have to be proactive, for example, a feedback agent has to interfere if the learner has difficulties to solve a problem. Moreover, all intelligent mechanisms needed in an ITS have to be implemented in agents. As a result, support of heterogeneous agents is needed. Fourthly, MASs that are used to implement ITS, are cooperative, they have to achieve a common goal – to teach the learner. Fifthly, our research has shown that the agents for the ITS development communicate using single messages and no explicit protocols are used. Thus, the most appropriate form to design interactions among

agents is single messages, not protocols. Sixthly, ITSs can be effectively developed using holonic MASs (Lavendelis & Grundspenkis, 2008).

So, a specific methodology for agent based ITS development can address the main characteristics of ITS in a more appropriate way in comparison with general ones and as a consequence be more effective. This motivates working out a specific methodology for the ITS development. The specific ITS development methodology has to:

- Integrate knowledge from ITS research, like known set of agents and architecture;
- Include suitable techniques to support main characteristics of ITS and exclude techniques that deal with aspects of agents that are not important in agent based ITS.

3. MASITS METHODOLOGY

The MASITS (MAS for ITS) is a full life cycle methodology for agent based ITS development. The MASITS methodology includes the most important results of ITS development research and AOSE methodologies. The most appropriate techniques from existing AOSE methodologies are used during steps where existing techniques allow to integrate specific knowledge for the ITS development. Additionally new techniques are introduced in steps where known techniques do not allow to integrate ITS knowledge.

The process of ITS development consists of the following phases: analysis, design (divided into two stages: external and internal design), implementation, testing, deployment and maintenance. These phases are presented sequentially, although the development process is iterative. Iterations are used both inside the phases and across different phases. A developer of the system is allowed to return to any previous phase and refine the previously created models. Phases of the development process and steps included in these phases are shown in Figure 1. The remainder of Section 3 gives a brief overview of steps done during all phases of development, for details, see (Lavendelis & Grundspenkis, 2009a) and (Lavendelis & Grundspenkis, 2009b).

3.1 Analysis Phase

During the analysis phase two consecutive steps are performed. The first step is the *goal modelling* resulting in the goal diagram that depicts the goal hierarchy of the system. At first, the higher level goals of the system are defined. Then each goal is decomposed into subgoals using "AND" and "OR" decompositions. Goals are decomposed until the lower level goals can be achieved by simple actions. A goal description is written for each goal, including an actor that needs the goal to be achieved, a brief description of the goal and conditions that must be met to achieve the goal. Goals identified during this step are used to define corresponding use cases and to crosscheck models created during the later phases against goals.

At the second step the *use case model* is created. During the goal modelling initial actors have been defined, by specifying actors that need to achieve goals. At the beginning of the use case modelling the set of actors is refined by adding all actors that participate in use cases, but do not have corresponding goals. When all actors are defined, use cases are created corresponding to the lower level goals of the goal hierarchy. One or more use cases are created to achieve each lower level goal. Use cases are added to the use case diagram and the use case description is created for each use case. The main part of the description is the use case scenario, denoting consecutive steps that have to be executed during the use case. The consecutive scenario steps are used in design phase to define tasks done by agents and interaction among agents. At the end of the use case modelling use cases are checked against goals by creating interdiagram relationships from goals to use cases that are defined to achieve the goal. If any lower level goal is not linked to any use case, then the goal is not supported and use cases have to be refined.

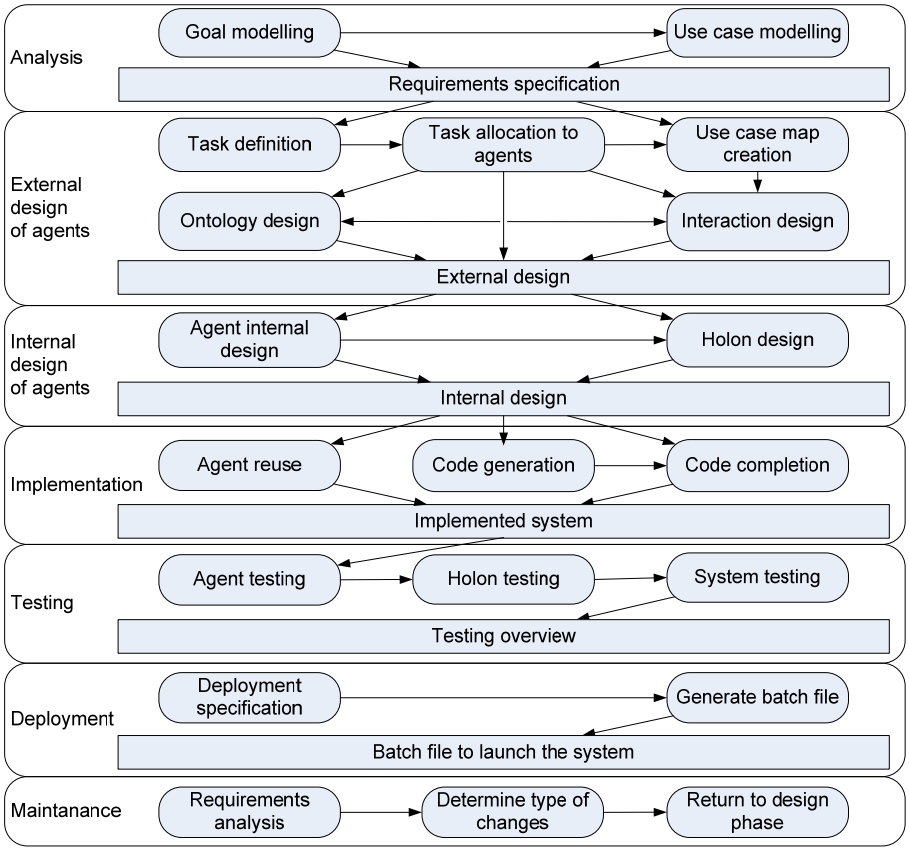

Figure 1. Structure of MASITS methodology

3.2 External Design of Agents

The ITS design phase has been divided into two stages – the external and the internal design of agents. During the first one (the external design) agents are designed in terms of their functionalities and interactions among agents. The external design of agents answers the question what agents have to do. This stage consists of the following steps: the task definition, the task allocation to agents, the interaction design and the ontology design.

The first step of the external design is the task definition. Tasks are defined according to the steps of use case scenarios. A task is defined either according to a single step or a few consecutive steps. After the task definition, a crosscheck of tasks against goals is done. It is carried out by creating relationships among goals and tasks that are accomplished to achieve the goal. After such relationships are created any unlinked lower level goal indicates that the goal is not achieved by the previously created tasks and the tasks must be refined.

When a set of tasks that is sufficient to achieve all goals is created, the second step is started. During this step tasks are organized into the hierarchy and assigned to agents. The task hierarchy is created by linking together tasks that are accomplished by the same agent resulting in the task diagram. So, the hierarchy in the task diagram is different from the goal hierarchy. The MASITS methodology contains informal rules describing tasks that must be allocated to agents from the set of agents used to implement the ITS. For example, the following tasks are allocated to the student modelling agent: (1) learner's level of knowledge and skills model building tasks; (2) learner's psychological characteristics modelling tasks; (3) learner's interaction with the system registering tasks; (4) determining tasks of learner's mistakes and their causes; (5) full student model creation task (for details, see Lavendelis & Grundspenkis, 2009b). Designer only has to decide which description of tasks is the most appropriate for each task. The task-agent diagram is created as a result of this activity. This diagram contains a hierarchy of tasks and an agent responsible for each task is specified.

After defining tasks and assigning them to agents interaction among agents has to be designed. The goal of interaction design is to create the interaction diagram. Interactions among agents are specified in terms of sent messages. The interaction diagram consists of agents, actors representing users and links among them. Links among two agents denote messages and are labelled by predicates sent in the messages. Links from an actor to an agent denote events perceived by an agent and links from an agent to an actor denote methods of user interface that are invoked to communicate with an actor. Interactions are designed according to use case scenarios. If tasks that correspond to two consecutive steps in the use case scenario are realized by different agents, then a message has to be sent after completion of the first step of the use case scenario. In most cases a designer is able to define interactions by using use case scenarios and the task-agent diagram. However, these artefacts do not explicitly show interactions among agents and identification of interactions that are needed to implement complicated use cases is difficult. Thus, use case maps are created for most complicated use cases before the interaction design. Use case maps denote control passing among entities. In AOSE they show sequence of tasks done by agents explicitly showing required interactions among agents: each link among tasks that are assigned to different agents means that a message among these agents must be sent. Thus, after creating the use case map a designer merely has to add message content.

During the interaction design message contents are designed using predicates from the domain ontology. So, the domain ontology has to be designed to specify message contents. The domain ontology in the MASITS methodology is a class diagram with two superclasses: "*Concept*" and "*Predicate*". All other classes are subclasses of these two superclasses. The domain ontology is used in communication among agents and to represent agents' beliefs.

3.3 Internal Design of Agents

During the internal design of agents, the internal structure of agents must be defined. This stage must give an answer to the question, how agents achieve their behaviour specified during the external design. Two approaches of external design exist in AOSE methodologies. First, agents are designed using high level design concepts. These methodologies (for example, Gaia) provide designs that are independent from the agent development platform and thus have wider usage. However, transformation from high level design concepts to implementation ones is almost unique for each combination of methodology and agent development platform (Masonet, 2002). Second, agents are designed using low level concepts that correspond to the selected implementation platform. This approach enables easy transition to implementation phase and easy code generation. This is why it is followed in the MASITS methodology. JADE (Java Agent DEvelopment Framework, http://jade.tilab.com/) has been selected as the agent development platform. Therefore, agents must be designed in terms of messages sent and received, events perceived and actions done by agents.

In the MASITS methodology agents are designed in the internal view of agents that is created iteratively by adding all elements corresponding to single action or single incoming message (event).The agent's internal view consists of:

- The agent diagram, showing incoming and outgoing interactions (denoted by links and contacts), actions performed by agents and relationships among them denoting sequence of actions (as it is shown in Figure 2).
- Message receiving and start-up IF-THEN rules. Message receiving rules specify agent's reaction to the received messages. Start-up rules specify agent's actions during the start-up.
- A list of agent's beliefs that are specified by name and type of the belief.
- Implementation details of actions. In JADE actions are implemented as behaviours. Thus, the name of the behaviour class and the type of the JADE behaviour are specified for every action.

As shown in (Lavendelis & Grundspenkis, 2008), holonic agents have the following advantages in ITS development. Firstly, holonic architecture consists of small-scale agents that are easier implementable and reusable than large-scale agents. Secondly, holonic architecture enables easier change implementation allowing to modify the system to meet the needs of different courses. Thus, the MASITS methodology supports design of holonic MASs where each agent can be implemented as lower level MAS. If an agent is implemented as a holon, interaction design of the lower level MAS and internal design of each holon's agent carried out done using above mentioned techniques.

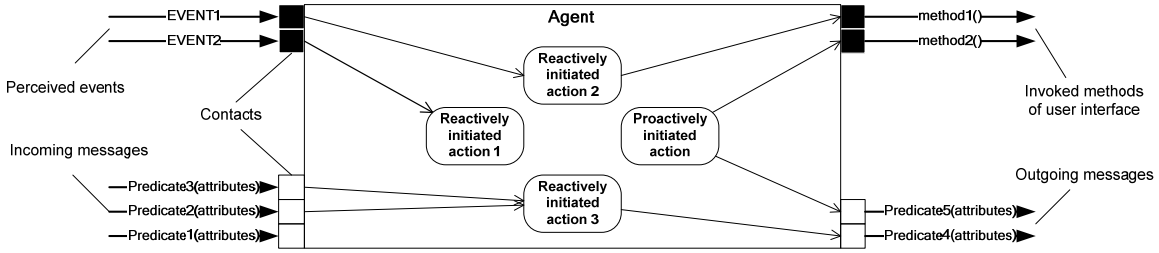

Figure 2. The agent diagram

3.4 Implementation, Testing, Deployment, and Maintenance

The holonic multi-agent architecture supported by the MASITS and described in (Lavendelis & Grundspenkis, 2008) consists of small scale agents that can be reused in systems with similar functionality. For example, if there is an agent that is capable to generate some kind of problems, then it can be reused in all systems that need to generate problems of that kind. The implementation phase consists of the following three steps. During the first step the abovementioned small scale agents are reused from previous projects. To facilitate the reuse process developers are advised to create a library consisting of all previously created lower level agents. The second step is code generation using the MASITS tool for agents that can not be reused from previously created systems. All classes of domain ontology, agents and their behaviours are generated. During the third step a programmer has to complete *action()* methods of behaviour classes generated by the system and create user interface of the system.

During the testing phase JADE test suite (Cortese et al., 2005) is used to develop and execute tests. Three steps of testing are realized. Firstly, separate agents are tested by creating a test group for each agent and executing it. Secondly, each holon is tested as an integral entity, i.e., as an agent. Thirdly, the functionality of the whole system is tested using traditional testing methods.

Deployment is accomplished by using modified version of the UML deployment diagram that shows JADE containers and agent instances deployed in each container. Agent instances are defined by names of agents and a class that is instantiated by an agent. The deployment diagram is used by the MASITS tool to generate a batch file that starts the JADE platform, containers, and all agents' instances.

The MASITS methodology supports change implementation into the system during the maintenance phase. The first step of change implementation is requirements analysis and identification of changes' types. Three types of changes are distinguished: (1) modification of functionality corresponding to an open holon. In such case changes are implemented by adding/removing agents to the holon; (2) changes that have to be made inside one holon, but existing code has to be changed. In this case only one holon is redesigned and changed; (3) changes that must be made in higher level holon. The holonic multi-agent architecture for the ITS development does not help to implement this type of changes and a whole system must be redesigned.

4. TOOL SUPPORT

The MASITS methodology is supported by the MASITS tool. The tool supports development of all diagrams that are used in the MASITS methodology. The user interface of the tool is shown in Figure 3.

The tool enforces correct notation of each diagram and a consistency among all diagrams. A consistency is ensured by interdiagram relationships among elements with the same semantics in different diagrams. Each design element that is included in different diagrams can be changed only in one diagram and in case of any changes all other instances of that element are changed automatically. For example, an agent can be deleted or its name can be changed only in the interaction diagram. In case of any changes all other occurrences of the agent are changed. This enables an iterative approach during the development: a developer is allowed to return to previously created diagrams and refine them. When any diagram is refined, all later diagrams are changed according to changes made in the refined diagram.

Above mentioned interdiagram relationships are created automatically. Additional interdiagram relationships are used for crosschecks among models. These relationships are created manually using specific

user interface in the MASITS tool. Manually created interdiagram relationships are defined to analyse how elements of one diagram are supported by other diagrams.

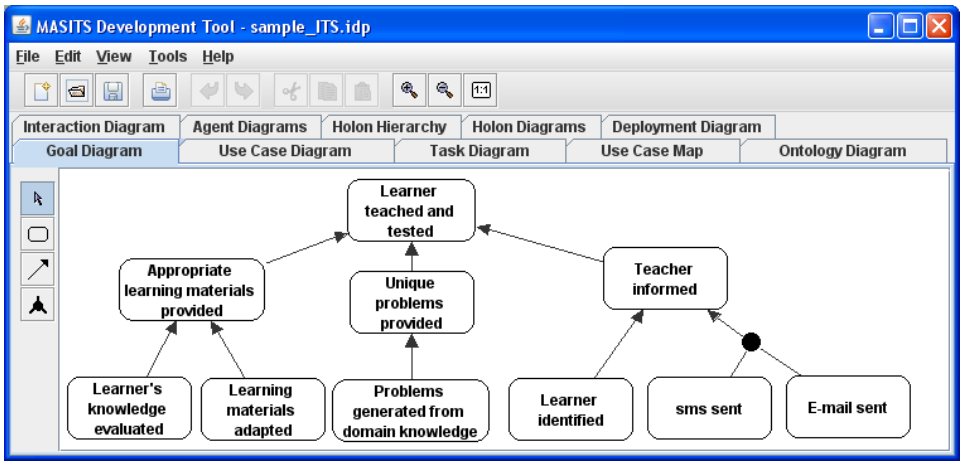

Figure 3. MASITS tool

The MASITS tool helps developers by automatically generating parts of diagrams that can be inferred from other diagrams. For example, incoming and outgoing messages in the agent's diagram are generated when a message is added to the interaction diagram. Additionally, the holon hierarchy diagram showing all holons in the system is completely generated from holon diagrams.

The most important functionality of the MASITS tool is code generation from the design diagrams. JADE agents, behaviours and ontology classes are generated. The ontology is completely generated from the ontology diagram. Agents and their behaviours are generated from agents' internal views. After code generation a programmer has to complete *action()* methods of each behaviour. In addition, the user interface has to be developed independently from the MASITS tool. For details about the MASITS tool and code generation from MASITS diagrams, see (Lavendelis & Grundspenkis, 2009c).

5. CASE STUDY

The MASITS methodology and tool have been used to develop the ITS for the first year graduate course "Fundamentals of artificial intelligence". The course teaches several different algorithms. Thus, an important part of the course is practical usage of learned algorithms. The ITS for such course should concentrate on the practical execution of algorithms.

The developed system supports the following learning process. When a learner registers in the system, a curriculum is generated for him/her. A curriculum consists of modules that, in their turn, consist of themes. In each theme a learner is supplied with a learning material. After finishing studying the material, a learner receives a problem to solve according to the theme. When the learner has finished solving the problem, his/her solution is evaluated by the system and feedback to the learner is given.

Problems given to a learner are of different type: various kinds of tests (single choice tests, multiple choice tests and fill in blanks), search algorithm execution tasks and two person game algorithm execution tasks are used in the current version of the system. The following criteria are used to choose the problem for a learner: (1) adequacy of the level of complexity of the task to the learner's knowledge level, (2) adequacy to learner's preferences like size of the problem, practicality of the problem and (3) frequency of similar tasks given to a learner. Such approach allows to provide unique problems to each learner and to adapt problems to all individual learners.

The holonic multi-agent architecture for ITS development (proposed in (Lavendelis & Grundspenkis, 2008)) is used to implement the system. The higher level holon consists of the following agents: an interface agent, a curriculum agent, a teaching strategy agent, a student modelling agent, a problem generation agent, a knowledge evaluation agent, an expert agent and a feedback generation agent. A problem generation agent, an interface agent, an expert agent and a knowledge evaluation agent are developed as holons that include a

body agent for each type of problems to allow easy modifying a set of used problems. A new type of problem can be added by extending the domain ontology with concepts and predicates that correspond to the added type of the problem and adding agents to the above mentioned second level holons corresponding to the new type of problems. This allows modifying a set of available types of problems without modifying the rest of the system. It can be used either during the further development of the system for current course or by adapting the system to other similar courses.

As mentioned in Section 3.4, agents that can be reused are added to the library of reusable agents. All agents from lower level holons correspond to specific type of problems and, as a consequence, can be reused in other ITSs that include the same type of problems. The only limitation of agents' reuse is the need to use same ontology classes (concepts and predicates) describing the problem and its solutions.

Code of all agents and ontology was generated by the MASITS tool. The ontology was completely generated. Agent classes (more precisely their behaviours) were completed manually. The relative amount of manually written code varied from 20 to 60 percent in different agent classes (60 percent were manually written for problem visualisation agents that have to build appropriate parts of user interface). Additionally, user interface and database connection classes were written manually.

The case study showed that the chosen techniques are suitable and work well for development of agent based ITSs. The knowledge from ITS research integrated in the MASITS methodology helped during the development. The major benefit from the knowledge was that there was no need to define agents, which is one of the most challenging tasks in other methodologies, for example, Prometheus methodology (Winikoff and Padgham, 2004). Instead tasks were allocated to agents from the known set of agents using simple rules. Consistency checking and crosscheck mechanisms appeared to be very useful. The MASITS tool allowed to refine previously created models and all changes were included in all later models. It helped to develop the system iteratively and to return to earlier phases of development if needed. Despite extra effort needed to complete crosschecks, they helped to identify missing elements in created models. A crosschecking can be especially useful if the development is iterative and previously created models are refined.

6. CONCLUSION

ITSs have some specific characteristics that must be taken into consideration during the development process. To do it, the specific AOSE methodology for agent based ITS development is proposed. The methodology is created to be easy usable for development of specific systems. Thus, the generality of methodology is sacrificed by adapting it to the ITS and specific agent development platform.

The main advantages of the proposed methodology are the following. Firstly, knowledge from ITS research is integrated in the methodology to facilitate different steps of development. For example, the agent definition step, that is one of the most challenging steps in general purpose methodologies, is reduced to task allocation to agents from known set using simple rules. Secondly, it supports the whole life cycle, including implementation, testing, deployment and maintenance phases, which are weakly supported in the majority of general purpose methodologies. Thirdly, it provides appropriate techniques for ITS development during the whole life cycle of the methodology. The main limitation of the methodology is its specific purpose. It has to be modified to be applicable to development of any other systems than agent based ITS or use any other implementation platform than JADE.

The MASITS tool supports a development process by providing environment for creation of all diagrams needed in a development process. Such mechanisms as consistency checking, crosschecking, diagram generation and code generation are included in the MASITS tool to facilitate the ITS development process and minimize possibilities of errors in design and implementation.

The future work with MASITS methodology is to develop more case studies for practical evaluation of the methodology in more details. It will give possibility to take into account the users' feedback related to his/her learning efficiency and satisfaction with using the system. Additionally, support to some intelligent mechanisms like planning and learning might be insufficient in current version of the methodology. Such intelligent mechanisms now are developed inside agent behaviours and no specific techniques to model them are provided. Integration of these techniques is another direction of future work. The MASITS might also be used as an example of integration of domain knowledge into the AOSE methodologies. It can be used to introduce specific methodologies for other specific agent-oriented software.

REFERENCES

Capuano, N. et al., 2000. 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)*, Rome, Italy.

Cortese, E. et al., 2005. JADE Test Suite – USER Guide. Available online: http://jade.tilab.com/doc/tutorials/JADE_TestSuite.pdf (Last visited: 26.02.2009).

Crowley, R.S. and Medvedeva, O., 2005. An Intelligent Tutoring System for Visual Classification Problem Solving. *Artificial Intelligence in Medicine*, August, 2005- 2006:36(1), pp. 85-117.

DeLoach, S., 2001. Analysis and Design Using MaSE and agentTool. *Proceedings of the 12th Midwest Artificial Intelligence and Cognitive Science Conference*, Oxford OH, March 31 - April 1 2001. pp. 1-7.

Devedzic, V. et al., 2000. Teaching Formal Languages by an Intelligent Tutoring System. *Educational Technology & Society*, Vol. 3, No. 2, pp. 36-49.

Gascuena, J.M. and Fernández-Caballero, A., 2005. An Agent-based Intelligent Tutoring System for Enhancing E-learning/E-teaching. *Int. Journal of Instructional Technology and Distance Learning*, Vol. 2, No.11, pp. 11-24.

Giunchiglia, F. et al., 2001. The Tropos Software Development Methodology: Processes, Models and Diagrams. Technical Report No. 0111-20, ITC - IRST.

Grundspenkis, J. and Anohina, A., 2005. 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.

Hospers, M. et al., 2003. An Agent-based Intelligent Tutoring System for Nurse Education. *Applications of Intelligent Agents in Health Care (eds. J. Nealon, A. Moreno)*. Birkhauser Publishing Ltd, Basel, Switzerland, pp. 141-157.

Iglesias, C. et al., 1998. Analysis and design of multiagent systems using MAS-CommonKADS. In Intelligent Agents IV (ATAL97), LNAI 1365, pp. 313-326. Springer-Verlag..

Lavendelis, E., and Grundspenkis, J., 2008. Open Holonic Multi-Agent Architecture for Intelligent Tutoring System Development. *Proceedings of IADIS International Conference „Intelligent Systems and Agents 2008"*, Amsterdam, The Netherlands, 22 - 24 July 2008, pp. 100-108.

Lavendelis, E. and Grundspenkis, J., 2009a. Requirements Analysis of Multi-Agent Based Intelligent Tutoring Systems. *Scientific Proceedings of Riga Technical University „Computer Science. Applied Computer Systems"*, RTU Publishing, Riga. (Accepted for publishing)

Lavendelis, E. and Grundspenkis, J., 2009b. Design of Multi-Agent Based Intelligent Tutoring Systems. *Scientific Proceedings of Riga Technical University „Computer Science. Applied Computer Systems"*, RTU Publishing, Riga. (Accepted for publishing)

Lavendelis, E. and Grundspenkis, J., 2009c. MASITS - A Tool for Multi-Agent Based Intelligent Tutoring System Development. *Proceedings of The 7th International Conference on Practical Applications of Agents and Multi-Agent Systems*, Salamanca, Spain, 25-27 March 2009 p. 490-500.

Massonet, P. et al., 2002. From AOSE methodology to agent implementation. *Proceedings of the First International Joint Conference on Autonomous Agents and Multiagent Systems*, Bologna, Italy. pp. 27 – 34.

Matsuda, N. and VanLehn, K. 2005. Advanced Geometry Tutor: an Intelligent Tutor that Teaches Proof-Writing with Construction. *Proceedings of the 12th International Conference on Artificial Intelligence in Education*, 18-22 July, 2005, p.443-450.

Picard, G. and Gleizes, M.-P., 2004. The ADELFE Methodology. *Methodologies and Software Engineering for Agent Systems. The Agent-Oriented Software Engineering Handbook (Eds. Bergenti F., Gleizes M.P., Zambonelli F.)*, pp. 157-174.

Taveter, K. and Wagner, G., 2005. Towards Radical Agent-Oriented Software Engineering Process. *Agent-Oriented Methodologies, (Eds. Henderson-Sellers B., Giorgini P.)* Idea Group Publishing, London, pp. 277-316.

Virvou M. and Tsiriga V. 2001. Web Passive Voice Tutor: an Intelligent Computer Assisted Language Learning System over the WWW. *Proceedings of the IEEE International Conference on Advanced Learning Technology: Issues, Achievements and Challenges*, Madison, WI, USA, 6-8 August, 2001. - pp.131-134.

Winikoff, M. and Padgham, L., 2004. The Prometheus Methodology. *Methodologies and Software Engineering for Agent Systems. The Agent-Oriented Software Engineering Handbook (Eds. Bergenti F., Gleizes M.P., Zambonelli F.)*, pp. 217-236.

Zambonelli, F., et al., 2005. Multi-Agent Systems as Computational Organisations: The Gaia Methodology. *Agent-Oriented Methodologies, (Eds. Henderson-Sellers B., Giorgini P.)* Idea Group Publishing, London, pp. 136-171.