

See discussions, stats, and author profiles for this publication at: <https://www.researchgate.net/publication/220625854>

Design of Multi-Agent Based Intelligent Tutoring Systems

Article in *Scientific Journal of Riga Technical University Computer Sciences* · January 2009

DOI: 10.2478/v10143-009-0004-z · Source: DBLP

CITATIONS

5

READS

89

2 authors:



[Egons Lavendelis](#)

Riga Technical University

34 PUBLICATIONS 148 CITATIONS

[SEE PROFILE](#)



[Janis Grundspenkis](#)

Riga Technical University

136 PUBLICATIONS 867 CITATIONS

[SEE PROFILE](#)

Some of the authors of this publication are also working on these related projects:



Model for Identification of Politically Exposed Persons [View project](#)

DESIGN OF MULTI-AGENT BASED INTELLIGENT TUTORING SYSTEMS

DAUDZAGENTU SISTĒMĀS SAKŅOTU INTELEKTUĀLU MĀCĪBU SISTĒMU PROJEKTĒŠANA

E.Lavendelis, J.Grundspenkis

Multi-Agent Systems, Intelligent Tutoring Systems, Agent Oriented Software Engineering

1. Introduction

Extensive research in the agent oriented software engineering (AOSE) is ongoing. Several agent oriented software engineering methods and methodologies have been proposed, for example, Gaia [1], Prometheus [2], PASSI [3], Agent UML [4], MAS-CommonKADS [5], etc. Majority of methodologies include an approach for the design phase.

At the same time Intelligent Tutoring Systems (ITS) are studied at their own. Traditionally ITSs consist of three modules, namely a tutoring module, a student module and an expert module [6]. During the last decades agents and multi-agent systems are widely used for ITSs development. Multi-agent based ITSs consist of specific agents that are intended to implement specific and complex process of tutoring. Sets of agents are developed to implement modules of ITSs [7]. Our previous work [8] has proved that a specific multi-agent architecture for ITSs development is recommendable. The proposed architecture facilitates both the development and maintenance of the system. All research work in the area of ITSs indicates that specific knowledge should be taken into consideration during the design of ITSs. Still, there are no specific approaches for multi-agent based ITS design. Moreover, none of the existing general-purpose methodologies provides easy usage of existing set of agents or agent architecture during the design phase. Thus, usage of ITS research results in the design of ITS is difficult unless specific design approach is developed.

The most part of the design approaches (including the majority of abovementioned methodologies) are independent from the concrete platform, i.e. these methodologies use either their own design concepts or common-known agent design concepts. This promotes a wider usage of design approaches. However, it is hard to convert the design concepts to concepts used in different agent platforms. As Massonet and colleagues have shown in [9], every pair methodology-agent platform has almost unique transformation from design to implementation concepts. Methodologies do not include these transformations. Therefore the designer of the system has to define a transformation for the chosen pair of the design approach and the agent implementation platform.

This paper describes a specific design approach for multi-agent based ITSs. The approach includes the main ideas from both fields: AOSE and ITS. Additionally, it differs from the majority of methodologies by using concepts from an implementation platform in the design phase. Such a choice enables a direct code generation from diagrams created during the design. The remainder of the paper is organised as follows. The second section describes the proposed approach in general. Sections three and four describe two main stages of the design. Section five includes conclusions and future work.

2. Proposed approach of ITS design

Design based on the proposed approach consists of two stages, namely, external and internal design of agents. During the external design stage interactions among agents and functionality of them are specified. At the internal design of agents, the internal structure of them is designed. Thus, a top down approach is used.

During the description of the design we assume that requirements analysis of the system is already finished. Moreover, during the requirements analysis two artefacts have been created and included in the requirements specification. These artefacts are the following [10]:

- The goal model of the system that consists of the goal hierarchy and goal descriptions. Goals are used to check if all system's goals are accomplished by created tasks.
- The use case model that contains a use case diagram, descriptions and scenarios. Use case scenarios are used for task definition. Tasks are defined according to steps included in scenarios. Use cases also define interface agent tasks related to interactions with learners. Use cases are used to create use case maps and to define interaction among agents.

The proposed approach includes main results from the ITS research into the design phase. We use the holonic multi-agent architecture for ITS development [8]. The architecture is a hierarchy of holons. A holon is an agent that consists of smaller agents and appears to its environment as a single agent. Each holon is represented by its head, for details see [11]. The higher level of the hierarchy is the main holon, consisting of the traditional set of agents used to implement ITS. These agents are an interface agent, a curriculum agent, a tutoring strategy agent, a problem generation agent, a feedback generation agent, a student modelling agent, a knowledge evaluation agent and an expert agent. This set can be customized to adapt the needs of the specific system. Each of higher level agents can be implemented as a holon. So, we propose to design hierarchical holonic multi-agent systems for ITS implementation. We have chosen to use agent implementation platform's concepts during the design process. This allows a direct code generation from diagrams created during the design and transformation from design concepts to implementation ones is not needed. We have chosen the JADE platform [12] as the implementation platform, because it provides simple way to create Java agents and organise their interactions. 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. These elements must be specified during the design.

3. External design of agents

During the external design of agents the following steps are done:

- Creation of a task diagram (decomposition) according to use cases;
- Crosscheck of task diagram according to a goal diagram;
- Task allocation to higher level agents;
- Agent interaction design. Interaction design of each agent pair consists of four substeps:
 - If interaction is too complicated to directly create interaction diagram, use case maps are created.
 - Interaction among agents is designed and specified in an interaction diagram.
 - In case interaction between two agents consists of too many messages and diagram becomes unreadable, the interaction among these agents is created in new diagram called interaction detalisation.
 - Initial ontology creation (is done simultaneously with previous two steps).

The algorithm for this stage is shown in Figure 1. The following subsections describe steps in detail.

3.1. Task decomposition and crosscheck

The first step of design is creation of task decomposition. As a result a task diagram is created. A task diagram consists of one or more task hierarchies (trees). Task diagram is created by defining tasks corresponding to steps defined in use case scenarios, i.e. a task is created for a single step or a few consecutive steps of the use case scenario. Task hierarchies are created by linking up tasks that can be assigned to the same agents not by use cases or goals. So, a structure of the task hierarchy is different from the goal hierarchy. The task diagram has the following notation: tasks are denoted with rectangles, decomposition links are denoted with lines (see Table 1).

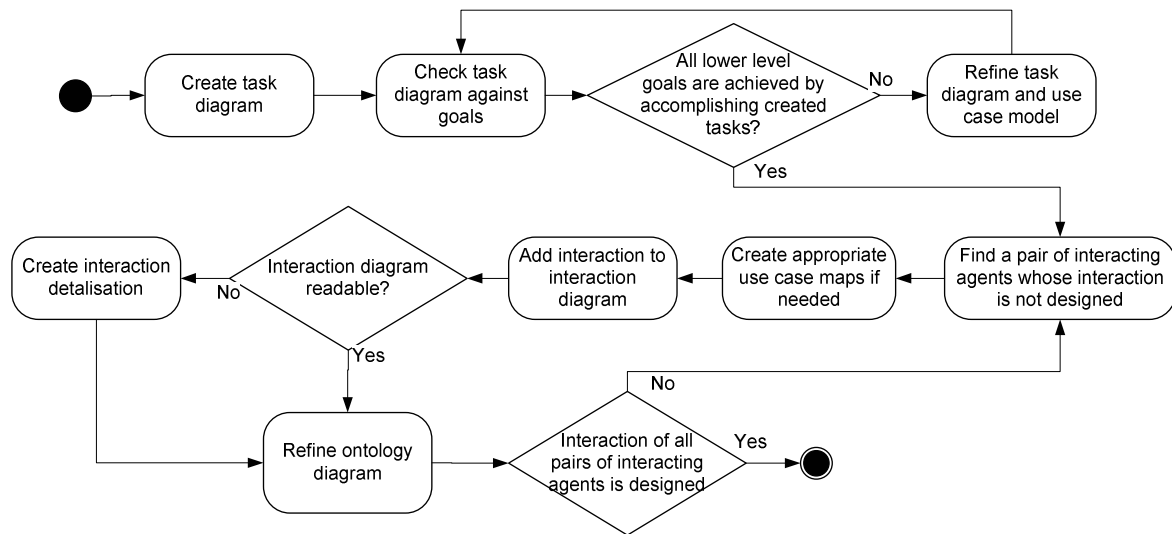


Fig.1. Algorithm used for external design

After a task definition a crosscheck against goal diagram is performed. It is checked, if all lower level goals from goal hierarchy are achievable by defined tasks. If some goals are not achievable, task hierarchy and corresponding use case scenario(-s) are looked through. If scenarios include achieving a goal then only additional tasks are added. If scenarios do not include needed steps to achieve a goal, use case scenarios are redefined before defining additional tasks.

Table 1

Notation of the Goal Diagram

No.	Element of diagram	Notation
1.	Task	<div style="border: 1px solid black; padding: 2px; display: inline-block;">Task</div>
2.	Task decomposition link	—

3.2. Task allocation to higher level agents

The second step of external design is task allocation to agents. During this step a task-agent diagram is created. The diagram shows, which tasks are allocated to which agents. Task allocation is based on ITS research, where a set of agents for ITS development are defined [7]. Each agent of the set is part of one of the traditional ITS modules and is responsible for definite tasks. In our approach we use the abovementioned holonic architecture and a set of higher level agents defined in [8]. Each of these agents is responsible for a set of certain tasks. Thus, tasks defined in the previous step have to be allocated to agents according to the ITS research, which can be summarised in the following way:

1. *Interface agent* or *animated pedagogical agent* are allocated to the following kinds of tasks:
 - 1.1. Tasks related to starting a learning session (e.g. user identification, user information retrieval and sending to other agents).
 - 1.2. Tasks related to monitoring of user actions.
 - 1.3. Tasks including interaction with the user. This is the only agent interacting with the user.
2. All subject matter related problem solving tasks are allocated to the *expert agent*.
3. The *student modelling agent* is allocated to the following kinds of tasks:
 - 3.1. Learner's level of knowledge and skills model building tasks.
 - 3.2. Learner's psychological characteristics modelling tasks.
 - 3.3. Learner's interaction with the system registering tasks.

- 3.4. Determining tasks of learner's mistakes and their causes.
- 3.5. Full student model creation task.
4. The *knowledge evaluation agent* is allocated to tasks related to learner's solution evaluation (by comparing it to the expert's solution).
5. Generation, evaluation, updating of the curriculum is allocated to the *curriculum agent*.
6. Learning material choosing or generation tasks are allocated to the *teaching strategy agent*.
7. Test, task and problem generation tasks are allocated to the *problem generation agent*.
8. Feedback, explanations and hints generation tasks are allocated to the *feedback generation agent*.
9. ITSs can have additional functionality, which is not included in the above mentioned traditional agents. In that case designers use generic principles to create additional agents and allocate remaining tasks to them. Such additional agents are needed in case if the system includes some domain specific functionality, for example, patient modelling and sterility agents of nurse education system Ines [13] are problem specific agents in medicine domain.

A task-agent diagram is a task diagram with added agents to tasks. Thus, notation of the task-agent diagram is the same as notation of task diagram, only agents are added to tasks. It can be done in two ways. If diagrams are drawn "on the paper" (or without any specific tools) the easiest way to specify agent is to draw a rectangle around tasks that are allocated to it. If a diagram drawing tool is used, then each task can have an agent name added to it. An example of task-agent diagram is given in Figure 2. The example contains three task trees consisting of one, two and three levels.

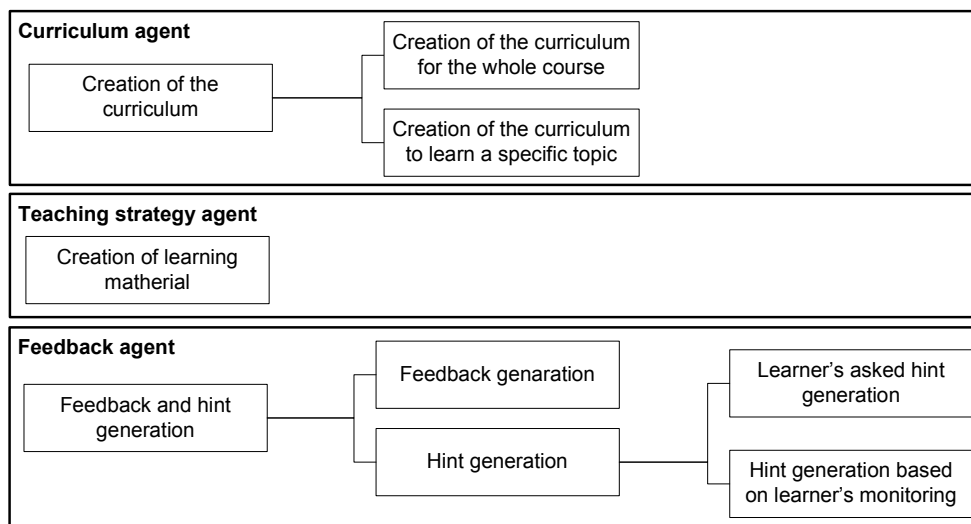


Fig.2. Example of task-agent diagram

3.3. Use case map creation






Use case map creation is the first step of agent interaction design. An interaction is designed for each pair of interacting agents, thus, the following two steps are done for each pair of interacting agents. Additionally, an interaction between the user and the interface agent (or animated pedagogical agent) is designed in the same way as an interaction between two agents.

In object-oriented approach use case maps are used to model the control passing path during the execution of use case, for details see [14]. In multi-agent design use case maps include agents, their tasks and message path among them. The message path corresponds to the use case scenario. The use case map represents the use case scenario in a form where interaction among agents is shown explicitly. Each link between two tasks can be considered as a message between respective agents. Thus, after creating the use case map an interaction among agents can be easily specified just by adding a message content.

In the use case map agents are denoted by rectangle, starting point of the path is denoted by full circle, branching is denoted by a diamond and task done inside the path is denoted by task's name or number. Path ends with bold line. Full notation of use case map is shown in Table 2. Example of use case map is shown in Figure 3. For details of use case map usage for agent design see [14].

Table 2

Notation of the Use Case Map

No.	Element of diagram	Notation
1.	Agent	
2.	Path	
3.	Starting point	
4.	Branching point	
5.	End point	
6.	Task	Task or T1

3.4. Interaction design


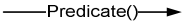


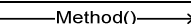
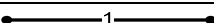
During this step an interaction in the form of transferred messages is specified for each pair of agents. Each message is specified as a link between two agents and its content (predicate sent). Predicates as message content are chosen, because in the selected agent development platform predicates are used in agent communication. This choice enables automatic transformation from the interaction design to message sending implementation code.

As a result of this stage, the interaction diagram is created. 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 sent predicates. The diagram includes the interaction between the interface agent and the user, too. Thus, a few additional elements are added: the user (denoted as actor), events perceived by agent (dashed line) and methods of interface called by interface agent (line with simple arrow).

In case an interaction between two agents consists of large number of messages, the interaction diagram becomes unreadable. This is solved by detalisation mechanism, when an interaction between a pair of agents is substituted with one link called detalisation link and an interaction among two agents is specified in the special interaction diagram consisting of two agents (one of them can be the user) and messages sent between them. A full notation of the interaction diagram is shown in Table 3.

Table 3

Notation of the Interaction Diagram

No.	Element of diagram	Notation
1.	Agent	
2.	Message	
3.	User	 User role
4.	Event	
5.	Call of interface method	
6.	Interaction detalisation	

An interaction among agents of each holon is modelled in one interaction diagram. One diagram called higher level interaction diagram is created for higher level agents (higher level holon) and one diagram is created for each lower level holon. A fragment of the interaction diagram is shown in Figure 4.

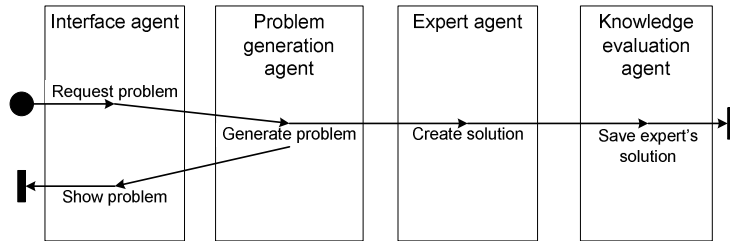


Fig.3. Example of use case map

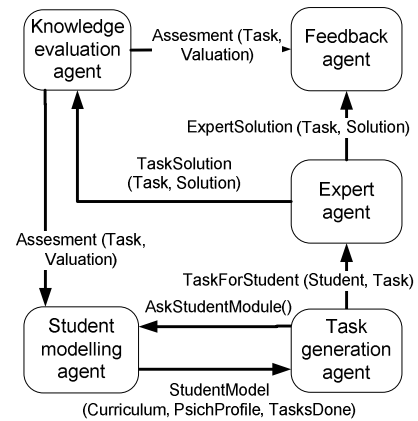


Fig.4. Fragment of the interaction diagram

3.5. Initial ontology design

During this phase the ontology diagram is created. The 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:

- Concept – subclasses of this class are domain concepts.
- Predicate – subclasses of this class are predicates used in agent interactions as message contents.

All other classes are subclasses of the abovementioned two superclasses.

Table 4

Notation of the Ontology Diagram

No.	Element of diagram	Notation
1.	Concept	<div style="border: 1px solid black; border-radius: 10px; padding: 5px; width: fit-content;"> Name <hr style="border: 0; border-top: 1px solid black;"/> Attributes </div>
2.	Predicate	<div style="border: 1px solid black; border-radius: 10px; padding: 5px; width: fit-content;"> Name <hr style="border: 0; border-top: 1px solid black;"/> Attributes </div>
3.	Inheritance	<div style="border: 1px solid black; border-radius: 10px; padding: 5px; width: fit-content;"> <hr style="border: 0; border-top: 1px solid black;"/> </div>

The initial ontology diagram is created simultaneously with the interaction design. During the interaction design predicates included in messages are added to the ontology diagram. Concepts needed to create predicates are added to the ontology diagram, too. Each concept and predicate is added to the ontology diagram as subclass of one of the existing classes. Hierarchy is denoted by inheritance links. Each class has attributes which have a name, a cardinality and a type. Attribute's type may be a primitive type, Java class or a concept defined in the ontology diagram. Predicates are not allowed to be used as attribute types. Notation of the ontology diagram is shown in Table 4. Initially created ontology diagram is refined during the agents' internal design by adding predicates and concepts used inside agents. A simple example of an ontology diagram is shown in Figure 5.

4. Internal design

After finishing the external design, a set of agents and tasks they have to accomplish are defined. The aim of internal design is to specify the way, how these tasks will be accomplished. During this step

agent's perception (message and events) processing, actions and beliefs are designed. Agent's actions are modelled using the agent diagram. Agent beliefs are described as attribute definitions. Such approach allows automatic generation of agents' source code using the created diagrams.

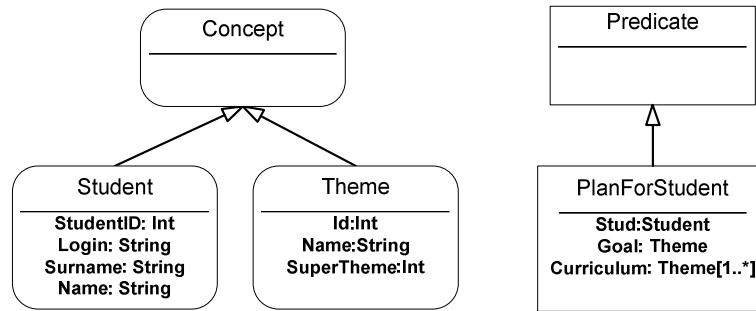


Fig.5. Example of ontology diagram

Agents can also be realised as multi-agent systems or holons. In that case firstly the head of the holon is designed as a higher level agent and secondly the body of the holon is designed as a multi-agent system. The interaction diagram is used to create the external design of holon's agents. A vertex of each holon is also added to the holon hierarchy. Agent diagrams are created for holon's body agents to create their internal design.

To create the internal design of a particular agent, the following algorithm is used (see Figure 6):

1. Create agent's internal view consisting of agent's diagram, agent's beliefs, message and event processing rules, start-up rules and agent's actions.
2. Refine the ontology diagram by adding concepts used in agent's beliefs.
3. If agent is realised as a holon, the following steps are done:
 - 3.1. The holon hierarchy diagram is created or refined;
 - 3.2. Holon is designed:
 - 3.2.1. Interaction diagram for the holon is created.
 - 3.2.2. The ontology diagram is refined by adding concepts used in holon's interaction diagram.
 - 3.2.3. Agent's internal design for each agent is carried out, by doing all steps of this algorithm.

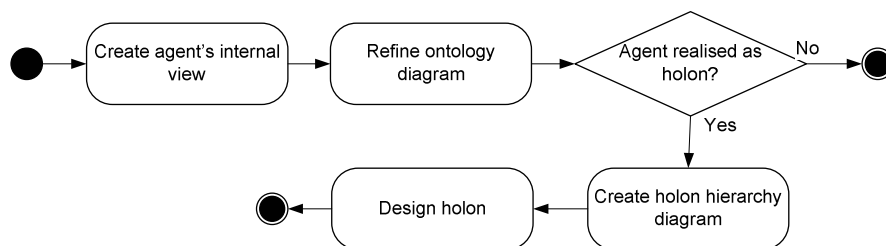


Fig.6. Algorithm used for internal design

4.1. Internal view of an agent

The main element of agent's internal design is agent's internal view, consisting of the following elements:

- The agent diagram, including agent's perceptions, messages sent by the agent, agent's actions and links among these elements.
- Perception (message and event) processing rules.

- Startup rules describing, what actions are done by an agent during the startup.
- Agent's beliefs.
- Agent's action list specifying implementation details of agent's actions.

Elements of agent's internal view can be created in any order; however agent's actions, beliefs and perceptions are used in rules. Thus, 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. Recommended order of design in each iteration is the following:

1. Define agents beliefs;
2. Add agent's action(s) to agent diagram, create appropriate relationships and specify actions properties in action list.
3. Create the rule(s) that process received perceptions and initiate an action or change agents beliefs.

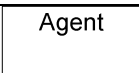







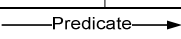

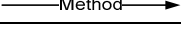
Agent diagram includes:

1. All messages sent and received by the 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 the user interface.

In the agent diagram the internal structure of an agent is represented with agent's actions and interaction among them. Actions are denoted with small rounded rectangles. Actions mainly are initiated reacting on received messages or perceptions. Additionally actions initiated by time can be used (denoted by a clock). 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 like in structural modelling [15]. An agent can have four types of contacts: message receiving contact, message sending contact, event receiving contact and action to external environment contact. Each message, event or action is added to its own contact. A link from a contact to an action means that corresponding production rule exists. The rule initiates the action if percept according to the contact is received. A link from an action to a contact means that a message is sent (or action to environment is performed) as a result of the action. Table 5 contains full notation of agent diagram. An example of agent's internal view is shown in Figure 7.

Table 5

Notation of the Agent Diagram

No.	Element of diagram	Notation
1.	Agent	
2.	Action	
3.	Action initiated by time	
4.	Link among actions	
5.	Message receiving contact	
6.	Message sending contact	
7.	Event perceiving contact	
8.	Action to environment contact	
9.	Message	
10.	Event	
11.	Action to environment (method of interface called)	

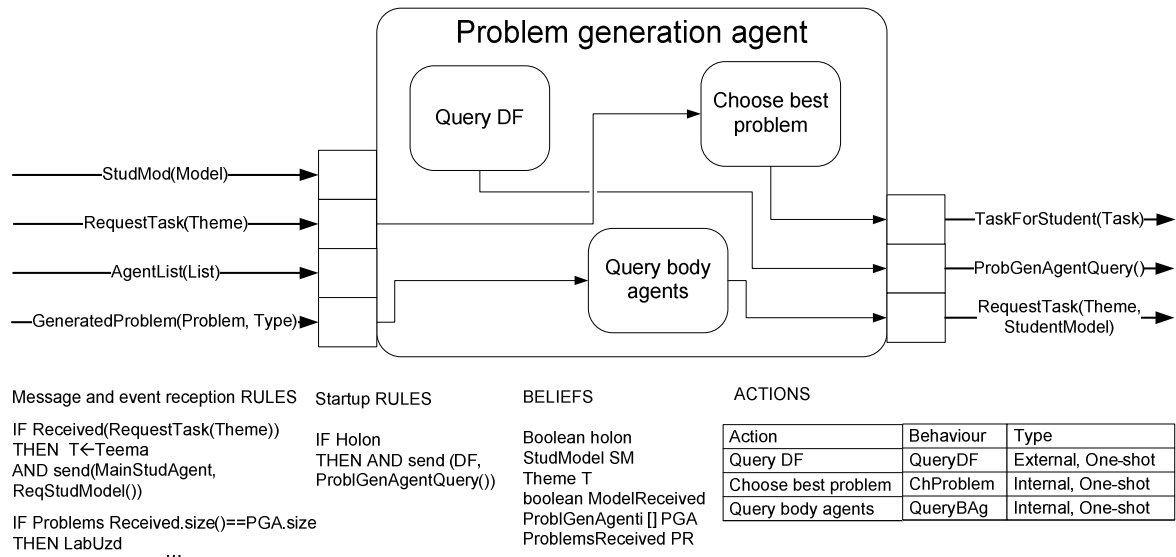


Fig.7. Example of agent internal view

Agents' beliefs are specified as pairs: <Type>, <Name>. Type can be either primitive type used in Java language, Java class or ontology class (predicate or concept) from the ontology diagram.

Actions are described in a table containing the following columns: name of action, name of corresponding behaviour class, type of action (one-shot, cyclic, timed, etc.), inner (implemented as inner class in agents class) or outer (implemented as a class in the same package as agent) behaviour class, type of action. Perception processing and start-up rules are designed as IF_THEN rules. Rules are created from the following templates:

1. IF_THEN rule: IF <Condition> THEN <Action> ELSE <Action>. The following templates can be used as condition: Received, Compare, Condition conjunction (AND), disjunction (OR), negation (NOT), True (the condition that is always true). Templates Action, Belief Set, Action conjunction (AND) and IF_THEN rule can be used as actions.
2. Received (Predicate) is a check if received message contains specified predicate.
3. Condition conjunction (AND) is true if and only if all arguments are true. Arguments can be the same type as the IF part of the IF_THEN rule.
4. Disjunction (OR) is true if at least one of the arguments is true. Arguments can be the same type as the IF part of the IF_THEN rule.
5. Negation (NOT) is true if and only if the argument is false. An argument can be the same type as the IF part of the IF_THEN rule.
6. Action. One of agent's actions which is initiated when the condition holds.
7. Action conjunction (AND). Conjunction in THEN or ELSE parts denotes sequential action execution from the left to the right.
8. Belief set. Sets value of one of the agent's beliefs.

During the creation of agent's internal view previously created initial version of the ontology diagram is refined by adding concepts and predicates used in internal view of the agent. All concepts and predicates used to specify agents' beliefs are added to the ontology diagram.

4.2. Holon design

If the higher level agent is realised as a holon, then the internal view of an agent is created for the head of the holon. Additionally design of holon's multi-agent system is done, using previously described agent design techniques. The main differences from higher level agents' design are the following:

- Holon hierarchy design is done, resulting in the holon hierarchy diagram. The diagram shows holons and hierarchical relationships among them. The diagram is created starting from higher

level holon and adding subholons to each holon. Notation of holon diagram is shown in Table 6. Example of the diagram including eight second level holons according to the multi-agent architecture for ITS development is shown in Figure 8.

- During the external design of holon's agents only the interaction design is done.
- The interaction diagram is extended, allowing designing open holons. The holons' interaction diagrams may contain sets of typical agents. A typical agent means that an open holon contains a set of typical agents, but number of agents and concrete instances are unknown. So, open holons can be designed just by specifying types of agents that can be added to the holon. The directory facilitator agent is also included to allow finding real instances of the typical agents. Extension of the interaction diagram's notation is shown in Table 7.

Table 6
Notation of the Holon Hierarchy Diagram



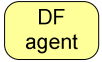
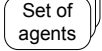
No.	Element of diagram	Notation
1.	Holon	
2.	Hierarchical link	

Table 7
Notation Extension of Interaction Diagram

No.	Element of diagram	Notation
1.	Yellow pages agent	
2.	Set of typical agents	

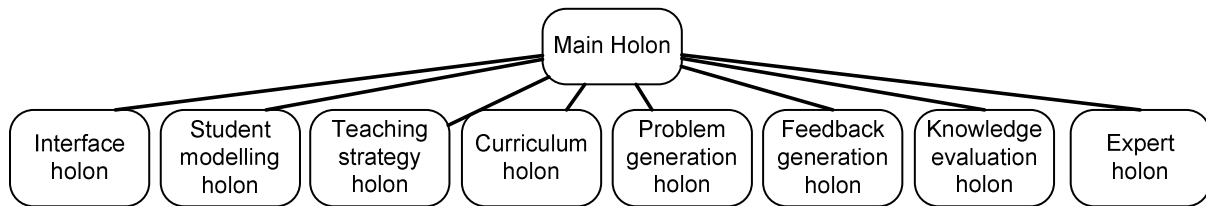


Fig.8. Example of holon diagram

After the holon's interaction design agents' internal views are created in the same way as internal views of the higher level agents. At last, previously created diagrams are refined in the following way:

- The ontology diagram is refined by adding concepts and predicates used in agents' communication and beliefs.
- Incoming and outgoing messages of the head of the holon are refined by adding messages sent to and received from body agents of the holon.

Each agent at any level can be realised as a holon itself. In that case holon design is repeated for the agent realised as a holon. So, the design approach fully supports the hierarchical multi-agent architecture for ITS development described in [8].

5. Conclusions and future work

The specific design approach for multi-agent based ITSs has been proposed. The proposed approach is a graphical one – it contains a set of diagrams to be created during the design. Knowledge from ITS research is included, too. Although the approach is developed for ITS development, diagrams can be used also to develop different purpose multi-agent systems. In that case the domain knowledge of agents and their architectures have to be added to the approach. The proposed approach clearly separates two stages of design, allowing designing multi-agent system first and only after that designing individual agents.

The design of the multi-agent system is carried out in concepts that are used in the well known agent development platform JADE. So, transformation between concepts used in design and implementation phases is not necessary. Moreover, it is possible directly generate partial Java code of agents.

Despite the case study enabling the evaluation of the proposed approach in more details is still under development, we think that the abovementioned advantages of the approach make it a promising one. Our future work is to develop algorithms for code generation from diagrams developed in the design phase. A design tool using the proposed design approach and code generation algorithms is planned, too. Approaches for later phases of the software development life-cycle are being worked out and will be integrated to create a full life-cycle methodology for multi-agent based ITS.

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. – pp. 285-313.
2. Padgham L., Winikoff M. Prometheus: A Methodology for Developing Intelligent Agents // *Agent-Oriented Software Engineering III, LNCS*, – Vol. 2585, Springer, 2003. – pp. 174-185.
3. Burrafato P., Cossentino M. Designing a multi-agent solution for a bookstore with the PASSI methodology // *Electronic Proceedings of Fourth International Bi-Conference Workshop on Agent-Oriented Information Systems (AOIS-2002) at CAiSE'02*. – Toronto, Canada, 2002.
4. Odell J., Parunak H.V.D., Bauer B. Extending UML for Agents // *Proceedings of the Agent-Oriented Information Systems Workshop at the 17th National conference on Artificial Intelligence*, 2000. – pp. 3-17.
5. Iglesias C., Garijo M., Gonzales J.C., and Velasco J.R. Analysis and design of multiagent systems using MAS-CommonKADS // *Intelligent Agents IV (ATAL97)*. – LNAI 1365, Springer-Verlag, 1998. – pp. 313-326.
6. Smith A.S.G. Intelligent Tutoring Systems: personal notes. – School of Computing Science at Middlesex University. 1998. <http://www.cs.mdx.ac.uk/staffpages/serengul/table.of.contents.htm> (Last visited 18.04.2005)
7. Grundspenkis J., Anohina A. Agents in Intelligent Tutoring Systems: State of the Art // *Scientific Proceedings of Riga Technical University, 5th series „Computer Science. Applied Computer Systems”* – Vol.22, (2005) – pp. 110-121.
8. Lavendelis E., Grundspenkis J. 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.
9. 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, 2002. – pp. 27 – 34.
10. Lavendelis E., Grundspenkis J. Requirements Analysis 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).
11. Gerber C., Siekmann J., Vierke G. Holonic multi-agent systems, Technical Report R-99-03, DFKI GmbH. 1999.
12. JADE Home Page. <http://jade.tilab.com/> (Last visited: 10.03.09).
13. Hospers M., Kroezen L., Nijholt A., Akker R., Heylen D. Developing a Generic Agent-based Intelligent Tutoring System and Applying it to Nurse Education // *Proceedings IEEE International Conference on Advanced Language Technologies (ICALT'03)*. – Athens, Greece, 9-11 July, 2003. – pp. 443-448.
14. Buhr R.J.A., Elammari M., Gray T., Mankovski S. Applying Use Case Maps to Multi-Agent Systems: A Feature Interaction Example // *HICSS(6)*. 1998. – pp. 171-179.
15. Grundspenkis J. Structural Modelling of Complex Technical Systems in Conditions of Incomplete Information: A Review // *Modern Aspects of Management Science*. – No. 1, Riga, Latvia, 1997. – pp. 111-135.

Egons Lavendelis, M.sc.eng., researcher, Riga Technical University, Meza 1/4, Riga, LV 1048, Latvia, phone: +371 29876891, egons.lavendelis@rtu.lv

Janis Grundspenkis, Dr.habil.sc.ing., professor, Riga Technical University, Meza 1/4, Riga, LV 1048, Latvia, phone: +371 67089581, janis.grundspenkis@cs.rtu.lv

Lavendelis E., Grundspenķis J. Daudzaģentu sistēmās sakņotu intelektuālu mācību sistēmu projektēšana

Projektējot daudzģentu sistēmās sakņotas intelektuālas mācību sistēmas ir nepieciešams ņemt vērā pētījumu rezultātus no divām sfērām – aģentorientētas programmatūras inženierijas un intelektuālu mācību sistēmu izpētes. Tādēļ ir nepieciešamas projektēšanas pieejas, kas iekļauj galvenās atziņas no abām šīm sfērām. Taču šādas pieejas neeksistē. Šajā rakstā ir piedāvāta daudzģentu sistēmās sakņotu intelektuālu mācību sistēmu projektēšanas pieeja, kas iekļauj galvenās idejas no abām minētajām sfērām. Piedāvātā projektēšanas pieeja paredz projektēšanu veikt divās stadijās: ārējās projektēšanas stadijā un aģentu iekšējās struktūras projektēšanas stadijā. Ārējās projektēšanas stadijā tiek projektēta aģentu uzvedība uz āru un mijiedarbība starp aģentiem. Šajā stadijā tiek veikti sekojoši soļi: uzdevumu modelēšana un piešķiršana aģentiem, lietošanas gadījumu karšu izveide sarežģītākajiem lietošanas gadījumiem, aģentu mijiedarbības projektēšana un ontoloģiju izveide. Šīs fāzes laikā aģenti tiek definēti atbilstoši holoniskai aģentu arhitektūrai intelektuālu mācību sistēmu realizācijai. Iekšējās struktūras projektēšanas stadijā tiek projektēts aģenta iekšējā struktūra, kas tiek attēlota speciālā diagrammā – aģenta iekšējā skatā. Aģenta iekšējais skats sastāv no darbībām, saitēm starp tām, ienākošo ziņojumu un uztveru apstrādes likumiem un aģenta pārliecībām. Aplūkoto projektēšanas pieeju ir plānots iekļaut pilna dzīves cikla metodoloģijā daudzģentu sistēmas sakņotu intelektuālu mācību sistēmu izstrādei. Projektēšana tiek veikta izmantojot JADE aģentu platformas konceptus un tādēļ ir iespējams veikt JADE aģentu Java koda ģenerēšanu no izveidotajām diagrammām.

Lavendelis E., Grundspenķis J. Design of Multi-Agent Based Intelligent Tutoring Systems

Research of two fields, namely agent oriented software engineering and intelligent tutoring systems, have to be taken into consideration, during the design of multi-agent based intelligent tutoring systems (ITS). Thus there is a need for specific approaches for agent based ITS design, which take into consideration main ideas from both fields. In this paper we propose a top down design approach for multi-agent based ITSs. The proposed design approach consists of the two main stages: external design and internal design of agents. During the external design phase the behaviour of agents and interactions among them are designed. The following steps are done: task modelling and task allocation to agents, use case map creation, agent interaction design, ontology creation and holon design. During the external design phase agents and holons are defined according to the holonic multi-agent architecture for ITS development. During the internal design stage the internal structure of agents is specified. The internal structure of each agent is represented in the specific diagram, called internal view of the agent, consisting of agent's actions and interactions among them, rules for incoming message and perception processing, incoming and outgoing messages, and beliefs of the agent. The proposed approach is intended to be a part of the full life cycle methodology for multi-agent based ITS development. The approach is developed using the same concepts as JADE agent platform and is suitable for agent code generation from the design diagrams.

Лавенделис Е., Грудспенкис Я. Проектирование интеллектуальных систем обучения основанных на многоагентных системах

Проектируя обучающие системы основанные на интеллектуальных агентах, необходимо брать во внимание два направления: методы разработки и проектирования интеллектуальных агентов и исследования в области обучающих систем. На данный момент не существует подходов объединяющих оба метода. В данной научной статье рассмотрен подход проектирования интеллектуальных систем обучения, включающий основные идеи из обеих выше упомянутых подходов. Предложенный метод проектирования выполняется в две этапа: этап внешнего проектирования и этап проектирования внутренней структуры агентов. На этапе внешнего проектирования создается модель внешнего поведения агента и модель взаимодействия агентов. Данный этап состоит из следующих шагов: моделирование заданий и присвоение заданий агентам, построение карт вариантов использования для наиболее сложных вариантов использования системы, проектирование взаимодействия агентов и создание онтологии. На этапе проектирования внутренней структуры агентов создается внутренний вид агентов, который представляется в виде диаграммы отражающей действия, взаимосвязи между действиями, входящие сообщения, законы обработки восприятий и убеждения агента. Рассмотренный в статье подход планируется включить в методологию разработки обучающих информационных систем основанных на агентах, которая обобщит весь жизненный цикл проекта. Проектирование выполняется с использованием концептов платформы JADE, вследствие чего существует возможность на основе диаграмм генерировать код программ.