12

Distributed Intelligent Tutoring System Architectures

Egons Lavendelis
Riga Technical University
Latvia

1. Introduction

Traditional architecture of Intelligent Tutoring Systems (ITSs) does not offer sufficient modularity. There is a lack of open distributed ITS architectures, despite ITSs being systems that may need frequent changes due to the modifications of particular course or adaptation to new courses. The chapter focuses on usage of distributed technologies in development of open ITSs to increase the modularity of the ITSs and facilitate the implementation of needed changes into ITSs. The aim of the chapter is to propose open and highly modular ITS architectures, using two distributed paradigms – intelligent software agents and services.

To realize intelligent tutoring various types of learning materials and problems have to be presented to the learner, moreover it should be done intelligently enough to successfully simulate the human tutor. Thus all known ITSs concentrate on a certain problem domain or course to provide specific functionality for problems and examples of the domain. For example, the system has to be capable to analyse learner's actions during the problem solving. Each new type of problems needs corresponding code to handle it. Problems differ from course to course and may change if the course is changed. As a consequence the functionality of ITS may be modified to adapt to the changes in the course or to a new course. The system should be open for certain types of components handling new types of problems, materials, feedback, etc. The architecture of ITS should support such an openness.

ITSs traditionally have modular architecture consisting of four modules, namely tutoring module, expert module, student diagnosis module and communication module (Grundspenkis & Anohina, 2005). The main principle is to build components using only one type of knowledge (pedagogical knowledge, domain knowledge and knowledge about the learner). As a result, the architecture does not have sufficient modularity for complex ITSs. To facilitate modularity and change management distributed technologies like services and agents are used in ITSs. Well known examples of agent based ITSs are Ines system for nurse education (Hospers et al., 2003), Formal Languages and aUTomata Education system FLUTE (Devedzic et al., 2000), IVET virtual training environment (de Antonio et al, 2005) and WADIES – a Web- and agent-based adaptive learning environment for teaching compilers (Georguli et al., 2003). Grundspenkis and Anohina (2005) have concluded that agent based ITSs mainly implement traditional modules as sets of agents. The authors have defined customizable set of agents that implements the system as a set of distributed components and at the same time maintains the traditional

idea of ITS's modules separating different types of knowledge. Nevertheless, there is a need for a distributed architecture, because the set of agents only defines ITS components and does not solve architectural problems of ITSs.

As of author's knowledge no specific service oriented architectures (SOA) for ITS development exist, except the ones presented further in the chapter. The only known distributed ITS architectures are a few multi-agent architectures for ITS development. These architectures mainly consist of agents from the set of agents defined by Grundspenkis and Anohina. The architectures are closed in the sense that system's functionality can not be changed just by adding/removing agents from the system. Examples of such architectures are ABITS (Capuano et al, 2000), IVET (de Antonio et al, 2005) and X-genitor (Triantis & Pintelas, 2004). Several open architectures are proposed to allow adding new student agents and create a learning environment for multiple learners, for example, JADE (Silveira & Vicari, 2002) and two level multi-agent architecture for distance learning environment (Webber & Pesty, 2002). Still, these architectures are open only for new student agents to join the system and are closed for any other types of agents. As a consequence there is a need for architectures enabling usage of all advantages of distributed technologies, including the possibility to change system's functionality by just adding and/or removing distributed components from the system. The chapter describes such ITS architectures. The remainder of the chapter is organized as follows. The Section 2 gives a brief introduction to ITSs by describing the tutoring process carried out by ITSs and presenting the traditional architecture of ITSs. The Section 3 is dedicated to agent based ITSs. It analyses the related work about the agent based ITS architectures and describes open holonic multi-agent ITS architecture. The Section 4 compares intelligent agents to web services. It analyses the lessons learned in the development of holonic multi-agent architecture and possibilities to apply them to SOA. The Section 5 describes service oriented ITS architecture that implements each module as a set of services. The Section 6 proposes hybrid architecture that includes both agents and services. Agents implement higher level deliberative components while services realize lower level reactive components. The Section 7 concludes the chapter.

2. Intelligent tutoring systems

ITSs can be defined as computerized systems used for tutoring and having the following characteristics (Anohina, 2007): (1) they use principles and methods from artificial intelligence (like, knowledge representation, reasoning, natural language processing and machine learning); (2) are adaptive systems that adapt the tutoring process to the characteristics of the learner, so carrying out adaptive or individualized tutoring; (3) simulate human tutor; (4) are based on the cognitive theory. One of the main characteristics of ITSs is that they try to simulate human tutor to implement adaptive tutoring. To cover all activities done by the human tutor the system has to do the following tasks: generate curriculum, provide learning materials and problems for the learner to solve in each topic, evaluate learner's knowledge and give meaningful feedback to the learner to help him/her improve his/her knowledge (Lavendelis, 2009). In adaptive tutoring these activities must be carried out adaptively – each learner should receive individual approach that fits his/her characteristics and/or preferences. Curriculum should be generated to meet the needs of individual learner as well as materials and problems should be adapted to each learner.

Simulation of the human tutor is a complex task for the system. It requires intelligent choices and actions to be made. All actions by the teacher, like creation of the curriculum, choice of the appropriate learning materials or evaluation of the learner's knowledge are complex actions and require intelligence. As a consequence, various intelligent mechanisms are needed for an ITS to simulate such actions. Intelligent mechanisms used to implement adaptive tutoring vary from system to system. Still, the main types of knowledge used in different ITSs are the same. Knowledge about the domain or subject is needed to know what to teach. Knowledge about the learner is needed to know whom to adapt and knowledge about the tutoring process is needed to know how to teach and how to adapt to the learner's characteristics. The goal of ITSs is to use the above-mentioned three types of knowledge to carry out adaptive tutoring. Usually each type of the knowledge is used by different intelligent mechanisms. It is beneficial to define components corresponding to the three main types of knowledge used in ITSs. As a consequence traditional modular ITS architecture consists of three modules, namely, the expert module, the student diagnosis module and the tutoring module, all together named traditional trinity (Grundspenkis & Anohina, 2005). Additionally the communication module is added to manage the user interface, resulting in the modular architecture consisting of 4 modules. The modular architecture is widely used in intelligent tutoring systems, for example, in Ines (Hospers et al, 2003), AlgeBrain (Alpert et al, 1999), FLUTE (Devedzic et al, 2000) and IKAS (Vilkelis et al, 2009) systems. Modules have the following features (Grundspenkis & Anohina, 2005):

- *The expert module* represents the domain expert's knowledge and includes problem solving characteristics. The task of the module is to solve domain problems. It serves as a standard to compare learner's knowledge to.
- *The student diagnosis module* collects information about learner's knowledge and misunderstandings, creating the student model.
- *The tutoring module* holds teaching strategies and instructions to implement tutoring process. The primary tasks of this module are controlling selection, sequencing and presentation of learning material that is most suitable for the learner, determining the type and contents of feedback and help, and answering learners' questions. Strategies contained by the module must be adapted to the needs of each individual learner without any help of humans. The goal of the module is to reduce the gap between learner's knowledge and expert's knowledge as far as possible or in ideal case eliminate the gap completely.
- *The communication module* is the only module interacting with the learner. It has to manage the user interface of the system. It perceives all learners' actions, receives all requests from learners and forwards them to other modules. It is responsible for presentation of all kinds of information (curriculum of the course, materials, problems, feedback, etc.) to the learner, too.

Despite the acceptance of the modular architecture its main drawback is insufficient modularity to build complex adaptive ITSs because the modules have many tasks. Distributed computing technologies, namely, services and agents are used to split the higher level modules into lower level components to increase modularity of ITSs. Moreover, the functionality of ITSs includes many functions and corresponding pieces of code that differ in systems for various courses and even may be needed to change for the same course. Usage of distributed technologies also enables implementation of open ITSs allowing introduction of new functionality without changing existing code. The remainder of the chapter analyses use of the two types of distributed technologies in the architecture of ITSs.

3. Agent based intelligent tutoring systems

Majority of agent based ITSs use the same approach. They implement traditional ITS modules as sets of agents. Popularity of the approach is based on the fact that each module contains functions that can be grouped in logical components. The defined agents correspond to these components. Modules may consist of various numbers of logical components. Each module can be realized as a single agent, a few agents or as a multi-agent system with its own architecture. Still, each module has some basic agents that with needed modifications are used in majority of agent based ITSs. The general set of agents to implement modules of ITS defined by Grundspenkis and Anohina (2005) is the following.

The goal of the *student diagnosis module* is to collect and maintain information about each learner. While part of this information is known before starting the tutoring and does not change during the tutoring process, other parts, like the knowledge level, change during the tutoring process and have to be collected by the agents of the module. Agents have to collect information about the learning process, various characteristics of the learner (emotional, cognitive and character). The following agents are used to collect information (Grundspenkis & Anohina, 2005):

- The knowledge evaluation agent that evaluates the level of the learners' knowledge and skills. Mainly it is done based on learner's results in the tests and problems.
- The psychological agent that collects information about the learner's preferences, learning style and emotions during the learning process.
- The cognitive diagnosis agent determines and registers learner's mistakes. It is also responsible for determining the causes of the learner's mistakes.
- The interaction registering agent registers history of learner's interaction with the system and follows the usage of the system's features. For example, if a learner does not use some important features of the system, the agent may suggest using them.

Agents that build the *pedagogical module* have to create and modify (when needed) the curriculum, provide learning materials, generate problems and provide the feedback to the learner. All these tasks are more or less independent, thus they are assigned to separate agents and the module consists of the following agents (Grundspenkis & Anohina, 2005):

- The curriculum agent that creates, evaluates and modifies the curriculum if needed.
- One or a few teaching strategy agents that implement teaching strategies. These agents provide learning materials in each topic according to the teaching strategy. It is needed to vary teaching strategies, because learners have different learning styles. Some learners prefer to receive an example first while others prefer to read theory and only then receive an example (Bicans et al, 2011).
- The problem generation agent that generates tasks, questions and problems. In the remainder of the paper all tasks, questions and problems given to the learner to evaluate his/her knowledge will be named problems.
- The feedback agent that provides feedback to a learner after he/she has finished the problem solving. The agent is also responsible for providing hints and explanations requested by a learner.

The *expert module* is responsible for solving problems and tasks in the domain taught by the system. This module consists of one or more expert agents that solve problems of the learning course. Usually each expert agent is responsible for problems in one topic or of one type. The *communication module* can be implemented as the interface agent and/or animated pedagogical agent (Grundspenkis & Anohina, 2005). The interface agent is responsible for

all tasks concerning the communication with a learner. It is responsible for the whole user interface. This agent may be responsible also for teacher's user interface. Similarly, the animated pedagogical agent is responsible for interactions with a learner. The main difference from the interface agent is that the animated pedagogical agent is two or three dimensional animated person that uses voice, gesture and mimics to interact with the learner. Animated pedagogical agents are made to be perceived as teachers. They make the learning process more interesting to a learner. Additionally to the abovementioned tasks of the communication module, namely, displaying curriculum, providing learning materials, showing the knowledge evaluations and feedback the animated pedagogical agents accomplish one important task of the teacher – they motivate learners to study. The agent achieves it by showing systems emotions, so making the interaction with the ITS more human like. The agents of the communication module are the only agents that communicate with the learner and thus have to carry out all communication tasks. They have to present all information that has been prepared by all other agents of the system like curriculum, learning materials, problems, feedback, etc.). These agents also have to register relevant actions done by the learner and forward them to the agents of other modules that are interested in the corresponding actions. The agents also control the work of all communication devices used to interact with the learner: the user interface, keyboard, mouse as well as various specialized devices like data gloves, video cameras and other equipment of the learning environment (Grundspenkis & Anohina, 2005).

Additionally the set of agents may contain manager agents that coordinate other agents of the set. These agents may be created differently. One manager agent can be created for each module or for a whole system. The Figure 1 depicts the described set of agents. The manager agents are not included in the set, because they can be defined differently. The described set of agents can be customized to meet the functionality of every individual ITS. Some agents can be removed from the set if the system does not need the corresponding functionality. If needed, some additional agents can be added to implement additional functionality that usually is connected with the problem domain, for example, the patient agent in the nurse

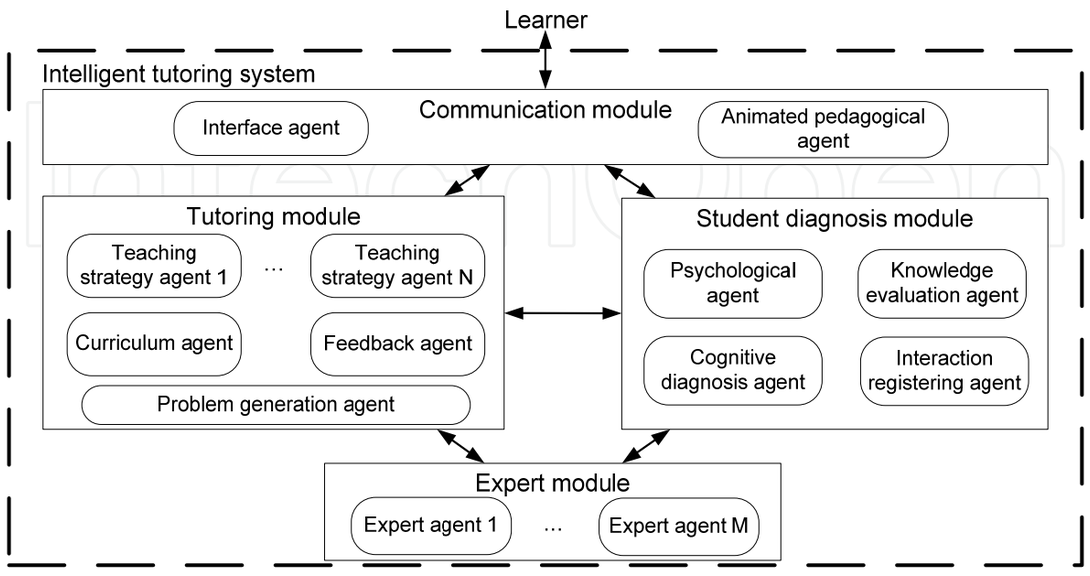


Fig. 1. The set of agents used to implement ITSs (Grundspenkis & Anohina, 2005).

education system Ines (Hospers et al, 2003). The set of agents can be taken as a basis and modified instead of designing the multi agent system from scratch. Nevertheless, implementation of ITSs based on the described set does not eliminate all drawbacks of modular architecture. Firstly, the set of agents does not define interactions among agents. Every agent may interact with any other agent, making the complexity of the interactions grow exponentially if the number of agents grows. Secondly, direct implementation of the set of agents does not ensure openness of the system. Introduction of new functionality into the system is impossible without changing existing components. Thirdly, some agents still are large and have many tasks, reducing modularity of ITSs. Reuse of large-scale agents is complex, too, because agents have many tasks and it is unlikely that there will be two systems that will need all these tasks unchanged. At the same time direct reuse of single task is impossible too, because it is only a part of the component.

A few distributed ITS architectures that facilitate ITS design by providing agents, their tasks and interactions, exist. Majority of them modify or extend the set of agents described above. Examples of such architectures are the multi-agent architecture for distance education systems (Dorca et al, 2003), the IVET architecture (de Antonio et al, 2005), the ABITS architecture (Capuano et al, 2000), the JADE architecture (Silveira & Vicari, 2002) and the X-Genitor framework (Triantis & Pintelas, 2004). Nevertheless, they do not solve the architectural problems in the ITSs. As they are similar to the defined set of agents, they have the same problems with modularity and reuse. Some of the architectures are defined to be open, for new agents of one type. New interface agents can be added to the system to represent new learners. Examples of open architectures in this sense are X-GENITOR and JADE. These open architectures facilitate group learning and collaborative learning in ITSs. Still they are not open for new components that add new functionality to the system without changing existing code. For example, it is not possible to add new agent that is capable to generate new types of problems, but the code of existing problem generation agent has to be changed. Moreover, it has to be analysed how the changes made in the problem generation agent will affect other agents. As a consequence these architectures do not allow benefiting from one of the main advantages of agent technologies – open and distributed computing.

Additionally, existing architectures have the following drawbacks. Agents have many tasks making the development complex. Agents used in ITS are not capable to deal with problems that can be solved by multiple agents, like resolution of various conflicts. Example of such tasks is choice of appropriate learning strategies using various criteria. As a consequence, it can be concluded that known agent based ITS architectures do not take full advantage of the distributed and open nature of multi-agent systems. Thus, there is a need for distributed ITS architectures to enable full usage of all these advantages in ITS development. The remainder of the section presents holonic multi-agent architecture that tries to implement open and highly modular ITSs as well as the MIPITS system (Lavendelis & Grundspenkis, 2010), which is an agent based ITS developed using the described architecture.

3.1 Multi-agent architectures

Small scale multi-agent systems can be developed just by defining agents and specifying interactions among them. This approach starts to fail if the number of agents increases, exponentially increasing the complexity of interactions, because every pair of agents may interact each other. To solve this problem, the concepts of architecture and organization are introduced into the multi-agent systems. Additionally, these concepts are used for agents to

create teams and cooperate in problem solving. Two best known multi-agent system architectures are holonic multi-agent systems (Fischer et al, 2003) and multi-multi-agent systems (Nimis & Stockheim, 2004). The idea of *holonic multi-agent systems* is that autonomy of agents is reduced and agents are merged into holons, which appear to outside as a single entity (Fischer et al, 2003). The term "holon" (Greek word "holos" has meaning "whole" and suffix "-on" denotes "part") is adopted from biological system research done by A. Koestler (1967). Holon is a self-organizing structure which consists of substructures and is a part of larger superstructure. In terms of multi-agent systems holon or holonic agent is an agent that consists of other agents named subholons.

In holonic multi-agent systems agents form a hierarchical structure, i.e., each holon can be a part of a higher level holon and consist of lower level holons. It allows adapting the system to the structure of the domain. Hierarchy makes holons suitable for task and result sharing. If the holon has a task assigned to it, the task can be decomposed into some subtasks that are assigned to subholons, which can decompose them into the next level subtasks and so on. If the agent receives a task that it is not able to accomplish it can also find other agents to create a holon with, to accomplish the task together (Fischer et al, 2003).

The autonomy of agents that form a holon is usually reduced by giving one agent (called head or head agent) the privilege to do resource and task allocation in the holon. It can have partial or total control over other agents. Agents that are parts of the holon, but are not head agents are called body of the holon (Gerber et al, 1999). To outside the holon appears as a single entity represented by the head of the holon. The body agents do not communicate outside the holon. So, holons have an interface (head) and they can be developed separately as modules of traditional software engineering. Holons also make change implementation easier, because changes of an agent in one holon directly affect only agents from the same holon. Lavendelis & Grundspenkis (2008) have concluded that ITSs comply well with the main criteria of holonic domains defined by the authors of the holonic approach (Gerber et al, 1999), namely operator abstraction, hierarchical structure, partial decomposability and cooperative system. Thus the holonic multi-agent systems are suitable for ITSs.

The second well-known multi-agent architecture - *multi-multi-agent system* has been developed inside the Agent.Enterprise methodology (Nimis & Stockheim, 2004). The main goal of the project is integration of several multi-agent systems. It is similar to the holonic multi-agent systems in the sense that both architectures propose to create systems that consist of subsystems - holons and multi-agent systems, respectively. Multi-multi-agent systems offer the concept of the gateway agent that accomplishes routing and message conversion between different message formats used in different multi-agent systems. Interactions among agents of various multi-agent systems are enabled. Still, comparing holonic multi-agent systems and multi-multi-agent systems one may conclude that in the context of ITSs there are significant advantages of holonic multi-agent systems. The main of them is that heads of holons unlike the gateway agents can accomplish not only mediator tasks, but also many other tasks. It allows implementing great part of the intelligent mechanisms into the heads of holons. One more important advantage is that holons can be dynamically changed and allow to build open systems. For other advantages of holonic multi-agent systems see (Lavendelis & Grundspenkis, 2011), where it is concluded that the holonic multi agent systems are more suitable for the ITSs and should be used to develop a specific agent based ITS architecture. The next subsection describes such architecture.

3.2 Open holonic multi-agent ITS architecture

Despite the fact that holonic multi-agent systems are suitable for agent based ITS development it is not clear how to use holons in ITSs without any specific architecture. Such architecture is proposed in (Lavendelis & Grundspenkis, 2008). From outside it is a single holon. The only agent that represents the system outside is the interface agent, which implements all interactions with the learner. So, it is the head of the higher level holon and the only agent implementing the communication module. The remaining modules are implemented as subholons and are included in the body of the higher level holon. Each module can be realized as one or more holons. Modules that carry out wide functionality (pedagogical module and student module) are implemented as multiple holons. The functionality of the student module contains the following two groups of functions: learner's knowledge evaluation functions (usually evaluation of the learner's solution) as well as building and maintaining of the student model. Thus the student model is realized as two holonic agents – student modelling and knowledge evaluation agent. The pedagogical module similarly to the above defined set of agents consists of the following holonic agents: the curriculum agent, the teaching strategy agent, the problem generation agent and the feedback agent. The expert module is implemented as a single agent – the expert agent. So, the higher level of the architecture contains 7 body agents (see Figure 2). The higher level agents have the same functionality as the agents from the above defined agent set, except three agents of the student modelling holon are merged into one higher level holon that has all the functionality of the three merged agents. Interactions among the higher level holons are defined in two degrees of detailed elaboration. Firstly the acquaintance diagram is developed showing which holons have any interactions. Secondly, messages sent among agents are defined in the interaction diagrams (Lavendelis, 2009).

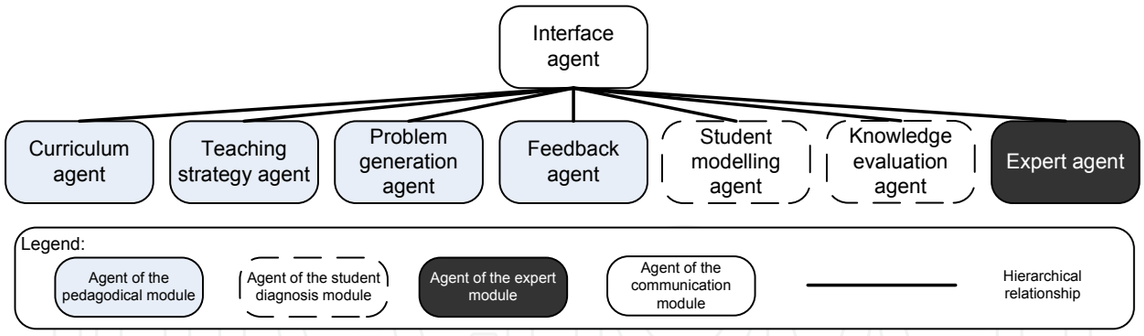


Fig. 2. The higher level of the holonic ITS architecture.

Higher level agents are realized as holons that consist of a single head agent and some subholons (body agents). The head of the holon is responsible for the coordination of all subholons that is done by centralized planning and task allocation to the body agents. The heads use directory facilitator service to find body agents and their capabilities. Heads are responsible for tasks that need one unique performer, like, the head of student modelling agent is responsible for building complete student model and providing it to other agents.

The architecture is open and contains two types of holons: open and closed. Open holons consist of the head and a certain type of body agents, however, the number and exact instances of body agents are not defined during the design of the system and can be freely changed during the maintenance and runtime, so modifying the system's functionality.

Body agents have to register their services at the directory facilitator agent. Heads of open holons use the directory facilitator agent to find actual body agents in open holons. Closed holons consist of agents that are specified during the design and can not be changed during the runtime of the system. Body agents are responsible for certain types of tasks that are subtasks of the holon's tasks. Body agents of closed holons usually carry out principally different tasks. Contrary, body agents of open holons are responsible for different subtypes of one type of tasks. So, it is possible to add a new body agent to an open holon that is responsible for a new subtype of the task. For example, each body agent of an open holon is responsible for generation of one type of problems. New type of problems can be introduced by adding new body agents. Student modelling, curriculum and feedback agents are closed holons, while problem generation, teaching strategy, expert and knowledge evaluation agents are open holons. The interactions in all open holons as well as algorithms used by heads are similar. The main steps carried out to fulfil the task of the holon are the following:

1. The head of the holon receives request to carry out a task. If the task can be done by the head of the holon, it is performed and the result is sent to the requester and the algorithm ends. Otherwise, it continues with the Step 2.
2. The head of the holon queries the directory facilitator agent to find all body agents.
3. If the directory facilitator has found at least one appropriate agent, the head queries the body agents. Depending on the holon only one body agent or all appropriate agents are queried. One agent is queried if only one type of subtasks suites the request. All agents are queried if all types of subtasks suite the request. If no body agents are found a system error is generated.
4. After receiving the request from the head of the holon, body agents carry out the task and send the results to the head.
5. If more than one body agent is queried during the Step 3, the head waits for replies from all of them or until the time-out has occurred. Then, it finds the most appropriate result provided by the body agents. For example, if each body agent has generated a problem of some type, then the head chooses the most appropriate one for the learner.
6. The head forwards either the only result or the result chosen during the Step 5 to the agent that sent the request during the Step 1. The head may also send the result to some other agents, if needed.

Interactions in the closed holons are simpler than in open ones. There is no need to use directory facilitator service, because all body agents and their services are known. After the head of the holon receives the request for some task, it just has to find the corresponding performer for the task. Usually there is only one such performer in closed holons. It might be the head of the holon (usually for global tasks of the holon, like building full student model in the student modelling holon) or any of the body agents. If it is one of the body agents, it is requested to carry out the task and its result is forwarded to the initial requester.

Open holons allow addition of new functionality of certain types without changing existing code. It is possible to add new body agents in the open holon and so add new types of subtask that holons are capable to accomplish. For example, each body agent of the problem generation holon generates some type of problems. To add new type of problems to the system, a new body agent is added to the open holon. Still, it is not the only agent that has to be added to the system, to add new type of problems. New body agents must be added to all open holons where each type of problems is treated differently. Such holons are expert agent's holon and knowledge evaluation agent's holon, because each type of problems must be solved and evaluated in different ways. Additionally, all new functionality must be

provided to the learner. For, example new type of the problems must be shown to the learner by the interface agent. Thus the functionality of the interface agent should be extendable, too. To achieve it, the conception of hierarchical holonic multi agent systems is extended by implementing the head of the higher level holon (the interface agent) as an open holon. The most common tasks in the user interface are done by the head of the interface agent's holon, while other tasks are done by the body agents. So, a corresponding body agent must be also added to the interface holon to add new type of problems. The whole holonic multi-agent ITS architecture is given in the Figure 3.

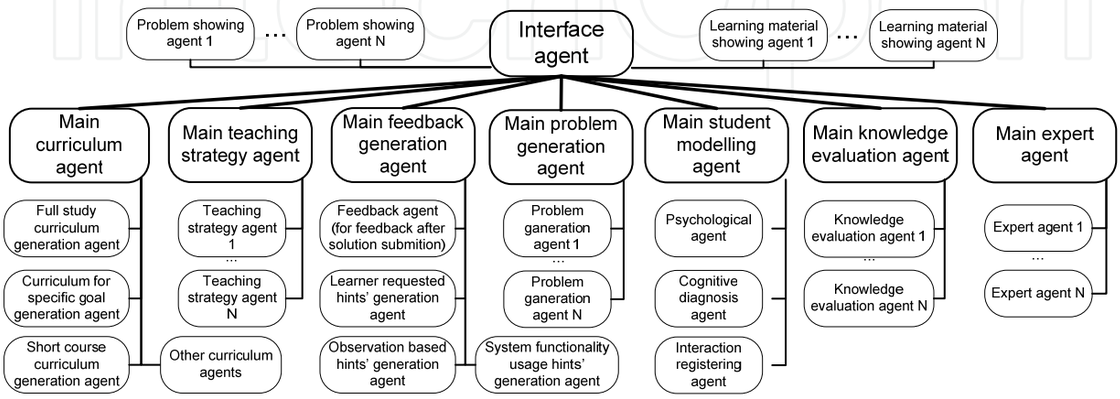


Fig. 3. Holonic ITS architecture (modified from (Lavendelis & Grundspenkis, 2008)).

3.3 Case study – The MIPITS system

The holonic multi-agent ITS architecture is approbated in the case study. An ITS named MIPITS has been developed using the proposed architecture (Lavendelis & Grundspenkis, 2010). The system supports the traditional classroom tutoring for the undergraduate course "Fundamentals of Artificial Intelligence" taught at Riga Technical University, because after attending lectures a learner can repeat the theory given at the lectures using the learning materials provided by the system and assess his/her knowledge with problems provided by the system. After finishing each problem, learner's knowledge is evaluated and he/she receives feedback about his/her solution explaining his/her mistakes. The main focus of the system is on problem solving. The following types of problems are included in the initial version of the system: (1) Different types of tests, including single and multiple choice tests and tests, where a learner has to write the answer by him/herself. (2) Search algorithm problems, where a learner has to do a state space search using the specified algorithm and lists OPEN and CLOSED (Luger, 2005). (3) Two person game problems, where a learner has to apply the MINIMAX algorithm or Alpha-Beta pruning to the given state space (Luger, 2005). The system adapts the problems to the following learner's characteristics. Difficulty level of the problem is adapted to the learner's knowledge level, practicality of the problem and size of the problem are adapted to the corresponding learner's preferences provided by the learner during the registration. Finally, the system follows the types of problems given to the learner and tries to minimize the repetition of problems of the same type.

The adaptation of problems is carried out in the problem generation holon showing that such holons can be efficiently used to implement adaptive tutoring. Previously described general algorithm of the heads of open holons is used to generate the most suitable problem. The suitability of the problems to the learner is measured by calculating the weighted sum

of the differences between desired values of criteria and real ones. The problem with the minimal weighted sum is considered to be the most suitable one. The following equation is used to calculate the appropriateness of the problem (Lavendelis & Grundspenkis, 2010):

$$A = -(\left|\text{dif}_{\text{pref}} - \text{dif}_{\text{r}}\right| * \text{c}_{\text{d}} + \left|\text{s}_{\text{pref}} - s_r\right| * \text{c}_{\text{s}} + \left|pr_{\text{pref}} - \text{pr}_{\text{r}}\right| * \text{c}_{\text{p}} + \text{f}_t * \text{c}_{\text{f}}), \text{ where} \quad (1)$$

dif_{pref} – the preferred difficulty of the task;
dif_{r} – the real difficulty of the task;
c_{d} – the weight of the difficulty;
s_{pref} – the preferred size of the problem;
s_{r} – the real size of the problem;
c_{s} – the weight of the size;
pr_{pref} – the preferred practicality;
pr_{r} – the real practicality of the problem;
c_{p} – the weight of the practicality;
f_t – the frequency of problem's type;
c_{f} – the weight of the frequency.

Weights are determined empirically and are the following: $\text{c}_{\text{d}}=2$, $\text{c}_{\text{s}}=3$, $\text{c}_{\text{p}}=3$, $\text{c}_{\text{f}}=6$, because with these weights all criteria have significant impact on the appropriateness.

As the MIPITS system has specific functionality with the main focus on problems, the general architecture is customized to meet the particular requirements of the system. There is only one type of materials in the MIPITS system; thus, there is no need for open holons dealing with the materials. The corresponding agents are implemented as monolith agents instead of holons. The main teaching strategy agent generates all materials and the main interface agent visualizes them in the interface. Similarly, there is only one type of the curriculum and it is not changed after generation. The student modelling and feedback agents also do not have complex functionality in the MIPITS system, thus these agents also are implemented in a monolith way. The agents that deal with the problems are implemented as open holons to allow adding new types of problems. At the initial version of the system agents for the described three types of problems are implemented in each of the open holons, namely, the problem generation, the knowledge evaluation, the expert and the interface holons. For example, the problem generation holon contains the head (the main problem generation agent), the test generation agent, the search algorithm problem generation agent and the game tree problem generation agent. The actual architecture of the system is shown in the Figure 4. The heads of open holons are denoted with grey colour.

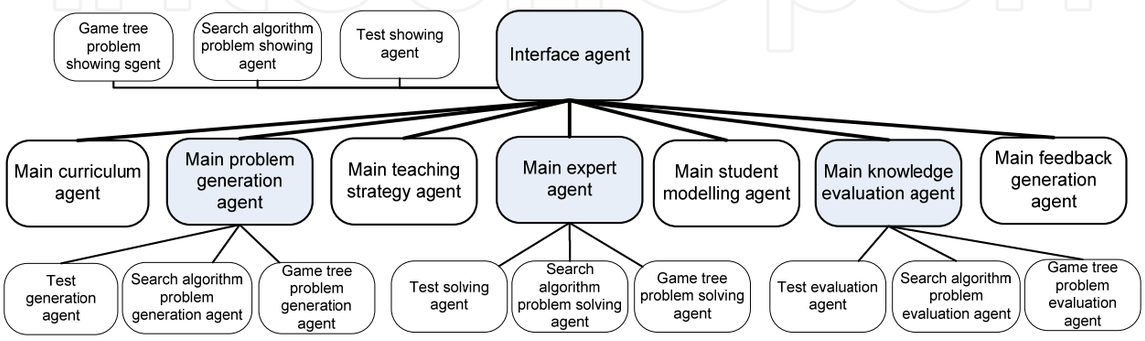


Fig. 4. The architecture of the MIPITS system.

The interface of the MIPITS system consists of two main parts (see Figure 5). The interface is in Latvian, because it is the language of the course. The left side of the main window shows the curriculum. It is shown as a hierarchy. The higher level shows the modules of the course, while lower level shows topics. When the learner chooses the topic to start learning the corresponding learning material is shown in the right side of the interface. The right side is used also to show problems that are given to the learner when he/she submits that he/she has finished the theoretical material and is ready to evaluate his/her knowledge. The layout of the right side differs for various problems. This part of the interface is created and managed by the corresponding body agents of the interface holon. The example of the problem is given in the Figure 5. It is an interface for two-person games algorithm MINI-MAX (Luger, 2005). It is created by the two-person games problem visualization agent. It has typical structure for the problems used in the MIPITS system. The top part of the right side (denoted with 1 in Figure 5) contains the description of the problem and defines what a learner has to do. The middle part (denoted with 2 in Figure 5) contains graphical information. In this case it is the game tree. The bottom part (denoted with 3 in Figure 5) contains controls for student to solve the problem. For the particular type of tasks it contains controls for assigning values to the vertexes in the state space. The student has to show how the hierarchical evaluations of the vertexes change during the execution of the algorithm.

The MIPITS system is open – it can be extended with new types of problems by adding four new body agents to corresponding holons: a problem generation agent, an expert agent, a knowledge evaluation agent and a problem visualisation agent for the particular types of problems. The extendibility of the MIPITS system has been proven by adding new type of problems to already running system without changing existing code. A topic about propositional logic and inference was added (Lavendelis & Grundspenkis, 2011).

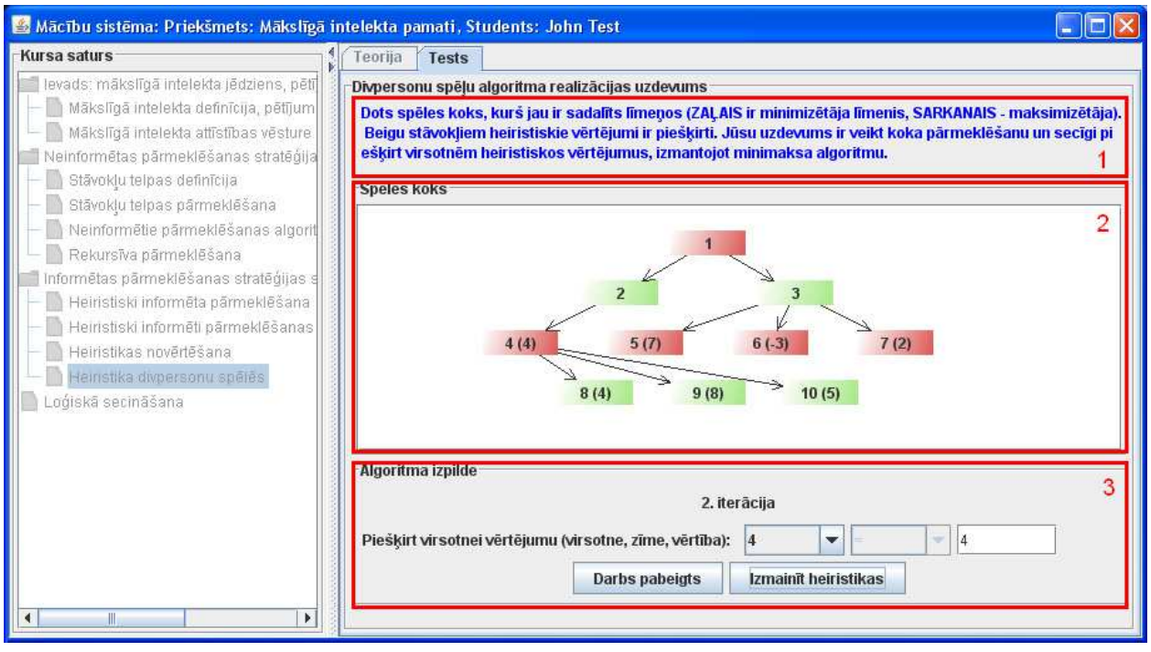


Fig. 5. The user interface of the MIPITS system.

4. Intelligent agents and web services

Despite being two independently developed technologies, services and agents have many similarities, because both of them are based on the principles of distributed computing. Thus it is worth to analyze how the lessons learned from the agent based architectures can be applied to the service oriented ITSs. Lavendelis and Bicans (2011) have indentified the most important similarities between agents and services in the context of ITSs:

- High modularity. Both services and agents offer high modularity, because they support systems that consist of small distributed entities.
- Openness. Both technologies allow dynamical addition of components (agents and services, respectively) to the system, to implement changes into the system. Both technologies offer mechanisms to find newly added components.
- Reactiveness. Reactive agents can be considered as services. For example lower level agents from the holonic agent architecture provide specific service upon request.
- Usage of protocols. Both agents and services use some kind of protocols to interact.

Still agents and services have come from two different fields of research. Intelligent agents have been proposed by artificial intelligence researchers, while services have been introduced by the software engineering specialists. Lavendelis & Bicans (2011) state that there are the following significant differences between agents and services that have to be taken into consideration during the ITS development:

- Reasoning capabilities. Services usually have no reasoning capabilities while agents are considered as reasoning entities.
- Autonomy. Agents are autonomous entities that are capable of proactive actions. Services are strictly reactive and have no autonomy. They are not capable to carry out any goal driven actions.
- Industrial acceptance. At the moment services appear to be industrially accepted and widely used technology. At the same time, agents are mainly used in research projects.
- Complex interaction protocols. Various interaction protocols like negotiations, auctions, etc, can be used in the multi-agent systems to reach agreements and solve problems. Agents have social capabilities to participate in these protocols. Services use simple protocols and have no social capabilities to participate in any complex interactions.

One can conclude that both distributed technologies can be used for ITS development and both of them have their advantages and disadvantages. ITSs contain components that are more suitable to develop as services and components that are more suitable to develop as agents. The main criterion to determine which technology is more suitable is the following. If there is no proactivity one can use services instead of agents to implement components of ITS and benefit from the industrial standards of SOA. Actually, services may be used in the same way as agents to implement modules of ITS. The remainder of the section analyzes how the similarities of the agents and services allow to use the lessons learned in the agent based ITS research in the service oriented ITSs. Additionally, some common principles of software architectures are analysed to include them in the architecture of ITSs.

General software architecture concepts say that the user interface should be separated from the logical part of the system. Usually it is done by creating a layered architecture that also allows separation of components that are dedicated to the repositories (like the learning object repository) and such fundamental technologies like video streaming. So it is beneficial to create a layered architecture that consists of three layers (Lavendelis & Bicans, 2011). *Layer one or the lower layer* contains repositories and fundamental technologies used in the

ITS. *Layer two or the logical layer* is a logical part of the system. It contains all three main modules of the ITS's modular architecture, namely, the student module, the tutoring module and the expert module. *Layer three or the presentation layer* contains all technologies needed to present the contents to the learner. It contains the communication module. The idea of layered architecture complies well with modular and agent oriented architectures. It allows keeping the traditional ITS modules and only separates components working with the repositories. These components usually are completely reactive and have no intelligence. Thus the layered approach allows to separate the intelligent part of the ITS from other parts.

The main lessons learned in the holonic multi-agent ITS architecture are the following. The implementation of the architecture showed that it is beneficial to keep the modules of the traditional ITS architecture and to implement them with distributed components. It allows keeping the main advantage of the modular architecture – the separation of the intelligent mechanisms that work with different types of knowledge. It allows for each component to process only one type of knowledge and abstract from other types. So the modules should be kept in the ITS architecture regardless of the technology used for implementation.

The distributed implementation gave two advantages to the architecture. Firstly, the realization using small-scale components increased the modularity of the system. Secondly, the introduction of holons decreased the coupling of the system, because each body agent is allowed to interact only with agents of the same holon. The head of the holon serves as an interface of the holon. It removed traditional drawback of distributed systems that complexity of interactions increases rapidly if the number of components increases. So, it may be concluded that some forms of organization should be included in distributed architectures. Despite, services do not support such hierarchical structure as holons, the organization and, as a consequence, increased modularity and decreased coupling can be sustained in the service oriented architecture in the following way. One or a few interface services should be introduced in each module. These services would fulfil the role of the heads of the higher level holons. If the higher level components are created the same way as in the holonic architecture, the expert module would have one main service, while the tutoring module, the student module and the communication module would have more than one. The interactions in the system are organized in the following way. The main service receives service requests from other main services. It uses the service registry to find other services (named lower level services) of the module and forwards the request to the appropriate service. The lower level service does its job and returns the result to the main service, which forwards it to the initial requester. So, like in holonic architecture, interactions take place only among main services and between main services and the lower level services of the corresponding module. It decreases the coupling of the system and facilitates its modularity. For example, the main interface service sends the request for problem in a topic to the main problem generation service, which finds the service that can generate a problem in the topic or even the most suitable problem for particular learner.

The holonic architecture implements modules of ITS in an open manner. The implementation of the MIPITS system proved that openness is important feature if the system is modified by adding some new features (Lavendelis & Grundspenkis, 2011). New type of problems that needs completely different processing in all holons was added to the system without changing anything in existing code. Thus, the open implementation of distributed ITSs makes the change implementation and adaptation of the ITS to new course or modifications of the existing course much easier.

Implementation of the multi-agent architecture proved that majority of intelligent adaptation mechanisms are included in the heads of the holons. Heads are not only the mediators, but they make intelligent choices, using different algorithms and reasoning. The heads of the higher level holons are the most important components to provide adaptivity. For example, body agents are capable to generate problems, but the head of the holon chooses the most appropriate problem to the particular learner. The body agents mostly work with repositories. They implement certain actions like extracting or generating materials or problems. So these agents fit more the concept of the service instead of intelligent agents because they are not proactive and execute reactive behaviour by accomplishing some task upon the request of the head of the holon. All proactive and intelligent actions are carried out by the heads of the holon, that fit the logical layer of the general architecture. The body agents on their hand implement the interface between the layers one and two because these agents are used by the agents from the second layer to access the first layer – the repositories.

To conclude, the main advantages of the holonic architecture can be sustained in the service oriented ITS architecture. Still, some features of agent architecture can not be implemented using services. Services do not have built in reasoning mechanisms that are natural to agents. So by implementing components as services the built in intelligent mechanisms are lost and it is not clear how to implement such mechanisms as reasoning inside the services. Potentially this is the main disadvantage in moving from multi-agent to service oriented ITS architecture. If some proactive behaviour is needed, then usage of agents is preferable. Still intelligent mechanisms and proactivity are mostly included in the logical part of the ITS and are rare in other parts. Thus there are components that can be successfully built using services instead of agents. As a consequence there is a point in both pure service oriented ITS architecture and hybrid ITS architecture, where deliberative components are implemented as agents while reactive components are implemented as services. Such architectures are presented in the following two sections.

5. Service oriented intelligent tutoring systems

It has been concluded in previous sections that there are several advantages to implement components of ITSs as services that are better known to software developers than intelligent agents. At the same time the principles of holonic multi-agent ITS architecture can be reused in the service oriented ITSs development. The following ideas have been identified to be adopted from the agent based ITSs and reused in the service oriented ITSs. Firstly, each module is implemented as a set of distributed components. Secondly, only some of the components interact with other components outside the corresponding module to decrease coupling of the system. Thirdly, the ITS is implemented as an open system and new functionality can be added to the system by adding new distributed components, in particular services. The service oriented ITS architecture that realizes all the identified principles is described in the following subsection. The case study of the architecture is given in the section 5.2 by presenting an ITS developed using the architecture.

5.1 Service oriented intelligent tutoring system architecture

The service oriented ITS architecture (Lavendelis & Bicans, 2011) consists of two levels, namely the higher and the lower level. At the higher level of the architecture each module contains one or a few main services that are used from the outside of the module. These

services implement interfaces of the modules. At the lower level unlimited number of other services may be included in each module. These services are used only by the main services of the corresponding module. They are not used from the outside of the module. The service oriented ITS architecture implements the main ideas of the layered architecture described in the previous section. Services of the communication module implement the third layer of the layered architecture. All main services of the remaining three modules (the pedagogical module, the expert module and the student diagnosis module) implement the logical part of the system and thus correspond to the second layer. Lastly, all lower level services implement all particular actions with all repositories and with all fundamental technologies like video streaming. Thus these services implement the interface to the third layer of the layered architecture.

Similarly to traditional ITS architectures *the communication module* is the only module interacting with the user. It manages the user interface and has to visualize the curriculum, learning materials, problems and feedback. It receives learner's requests and forwards them to the corresponding services. The module consists of the following higher level services:

- The main interface service, whose role is to register learner's actions and forward them to the corresponding services.
- The main material visualisation service that processes requests to visualize learning materials of different types.
- The main problem visualisation service that similarly to the main material visualisation service processes requests to visualize problems of different types.
- The curriculum visualisation service visualizes the curriculum of the course.
- The feedback visualisation service that has only one function – to give feedback to a learner, when he/she has finished solving the problem or has requested a hint.

To make the architecture open for new types of materials and problems, the communication module contains two types of lower level services that interact only with the corresponding main services. Firstly, the main material visualisation service has one lower level service for each type of materials that is needed to be visualised differently, like video streams and text materials. If new types of materials are introduced in the system, the corresponding lower level service will be added to the module. Secondly, the main problem visualisation service uses the lower level services corresponding to the types of problems used in the system. Thus the main material and problem visualisation services are only mediators – they only have to find appropriate lower level services to visualize materials and problems respectively. Lower level services also have only one task – to visualize the particular type of materials/problems. The higher level services that do not have corresponding lower level services do all tasks by themselves, for example, the curriculum visualisation service is responsible for visualisation of the curriculum. Still, the general architecture can be customized by adding new types of lower level services, if needed. For example, if various types of curriculum are used, corresponding lower level services can be added.

The services of the *pedagogical module* have the same tasks as the module has in the modular architecture. To accomplish them it consists of the following services:

- The curriculum generation service has to provide the curriculum of the course. It may be generated automatically by the service or created by the teacher and stored in the database for the service to read from.
- The main material generation service that is responsible for material generation corresponding to the chosen topic from the curriculum of the course. It uses lower level

services capable to generate particular types of materials. Each type of materials supported by the system has corresponding lower level material generation service. For example, separate lower level material generation services can be created for each of the abovementioned different formats of learning materials.

- The main problem generation service, that is responsible for problem generation. Similarly, to materials, each type of problems has a corresponding lower level service, that generates the corresponding type of problems upon the request of the main service. Such an approach implements similar openness to the multi-agent architecture. The openness of the architecture is further discussed below.
- The feedback generation service, which is responsible for providing feedback to the learner. It receives the knowledge evaluation and creates meaningful feedback that can vary from comments about answer's correctness to detailed explanation of mistakes.

The main material and problem generation services use similar algorithm to carry out interactions during the generation. In fact, majority of the main services use similar algorithm. The following steps are carried out when the main service receives a request to generate a material/problem in the particular topic. The definition of the algorithm will include also actions done by other services to illustrate all interactions:

1. The main service receives generation request.
2. The main service requests student model from the student modelling service described below to know the characteristics of the learner to adapt to.
3. When the main service receives the student model, it queries the service registry to find the corresponding lower level services.
4. If at least one lower level service is found, the request for materials or problems in the certain topic together with needed characteristics is sent to the lower level services.
5. Lower level services create or retrieve from the repository the most suitable materials/problems for the learner's characteristics in the current topic.
6. The lower level services send their results to the main services.
7. The main service receives responses from all services requested before. If more than one service provides a result, the main services chooses the most appropriate result and forwards it to the main interface service and other main services, if needed.

The *expert module* contains only one higher level service named main expert service. It is responsible for solving problems given to the learner. Each type of problems has its own second level service that solves the problem. The main service just has to find the correct second level service. The *student module* contains two higher level services, namely, the main knowledge evaluation service and the main student modelling service. The student modelling service is responsible for collecting information about a learner, his/her actions priorities and knowledge evaluations. It also creates full student model and provides it upon request of other services. The knowledge evaluation service is responsible for evaluating learner's knowledge level by comparing his/her solution of particular problem to the so called system's solution provided by the expert agent.

Similarly to the multi-agent architecture, the service oriented architecture is open in the following sense. New types of materials and problems can be added to the system without changing the code of existing services. Only new services corresponding to the new type of problem or material must be added in each component of the architecture where each type of material/problem is handled by separate service. If any new type of materials is added to the system then two services must be added to the system, namely corresponding lower

level material generation service and lower level material visualisation service. Four new lower level services must be added to introduce new type of problems. Corresponding problem generation service, expert service, knowledge evaluation service and problem visualisation service must be added to the system. The service oriented ITS architecture is given in Figure 6. Besides the described components the figure contains repositories used to store data, but links among services and repositories are omitted to keep the figure readable. The following repositories can be used in the ITS: the student data repository with personal data and student models, the course repository with data about courses and topics, learning material repository and the problem repository (for details, see (Lavendelis & Bicans, 2011).

The described openness is not the only way to customize the architecture. Specific functionality that does not correspond to any of the traditional modules may be needed, Additional separate modules can be created to include such functionalities. For example, teacher's interface is needed in any practically used ITS to modify the course by adding, removing and changing topics, learning materials and problems used in the system. Similarly, to the traditional modules additional ones are implemented as sets of services (Lavendelis & Bicans, 2011).

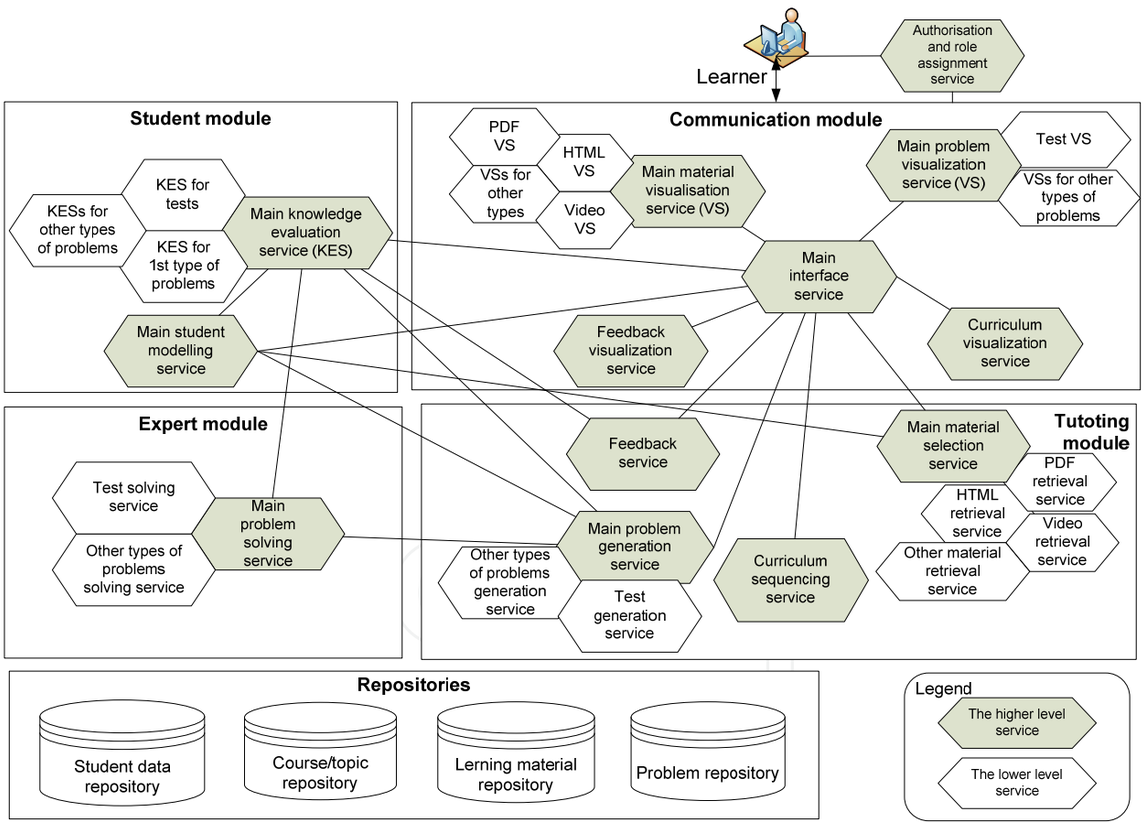


Fig. 6. Service oriented ITS architecture.

5.2 Case study

The described service oriented ITS architecture has been tested during the implementation of the ITS for the graduate course "Artificial Intelligence" taught at Riga Technical

University (Bicans et al, 2011). The course covers general artificial intelligence topics, like agents, planning, ontologies and reasoning. It has video and text learning materials with different levels of granularity. The video LOs are filmed lectures of the course. Materials cover theory, examples and tasks about the topics of the course. The aim of the ITS is to provide out of classroom tutoring, that takes into consideration learning styles of students by presenting a single or a combination of appropriate materials in each topic. The ITS stores and identifies all kinds of LOs describing them with standardized metadata. Links between topics and corresponding learning materials are not stored in the repositories. They are created dynamically. A keyword based algorithm is used to find the materials that can be used at the current topic and select the most suitable materials for the learner's preferences. LOs and topics are described with keywords and if the intersection between the sets of LO's topic's keywords is possible, the LO is considered as linked to the topic. Then only those LOs that match the preferences stored in the student model are selected from the results' set of the first step (Bicans, et al, 2011). Such an approach provides ITS with option to use more LOs from different repositories without manually linking LOs and topics. The curriculum of the course is encoded by a topic map that is a graphical knowledge representation form using topics and associations among them. In the developed ITS a topic map is used to show the hierarchy of the topics, because it visually shows the place of each topic in the course.

The general service oriented ITS architecture is customized during the development of the system to meet its specific needs. The main focus of the developed ITS prototype is on the adaptive LO presentation to learners. Therefore, only corresponding services dealing with LO selection, student model development, tutor's functionality are implemented in the current version of the system. The actual architecture of the ITS is shown in Figure 7. The communication module contains the main interface service and two more higher level services – the topic map viewer service (visualizes curriculum) and the main material visualisation service, which also has two lower level services that visualize both types of

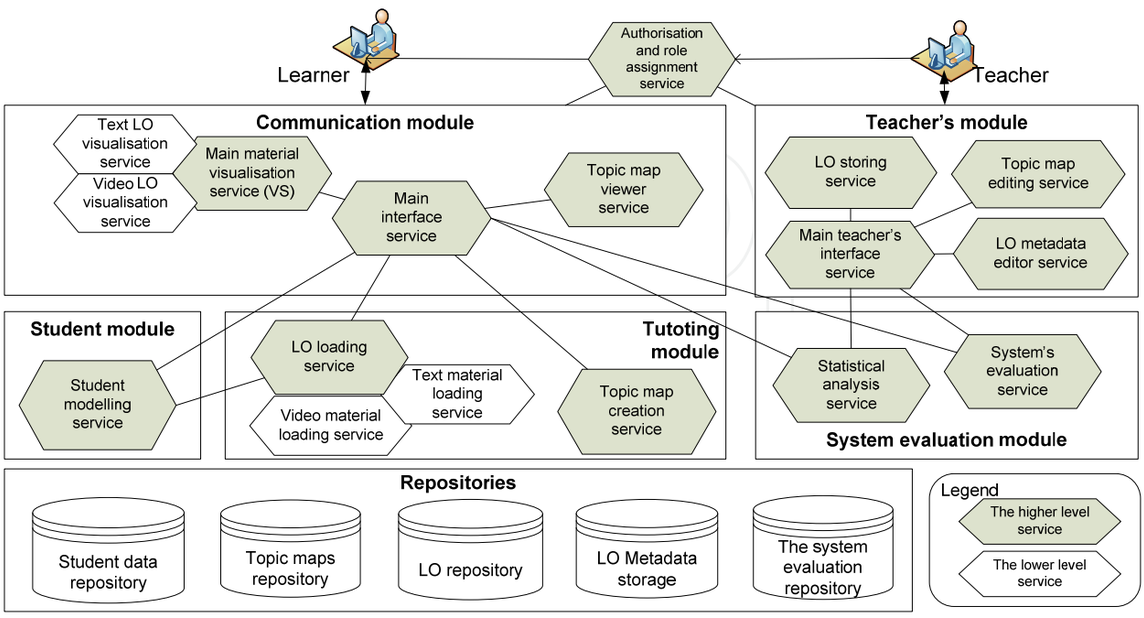


Fig. 7. The actual architecture of the implemented ITS prototype.

LOs used in the system – text and video LOs. The student module just has to maintain static student's profiles. Thus it contains just one service named student modelling service. The tutoring module is implemented using two higher level services, named topic map creation service (curriculum generation service) and LO loading service used to retrieve LOs. It has two lower level services corresponding to both types of LOs. The teacher's module besides the main interface service has services for editing topic maps, as well as for LO metadata editing and LO storing. Finally, the system has one additional module – system evaluation module, whose main task is to collect evaluations of the system provided by the learners.
The user interface of the system is given in Figure 8. The given screenshot is shown to the student when he/she chooses the topic to watch the video about. The menu is in the top left corner of the window (not included in the figure). The centre of the screen is occupied with the video player. One of the main parts of the system is shown in the bottom part of the figure. During the video session the system displays topics that are covered within lecture and related content that corresponds to the selected topic. Also, related topics, lectures and LOs are shown, allowing the learner to navigate to any related topic or material.

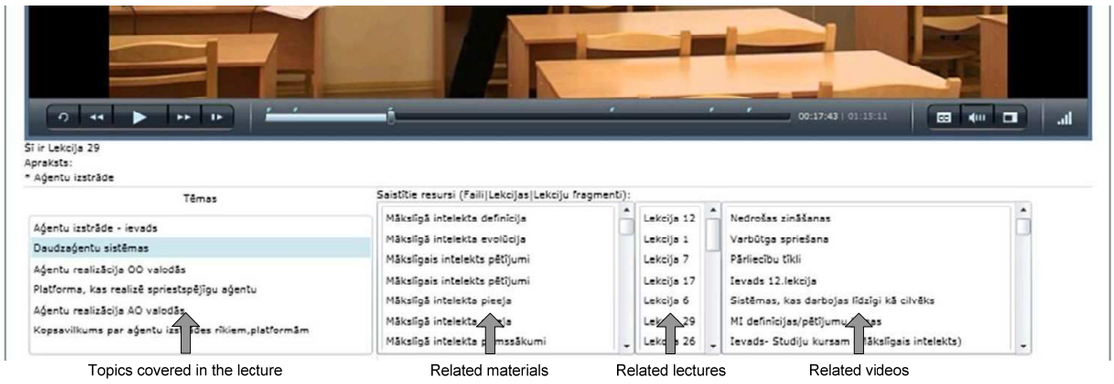


Fig. 8. The user interface of the developed ITS prototype.

The group of 19 students have approbated the developed ITS. They watched videos, red theoretical materials of lectures and explored examples. The feedback showed that granularity of LOs should vary, because some students like LOs with low granularity, while others prefer LOs with high granularity. The videos were used frequently and on average each student viewed 90% of available videos at least once. Another encouraging result is that 90% of students pointed out topic maps as appropriate tool to visualize the curriculum.

6. Hybrid intelligent tutoring system architecture

As concluded above, agents are more suitable to implement some components than services and vice versa for other components. To benefit from the advantages of both technologies hybrid architecture is proposed. It preserves the approach used in the previously described architectures in two senses. Firstly, each module from the traditional ITS architecture is implemented as a set of distributed components. Secondly, the architecture consists of two levels. Still the implementation differs from the previously described architectures. The higher level consists of agents that implement the logical or deliberative part of the system. The agents correspond to the higher level holons of the multi-agent architecture. The lower level of the architecture is mainly implemented as services that carry out simple reactive

behaviour upon request like retrieving data from repositories. Agents have the same functions as heads of the holons in the multi-agent architecture – they make all intelligent choices to carry out the tutoring process and use services for particular tasks. Services of the hybrid architecture are used instead of the body agents of the multi-agent architecture. It allows keeping the advantages of both previously described architectures, like openness to new functionality and high modularity. The proposed architecture is presented in the following subsections by specifying agents of each module as well as services used by these agents and, as a consequence, included in the corresponding modules.

6.1 The tutoring module

The tutoring module is responsible for four types of functions, namely, curriculum sequencing, material retrieval or generation, problem generation and providing feedback to the learner after he/she has finished the problem. As a consequence the module contains the following four agents: curriculum agent, teaching strategy agent, problem generation agent and feedback agent. *The curriculum agent* creates the curriculum of the course for each learner. The agent uses topic retrieval service to retrieve topics from the database. The *teaching strategy agent* is responsible for choosing the most suitable material for the learner in each topic from the materials available in the repository. To retrieve materials from the repository it uses services. There is one service for each type of materials used in the system. *The problem generation agent* is responsible for generation of problems that are suitable to the learner's characteristics. The agent gets the student model from the student modelling agent described below. It determines the most suitable type of the problem and calculates parameters like the difficulty level of the problem. The agent uses the problem generation service to generate the problem of the given type with given parameters. *The feedback agent* generates feedback for a learner each time he/she submits his/her solution of the problem. During the creation of the feedback the evaluation of the learner's solution provided by the knowledge evaluation agent is used. Components that implement the tutoring module are shown in Figure 9. Services are shown together with the agent that is using them.

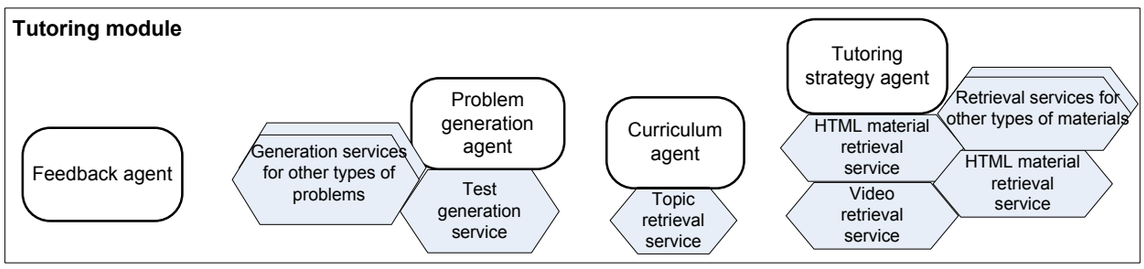


Fig. 9. Components of the tutoring module.

6.2 The communication module

The communication module contains single agent – the interface agent that is responsible for carrying out all interactions with the learner. Firstly, it perceives learner's actions and forwards them to agents of other modules that need information about corresponding actions. Secondly, it is responsible for presenting all information to the learner. It uses services of the module for this purpose. Separate services are created for each type of information given to the learner, namely, materials, problems, curriculum and feedback. To

make the architecture open, the additional lower levels of services are introduced in the communication module. The abovementioned services (for example, material visualisation service) are responsible for presentation of all types of the given information (materials). In case such service is implemented as a monolith component, it's code must be changed to support new type of materials. For, example, if there is a need to add an audio material the code of material visualisation service must be changed. Thus such services are implemented only as dispatchers that find the corresponding lower levels service that is capable to visualize the corresponding type of information. The visualisation job is physically done by the lower level service. It allows implementing visualisation of new information just by adding new lower level service, for example, by adding audio material visualisation service. The architecture of the module is given in the Figure 10.

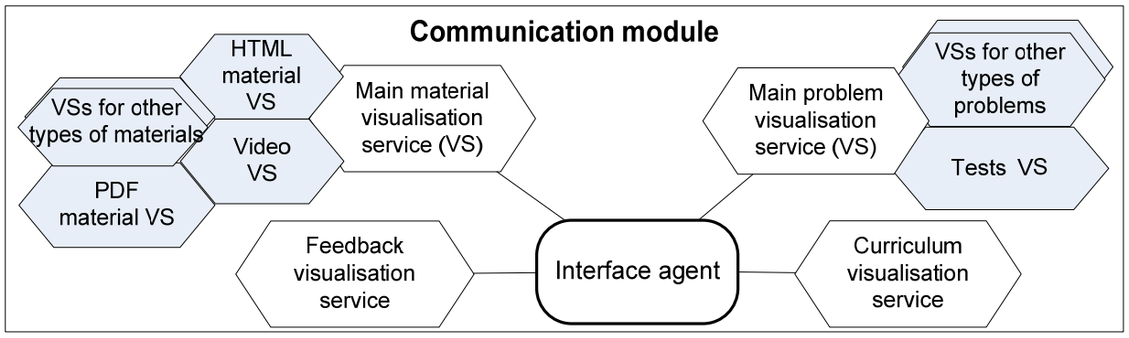


Fig. 10. The architecture of the communication module.

6.3 The expert module

The expert module is implemented identically as in the holonic multi-agent architecture. Both layers of the architecture are implemented as agents, because solving any problem given to the student is a complex task that usually requires some intelligent mechanisms like reasoning which are easier to implement inside agents not services. The higher level of the module's architecture contains only one agent – the expert agent. It receives all requests to solve problems. It finds the corresponding lower level agent and forwards the request to that agent. The lower level consists of agents corresponding to the types of problems used in the system. The architecture of the expert module is depicted in the Figure 11.

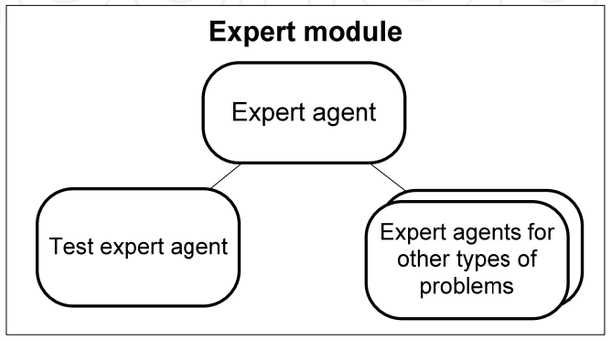


Fig. 11. The expert module's architecture.

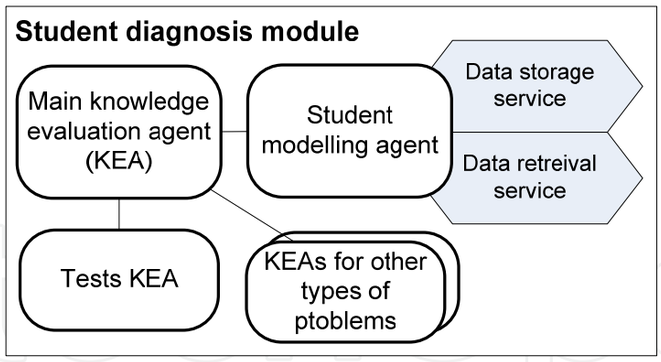


Fig. 12. The student module's architecture.

6.4 The student diagnosis module

The higher level of the student diagnosis module's architecture consists of two agents, namely, student modelling agent and main knowledge evaluation agent. *The student modelling agent* is responsible for creation of the student model and providing it to other agents upon request. The student modelling agent uses data storing and data retrieval services. Data storing service stores facts about the learner (for example, actions done by him/her) upon the request by the agent. The data retrieval service retrieves the stored facts for the agent. The student modelling agent receives all facts about the student, analyzes them and creates the student model. *The knowledge evaluation agent* is responsible for evaluating learner's knowledge in certain topic using the solution of some problem provided by him/her. The knowledge evaluation is done by comparing learner's solution to the correct solution provided by the expert agent. Comparison of two solutions and knowledge evaluation may require complex intelligent mechanisms. Thus, the components of the second level are implemented as agents instead of services. Each type of problems used in the system have corresponding lower level knowledge evaluation agent. The architecture of the student diagnosis module is shown in Figure 12.

6.5 The whole hybrid ITS architecture

The whole hybrid architecture is given in Figure 13 that shows components implementing all four modules. Additionally, interactions among components of different modules are shown. Only agents of the architecture's higher level interact to components of other modules. So, interfaces with other modules are implemented in one or a few agents of each module. The hybrid architecture is open in the similar sense to both above described architectures. Some kinds of new functionality can be added to the system by adding new lower level components and without changing existing components. The most common changes are additions of new types of materials (e.g., audio materials) and new types of problems (e.g., course specific tasks). New type of materials usually needs corresponding functionality like audio streaming to be implemented. New types of problems need the corresponding problem generation, problem solving and knowledge evaluation functionality. To enable easy addition of such functionality the architecture specifies open sets of lower level components corresponding to the functionality that varies from type to type of materials and problems. Open sets are denoted by double boxes in the Figure 13. New components may be added to these sets at any time.

Thus the hybrid architecture preserves the main strengths of the above described service oriented and multi-agent architectures, namely, high modularity and openness for new components. Moreover, it adds significant advantage by implementing each component in suitable technology. The logical layer with all intelligent mechanisms is realized by intelligent agents, while reactive components without any proactive or intelligent actions are realized using simpler technology – services. It removes the main drawbacks of both homogenous architectures. Neither simple components without any intelligent behaviour are implemented as agents, nor are the intelligent mechanisms developed from scratch in the services.

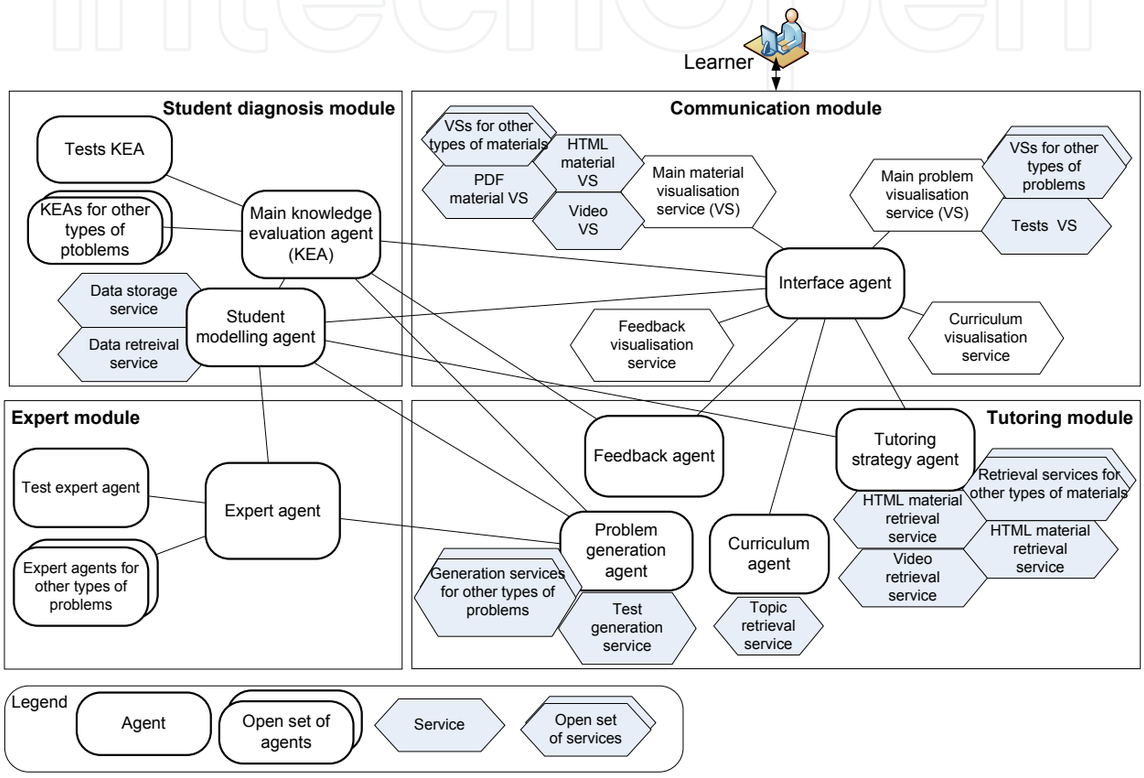


Fig. 13. The hybrid ITS architecture.

7. Conclusion

ITSs based on the traditional architecture have problems with modularity and as a consequence complex development and change implementation, as well as reuse of ITS components is almost impossible. Distributed technologies may be used to eliminate these drawbacks. Still, majority of the known agent based ITSs do not fully use advantages of distributed technologies. They are not open and do not enable reuse of small scale components. To solve these drawbacks three open architectures for ITS development are proposed. The main advantages of the proposed architectures are the following. All of the proposed architectures are open and consist of small-scale components that have one or very few tasks. The openness of the architectures allows creation of systems that are extendable with certain types of functionality by just adding new components and without changing existing code. Such option enables easy adaptation of the ITS's functionality to the

changes made in the course as well to any new courses. Usage of small scale components that have only one task means that these components can be reused in any system that needs this task. Additional advantages of the proposed architectures are increased modularity and decreased coupling and as a consequence complexity of interactions achieved by introduction of the organisational principles into the distributed technologies, namely multi-agent systems and service oriented architectures.

Two technologies analyzed in the chapter are similar because of their distributed nature. Therefore the architectures of systems using these technologies should be built using the same principles. Thus the lessons learned in usage of one technology can be used in building systems with another technology. The chapter shows how the lessons learned in agent based ITS development can be used in service oriented ITSs.

Agents and services have significant differences. While agents offer natural implementation of the intelligent mechanisms needed in the logical part of the system, services are easier to implement, and better known for developers. Services also support easy integration of various technologies, like various tools for the user interface of the system, video streaming tools, etc. These characteristics are important for implementation of the lower level components and the user interface. Thus agents are more suitable for some components (more deliberative components), while services are more suitable for other components (more reactive components). That is a reason why in this chapter the multi-agent and the service-oriented architectures are proposed to use as a basis to develop hybrid architecture implementing each component in the technology that fits the nature of the component. The main direction of the future work is to implement the proposed hybrid architecture and test how does it work in the practically implemented prototype of an ITS. Additionally, the possibilities to use hybrid architectures consisting of services and agents to other systems with some reactive and some deliberative components should be analysed.

8. References

Alpert, S.R. Singley, M.K. & Fairweather, P.G. (1999). Deploying Intelligent Tutors on the Web: An Architecture and an Example. *International Journal of Artificial Intelligence in Education, Vol. 10, No. 2,* pp.183-197.

Anohina, A. (2007) *Development of an intelligent supporting system for adaptive tutoring and knowledge assessment.* Doctoral Thesis. –Riga, RTU.

Bicans, J.; Lavendelis, E. & Vanags, M. (2011). Adaptive tutoring by means of student model based learning object selection. *Proceedings of the IADIS International Conference "e-Learning 2011", Rome, Italy, 20 - 23 July 2011, Volume 1*, pp. 251-257.

Capuano, N. et al. (2000). A Multi-Agent Architecture for Intelligent Tutoring. *Presented at the International Conference on Advances in Infrastructure for Electronic Business, Science, and Education on the Internet (SSGRR 2000)*, Rome, Italy.

de Antonio, et al. (2005) Intelligent Virtual Environments for Training: An Agent-based Approach. *Proceedings of 4th International Central and Eastern European Conference on Multi-Agent Systems*. Budapest, Hungary, September 15-17.

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*, L'Aquila, July 31 - August 06.

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

Dorça, F.A., Lopes, C.R., Fernández, M.A. (2003). A multiagent architecture for distance education systems. *Proceedings of the 3rd IEEE International Conference on Advanced Learning Technologies, ICALT'03*, p. 368.

Fischer, K., et al. (2003). Holonic Multiagent Systems: A Foundation for the Organisation of Multiagent Systems, *Lecture Notes in Computer Science 2744,* Springer.

Georgouli, K. et al. (2003). A Web Based Tutoring System for Compilers. *Proceedings of the 14th EAEEIE Annual Conference on Innovation in Education for Electrical and Information Engineering (EIE),* Gdansk, Poland, June 16-18.

Gerber, C.; Siekmann, J. & Vierke, G. (1999). Holonic multi-agent systems, *Technical Report R-99-03, DFKI GmbH.*

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* Birkhauser Publishing Ltd, pp. 141-157.

Koestler, A. (1967). *The Ghost in the Machine*. Hutchinson & Co, London.

Lavendelis, E. & 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, pp. 100-108.

Lavendelis, E. (2009). *Open multi-agent architecture and methodology for intelligent tutoring system development.* Doctoral Thesis. –Riga, RTU. 222 p.

Lavendelis, E. & Grundspenkis, J. (2010). MIPITS - An Agent based Intelligent Tutoring System. *Proceedings of 2nd International Conference on Agents and Artificial Intelligence (ICAART 2010) Vol. 2.,* Valensia, Spain, January, 22-24. pp. 5-13.

Lavendelis, E. and Bicans, J. (2011). Multi-Agent and Service Oriented Architectures for Intelligent Tutoring System Development. *Scientific Journal of Riga Technical University 2011, Series 5, Volume 43*, pp. 27-36.

Lavendelis, E. & Grundspenkis, J. (2011). MASITS Methodology Supported Development of Agent Based Intelligent Tutoring System MIPITS. *Communications in Computer and Information Science. - 129 (3).,* pp 119-132.

Luger, G.F. (2005). *Artificial Intelligence: Structures and Strategies for Complex Problem Solving*, Addison-Wesley, Harlow, England, 903 p.

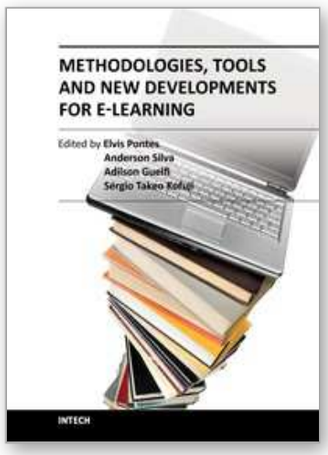
Nimis, J. & Stockheim, T. (2004). The Agent.Enterprise Multi-Multi-agent System. *Proceedings of the Conference on Agent Technology in Business Applications, Essen, Germany.*

Silveira, R.A. & Vicari, R.M. (2002). Developing Distributed Intelligent Learning Environment with JADE – Java Agents for Distance Education Framework. *ITS 2002, Cerri, S.A., Gouardères, G., Paraguaçu, F. (Eds.), LNCS 2363,* pp. 105–118.

Triantis T. & Pintelas P. (2004). An Integrated Environment for Building Distributed Multi-agent Educational Applications". *AIMSA 2004, LNAI 3192,* pp. 351–360.

Vilkelis, M.; Lukashenko, R. & Anohina, A. (2009). Technical Evolution of the Concept Map Based Intelligent Knowledge Assessment System. *Proceedings of the Workshop on Intelligent Educational Systems and Technology-Enhanced Learning,* September 7, Riga, Latvia, pp. 214-221.

Webber, C. & Pesty, S. (2002). A two-level multi-agent architecture for a distance learning environment. *ITS 2002/Workshop on Architectures and Methodologies for Building Agent-based Learning Environments, E. de Barros Costa*, pp. 26-38.



Methodologies, Tools and New Developments for E-Learning

Edited by Dr. Elvis Pontes

ISBN 978-953-51-0029-4
Hard cover, 332 pages
Publisher InTech
Published online 03, February, 2012
Published in print edition February, 2012

With the resources provided by communication technologies, E-learning has been employed in multiple universities, as well as in wide range of training centers and schools. This book presents a structured collection of chapters, dealing with the subject and stressing the importance of E-learning. It shows the evolution of E-learning, with discussion about tools, methodologies, improvements and new possibilities for long-distance learning. The book is divided into three sections and their respective chapters refer to three macro areas. The first section of the book covers methodologies and tools applied for E-learning, considering collaborative methodologies and specific environments. The second section is about E-learning assessment, highlighting studies about E-learning features and evaluations for different methodologies. The last section deals with the new developments in E-learning, emphasizing subjects like knowledge building in virtual environments, new proposals for architectures in tutoring systems, and case studies.

How to reference

In order to correctly reference this scholarly work, feel free to copy and paste the following:

Egons Lavendelis (2012). Distributed Intelligent Tutoring System Architectures, Methodologies, Tools and New Developments for E-Learning, Dr. Elvis Pontes (Ed.), ISBN: 978-953-51-0029-4, InTech, Available from: http://www.intechopen.com/books/methodologies-tools-and-new-developments-for-e-learning/distributed-intelligent-tutoring-system-architectures

INTECH
open science | open minds

InTech Europe
University Campus STeP Ri
Slavka Krautzeka 83/A
51000 Rijeka, Croatia
Phone: +385 (51) 770 447
Fax: +385 (51) 686 166
www.intechopen.com

InTech China
Unit 405, Office Block, Hotel Equatorial Shanghai
No.65, Yan An Road (West), Shanghai, 200040, China
中国上海市延安西路65号上海国际贵都大饭店办公楼405单元
Phone: +86-21-62489820
Fax: +86-21-62489821

IntechOpen