Pedagogical Agents for Teacher Intervention in Educational Robotics Classes: Implementation Issues

Ilkka Jormanainen [1], Yuejun Zhang [2], Kinshuk [3], Erkki Sutinen [1]
[1] *Department of Computer Science and Statistics, University of Joensuu, Finland*
[2] *Department of Information Systems, Massey University, New Zealand*
[3] *School of Computing and Information Systems, Athabasca University, Canada*
ilkka.jormanainen@cs.joensuu.fi

Abstract

Teachers working in robotics classes face a major problem: how to keep track on individual students' or even small groups' progress in a class of 30-40 students. A multi-agent environment to help teachers with this problem is based on having pedagogical agents to monitor students' interaction, robots' movements, and the construction and programming process of robots. Mobile interaction agents move in local area network delivering other agents' observations to teacher's visualization agent. The implemented prototype system has revealed the strengths and the weakness of the proposed architecture.

1. Introduction

Educational robotics is successfully used for teaching in several school contexts. These small-scale computerized teaching tools have many advantages over the PC based tools which have traditionally been used to teach, for example, programming or engineering. Lego Mindstorms is one well-known example of educational robotics. This robot construction set is flexible and simple enough platform for building and programming even for novices.

Educational robotics sets let room for student's own creativity by emphasizing active learner as the center of the learning process. However, in a typical classroom setting, especially at the elementary level, a teacher might have 30-40 children to teach. When using educational robotics in large classroom settings, students are usually divided into groups of 3-4 students. A typical educational robotics project follows an iterative cycle of building, programming, testing, and evaluation. It is characteristics that groups proceed differently, being in different phases of the cycle at the same time. This causes difficulty for the teacher to notice the needs for intervention. Our approach is to use educational agents to help the teacher to focus his/her attention in potential problems. The problem can be generalized as follows [4]: *How could the robotics environment inform the teacher what students are doing and how they are progressing?* In this article, we describe a prototype implementation of educational robotics environment which aims to support the teacher to focus his/her attention in potential problems in the classroom. The implementation is based on the concept described by Jormanainen et. al [3,4].

The prototype system contains four separate parts. Each of these parts implements independent, agent-like behavior. The first agent module is an intelligent agent which inhabit in the IPPE programming environment (*the IPPE Agent*). The main purpose of this agent is to observe the user's activity with the programming environment and build decisions based on the input data coming from the graphical user interface of the programming environment. The second agent module implements the similar observing behavior than the IPPE Agent, but it inhabits in Lego robots' RCX unit (*the LM Agent*). Due to the different host platform, we describe these agents as s separate parts of the model. Third agent is a screen-based model of classroom setting constructed with the tools provided by Empirical Modelling environment. The model works as a *visualization agent* in our system. Finally, we have implemented *an interaction agent*, which has the ability to move from one computer to another, for example from a students' computer to the teacher's computer to report a learner's problem observed by the other agents.

In this article, we discuss the implementation issues of the different types of agents. Technical aspects of

the LM and IPPE Agents will be described more deeply whereas visualization agent and interaction agent will be described in more abstract level. The LM and IPPE Agents implement the core functionality of the system, and therefore in this paper, we focus on to a deeply description of these agents. The article is organized as follows. Chapter 2 describes the existing work in the area and sets the work to the right context. Chapter 3 introduces the overall architecture and tools we have used to realize the prototype of the system. Chapter 4 discusses the actual implementation of the system by describing the essential parts of if. Furthermore, chapter 4 discusses also some practical implementation issues we have faced during the project. Chapter 5 concludes the paper and draws some directions for the future work.

2. Background and related work

Jormanainen et. al [3,4] introduce an agent-based architecture for the educational robotics environment that helps the teacher to intervene into small groups' work, based on the observed student-to-student interaction, robots' movements, and the group's progress in constructing and programming their robots. Unlike most current pedagogical agents which are usually applied to computerized learning environments or simulated virtual realities, the agency technologies described in the previous work of authors and in this paper, are used in a traditional classroom, a real physical world, in which students are working on a Lego project. The physical locations and activities of all objects, especially interactions among group members, are complicated and hard to model. Automating all the agency processes is difficult, if not impossible. Currently, we have decided to focus on two student activities: programming and manipulating the robot.

2.1 Agents and mobile agents in education

Software agents have long been applied in educational environments to provide learning support. Educational agents are a class of agents that assists a user in an education-related task. Agents can monitor progress, give instruction when needed, help organize students' work, and provide feedback for tutors. However, there is no agreement in the literature on a unified definition for the term "agent". People use this term slightly differently according to the context where agents are employed. Agents exhibit some properties, such as autonomy, social ability, responsiveness, and proactiveness, upon which a common consensus has been reached [8]. Agents possessing mobility are called mobile agents. Mobile agents are usually software programs, which may be dispatched from one computer and transported to a remote computer for execution. The motivation for using mobile agents stems from a number of potential benefits, such as efficiency and reduction in network traffic, asynchronous autonomous interaction, interaction with real-time entities, local processing of data, and support for heterogeneous environments [5].

2.2 Related work

The idea of using agency technology to monitor learning progress and providing learner help in a robotics classroom setting is not new. Previous work can be found in [1], where a multi-agent framework for distance support via the Internet in educational robotics is described. In this framework, the teacher can obtain information about the learners' work and can be informed automatically by the system when learners encounter difficulties.

George and Despres [1] summarize the advantages offered by the agency approach in educational robotics: First, the system and its functionalities can easily be distributed between tutor's workstation and learners' workstations; secondly, the modular aspect of multi-agent system ensures easy modification of an isolated agent's behaviour, and finally, a multi-agent system provides an open architecture which allows for the integration of new agents if required. We follow this argument, believing that agency approach is the right choice for monitoring pedagogical activities in the robotics classroom. Based on the previous work done by other researchers, we further develop an agency system by not only focusing on physical manipulations of the robot, but also controlling learners' programming activities and using mobile agents for asynchronous interactions as well as Empirical Modelling for visualization of observations.

3. Design in conceptual level

One of the key elements with the design of our agent architecture is the usage of existing software and hardware tools. The main idea is to embed agents to the existing applications in a way that they could work independently, without user's intervention. The agents work also in a physical world, which set certain restrictions and demands for example to inter-agent communication. Thus, a strong and reliable communication protocol is an essential design issue. The aim of the agents is to collect data and analyze the

activities in the class room. It is crucial to note that the system just indicates the progress of the students; it does not assess the results of the learning process. The teacher has to make his/her own decisions concerning the learning outcomes, and the system intends to help the teacher in this task.

Next in this chapter, we will introduce an overall concept of the architecture. The description shows how the agent architecture adds new functionality to the existing technical setting used in the educational robotics classes. After that, we will introduce the software and hardware tools we have selected to use to realize the architecture.

3.1 Concept of the framework

The framework for this agent application is based on to the existing tools. IPPE Agent works with the IPPE programming environment whereas LM Agent's host is RCX unit of Lego Mindstorms robotics kit. Figure 1 shows the general concept of the system. According to the concept presented in [3] and [4], the agents work in a learning setting gathering data and building reactions based on the data. Interaction agents deliver these reactions and other messages to teacher through the network. Teacher has a visualization agent to illustrate the activities in the class room, and a graphical interface to adjust the agents' behavior according to the current needs. Figure 1 shows with the dotted shapes the components we have implemented in this project. Next, we will introduce the existing tools, Lego Mindstorms robots, the IPPE programming environment, and the Empirical Modelling environment.

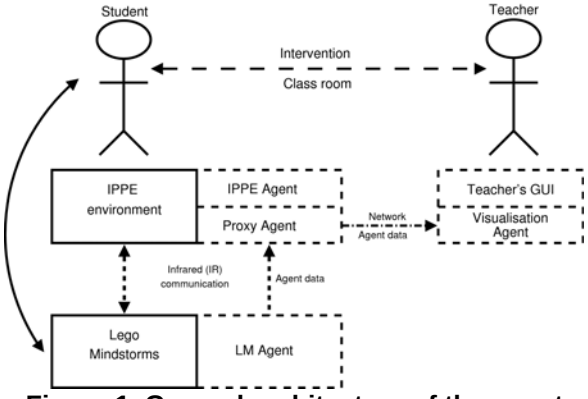

Figure 1. General architecture of the agent environment [4]

3.2 Lego Mindstorms

In the past few decades, researchers and industries have developed a number of different educational robot kits designed to help learning in scientific fields such as mathematics, physics, engineering, and computer science. These kits typically contain all the components which are needed to construct an autonomous robot: a small computer unit, motors, sensors, wheels, gearwheels, and belts. Autonomous robots can communicate and move independently according to the program the user has constructed. A well-known example of these tools is the Lego Mindstorms robot kit. The history of Lego Mindstorms goes back to 1986 when a research group supervised by Seymour Papert and Mitchel Resnick started to develop the Programmable Brick, a small unit capable of connecting to the external world through a variety of sensors and actuators [6]. The Mindstorms kit includes an RCX unit (Robotic Command Explorer), an independent computer, which is the core of the kit. An advantage of the RCX unit is its flexibility. The unit can be programmed by using a variety of programming languages such as NQC, Java or Visual Basic. The uses of the Lego Mindstorms kit in different contexts have been widely reported.

3.3 The IPPE programming environment

One of the key elements of educational robotics projects is programming. The robots we have used, Lego Mindstorms, can be programmed with a variety of programming languages. As an example, we have used the IPPE programming tool as a programming environment for robots. IPPE (Instructive Portable Programming Environment) is a tool for programming Lego Mindstorms robotics with a pseudo-like programming language near to the student's own natural language [2]. The environment has been developed at the University Joensuu since 2001. Programming can be done by composing the student's program with a graphical user interface, or by writing the code straight to the program editor window (Figure 2). The modular structure of the IPPE environment allows developers to write new features to support learning and teaching, such as intelligent agents described in this paper.

When programming Lego robots with the IPPE environment, the programmer first creates a set of commands to use to construct the program. The commands can be used to control different actuators and sensors in the robot as well as to create a logical structure for the program with loops and conditions. The commands are placed to a sandbox (Figure 2, the column in the middle of the IPPE GUI). After the user has constructed a large enough set of the commands to

the sandbox, he/she can start to construct the program. The IPPE environment automatically intends the code and highlights the syntax with colors (Figure 2, the rightmost column of the IPPE GUI).

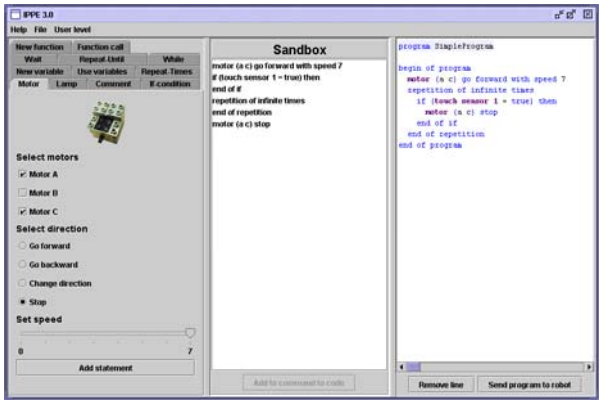

Figure 2. The IPPE environment

3.4 The Empirical Modelling tools

An essential part of our system is a visualization agent for teacher's use. We use *Empirical Modelling* (EM) environment to implement this agent. The main reason to select EM as an implementation approach for teacher's agent rises from the nature of EM. The EM approach supports well a cyclic process and user's own observations about the phenomena. This means that teacher can update the model according his/her observations and, on the other hand, the teacher gets real-time feedback from the agents running in the environment. The Empirical Modelling is an approach for constructing computer based models that can assist in the understanding of a phenomenon. The modeling approach has been developed at the University of Warwick since 1980's. The approach has an emphasis on experiment, observation and interaction during the development process [7] and it offers an alternative approach to the goal of developing interactive artifacts to support experiential learning. Empirical Modelling describes the characteristics and features of a construal (model) with three key concepts: observables, dependencies and agents. This description is made by using a set of definitions, definitive scripts, and in this way, representing state-as-experience. The process of constructing computer-based artifacts using definitive scripts is called definitive programming. Interaction with EM models takes place through a continuously process of typing definitions on-the-fly. The modeling environment maintains automatically relations and dependencies between the different parts of the model. The definition for a relationship between two observables is of the form `identifier is expression`, for example `weight is 2*width`. If the value in the right side of the definition (`width`) is updated, the value of a dependant observable (`weight`) is updated automatically. The approach has many similarities with a spreadsheet, where dependencies between cells are recorded by using definitions of the cells' content and their relations.The most used tool for building models with the EM approach is an EDEN (Engine for DEfinitive Notations) interpreter called *tkeden*. The modeling process with core EDEN notation can be enhanced with different extensions. For example, DoNaLD and Sasami are definitive notations which allow usage of 2D and 3D graphics with EM models. In this work, we have used EDEN and DoNaLD notations to build the model of the classroom setting and visualization agent.

4. Implementation

The implementation of the architecture has been done mostly with Java-based tools because Java supports agent's autonomy by providing threaded applications, event-processing mechanism and networking facilities. Furthermore, the IPPE programming environment and Lego RCX unit's programming platform LeJOS are Java-based tools. Besides Java, the Empirical Modelling tools have been used for the implementation of visualization agent and teacher's GUI. However, due to the fact that EM tools have limited capabilities to communicate with the other applications, we have implemented also a communication module as a part of teacher agent. This part will be implemented as a Java application to make the communication with other agents as easy as possible. Figure 3 presents the general architecture and how different parts of the application discuss with each others. In this chapter, we will describe the implementation of different agent modules more deeply. We cover the LM and IPPE Agents on the level of technical description, whereas the interaction and visualization agents have been described on more abstract level. Finally, we will discuss some general implementation issues and experiences we have got during the implementation process.

4.1 LM Agent

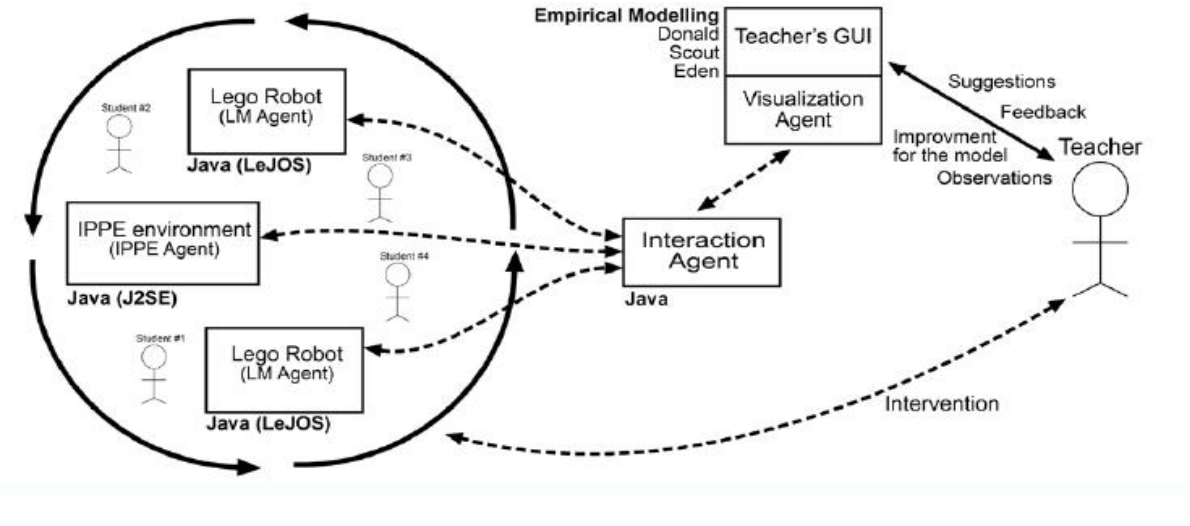

Figure 3. The general structure of implementation

The LM agent monitors students' interaction with the physical robot. It observes the manipulation of the students on the robot set by collecting data from motors, sensors, and buttons. Its main functionalities include monitoring the idle time for manipulation of the robot, detecting motor movements, and checking sensor signals. This agent resides in the RCX unit and sends data via the IR tower to an interaction agent running on the student computer. Ultimately, the interaction agent delivers the agent data to the teacher agent and the Empirical Modeling process. The LM agent is implemented as a Java *thread*. The execution of the thread is started whenever the user switches on the RCX unit. The agent runs in the thread independently, gathering data from the RCX unit and sending information to the interaction agent based on these data. The implementation of the agency is transparent. This means that the user does not notice the agent running in the robotics system. The agent program is included when the LeJOS firmware is downloaded into RCX.

Two vital issues in the implementation of the LM agent exist: the limited computing capability of the RCX unit and the unreliable communication facilities between RCX and the host computer. Therefore, a requirement arises that the functionalities of the LM agent and the data processing strategies must be carefully scheduled. Considering the limited computation ability and memory capacity of the RCX unit, it is quite reasonable to reduce as far as possible the number of tasks for the LM agent by, for example, letting it just simply monitor and report occurrences such as sensor events and button clicks, while leaving further computations to other applications running on the host computer. However, this implementation will significantly increase the communication overhead between the RCX unit and the host computer, as every event happening to the RCX unit forces a transportation of data from the LM agent to the host computer. The communication protocol for this data transportation, as we discussed above, is not so reliable. We are looking for the new technical solutions to overcome this problem (for more information, see Section 4.5).

In the mean time, communication between the RCX unit and the host computer is a key to the implementation of the LM agent. Only by this communication can information collected by the LM agent be delivered to the interaction agent, and ultimately to the EM process. Java programs use streams to communicate. Particularly LeJOS uses standard Java IO Streams to send primitive Java data types from the PC to the RCX or from the RCX to the PC through the IR tower. Both ends can initiate communication, and data can be sent in both directions. LeJOS provides the `josx.rcxcomm` package for communication between a PC and the RCX. In our implementation of the LM agent, the focus is put on the data transfer from the RCX to the host computer. Whenever a button click, a sensor event, or a change in motor status happens, the LM agent should be able to detect the occurrence and notify the interaction agent of this occurrence via the communication protocol.

The RCX port is the physical infrared port on the RCX unit. The `RCXPort` class included in the LeJOS `josx.rcxcomm` package hides a lot of the low-level details of the communications from the user. It provides an `OutputStream` and an `InputStream` that can be made into a `DataOutputStream` and a

DataInputStream respectively. In our implementation, the LM agent running in the RCX unit writes data to the DataOutputStream when an event such as a button click occurs, while the interaction agent running on the host computer reads this data from the DataInputStream to know what has happened to the robot. Figure 4 illustrates how the program code looks like for the LM agent to write data to the RCX port when the PRGM button has been pressed. The code for the interaction agent to read this data is very similar to the Figure 4.

```
import josx.rcxcomm.*;
public class SensorEventRCX {
  Button.PRGM.addButtonListener(new
    ButtonListener() {
      public void buttonPressed(Button b) {
          sendEvent(b);
      }
  });

  public void sendEvent(Object source) {
    int sentValue = source.getId();
    RCXPort port = null;
    try {
      port = new RCXPort();
      DataOutputStream out = new
        DataOutputStream(
        port.getOutputStream());
      out.writeShort(sentValue);
      out.flush();
    } catch (IOException ioE) {
      LCD.showNumber(1111);
    } finally {
      port.close();
    }
  }
}
```

Figure 4. Code fragment from the LM agent

4.2 IPPE Agent

The aim of the IPPE agent is to observe the user's interaction with the programming environment. The agent intends to detect the possible problems that the user might have with programming. One instance of the IPPE programming environment running in a workstation can have several agent instances to observe the different activities. The individual agent instances build their decisions according to the input data, which the programming environment passes to the agent maintainer. The agent structure has been implemented as a Java package called IPPEAgent. This package has a connection the IPPE programming environment through the event listener mechanism provided by the Java programming platform. The IPPEAgent package contains the following base classes: AgentMaintainer, Agent, AgentRule, and AgentParser (Figure 5).

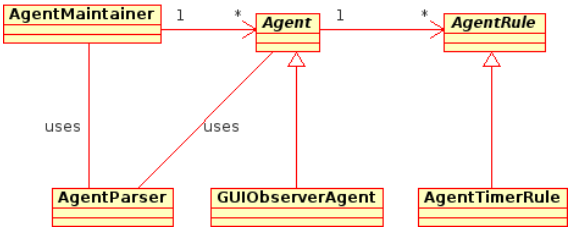

Figure 5. Core classes and implemented examples

The aim of AgentMaintainer class is to maintain a set of individual agents. The maintainer instance creates and destroys the agents according to the definitions available at the particular moment of time. The maintainer also passes the events to agents it holds in a collection. AgentMaintainer is a subclass of java.lang.Thread, and it can be run independently as a background process to observe the user's activity transparently (that is, without user's intervention). When the IPPE environment is launched, also an instance of AgentMaintainer class is created and the execution of the thread is started. Next, the agent maintainer registers itself as a listener to the graphical user interface components. In this way, the Java Virtual Machine (JVM) passes to the agent maintainer the same events than it passes to the IPPE programming environment.

Agent class is an abstact class, which provides a basic functionality for the specific agent. This functionality includes initializing the agent instance, accepting or rejecting the events coming from the agent maintainer, maintaining the definitions of the agent instance, and communication with other agents. To complete the functionality according to the particular needs, the Agent class has to be derived to a sub-class which contains the actual implementation of the agency (making the decisions based on the data available). As an example, a GUIObserverAgent class has been created. The aim of this agent is to observe the GUI events coming from the agent maintainer. The agent can react, for example, if the delay in programming exceeds the given limit, or if the user's program code in the IPPE environment does not contain the essential elements, such as event loop. These definitions can be modified by the teacher during the execution of the agent through the visualization agent and teacher's interface (see Chapter 4.3).

AgentRule is a class which defines the rules for an individual agent. The rule has a boolean state (true/false), which tells whether the agent has met the condition defined by the rule. The agent can hold several of these rules, and the rules can be connected

with logical operator 'AND' and 'OR'. Operator AND means that if all of the agent's rules are in state TRUE, the agent should trigger a pre-defined event, which can be for example sending a message to the visualization agent or to the other agent running in the system. Also AgentRule can be derived to achieve more specific functionality. As a trivial example, AgentTimerRule has been implemented.

AgentParser is a static class which takes care of parsing the agent definitions. The methods of AgentParser class are called from the instance of AgentMaintainer class. The parser has methods to parse definitions of agents and definitions of rules, and these methods returns and instance of agent or rule respectively. The agent definitions are saved to the local file system by the interaction agents. The definition file has the syntax presented in Figure 6, where ID defines a unique identification string for the agent. TYPE element defines what kind of agent the particular definition defines. COMPONENT element defines a list of components that the agent should listen. This list is used to filter the events which the agent receives from the agent maintainer. RULE component defines the logical rules for agent's reaction. EVENT element describes the action which agent should do when the conditions defined by the RULE component are met.

```
# Agent's ID
ID:EXAMPLE_AGENT
# Agent's type
TYPE:GUI_EVENT_OBSERVER
# List of components that the agent listens
COMPONENTS:Comp1,Comp2
# Agent should fire the event if...
RULE:DELAY(Comp1) > 10 AND DELAY(Comp2) > 8
# What to do if event is fired
EVENT:SEND_MESSAGE(message)
```

Figure 6. Agent definition file

We ended up to the architecture described above to minimize the amount of modifications that were needed for the existing code base of the IPPE programming environment. Furthermore, with this solution, we were able to achieve greater degree of independency for the agents. With the solution based on threads, the agents work independently, regardless the state of the IPPE environment.

4.3 Interaction and Visualization Agents

The interaction agent is responsible for coordination among agents in general, and data transfer in particular. It is an independent Java program, which receives data from the *sender agent*, then moves towards the node the *receiver agent* resides and delivers the data locally to the receiver agent. When student activities are reported to the teacher, the sender agent is the LM or IPPE Agent on a student computer, while the receiver agent is the visualization agent on the teacher computer. When a new agent definition is generated and sent to the agent maintainer running at the student's computer, the situation is opposite. Each student computer as well as the teacher computer has an instance of the interaction agent running on it.

The visualization agent in the teacher's computer presents agent data to the teacher. In this way, the system assists the teacher to achieve a better understanding about the progress of the students. The agent contains visualization engine which maintain the classroom model built with the Empirical Modelling tools. Due to the fact that EM tools have limited capabilities to communicate with the other applications (especially implemented with other programming languages), we have implemented also a communication module as a part of visualization agent. This part has been implemented as a Java application in order to achieve a fully advantage of Java programming language's capabilities to communicate with other applications especially over the network. This communication module and the EM model communicate through a local file system. The communication module passes agents' observations to the model as well teacher's definitions to the agents. With the visualization agent, the teacher (or the technical and pedagogical designers in general) can for example create new agents with a pre-defined EDEN command create_agent, which takes as parameters the agent definitions presented in the Figure 7. Based on this command, the EDEN interpreter creates a new agent definition file and writes it to the local file system. The communication module of visualization agent observers the file system and if a new definition exists, it passes this definition to the interaction agent. The interaction agent delivers this definition over the network to the agent maintainers running in the students' computers. In similar way, the EDEN command destroy_agent destroys the agents matching to the parameter that run in the system.

4.5 Implementation issues

The first working prototype of the agent environment described in this paper has been developed. We have faced several issues with the implementation, mostly due to the fact that agents work also in a physical world. First, it is necessary to build a framework which allows a ***reliable***

communication between agents and other parts of the system. Lego Mindstorms robotics kit uses infrared light (IR) for robots' communication. However, IR communication is very sensitive to the surrounding world, for example changing light conditions may affect the message passing. Also, the obstacles between the IR transmitter and receiver can cause messages to get lost. This means that a strong communication protocol is needed to make sure that the messages containing agent data are passed successfully from the robots to the interaction agent. A partial solution for this is to allow the LM Agent to communicate with all IR receivers in the classroom when sending agent data. In this way, the LM Agent could send data to the teacher through any free Interaction Agent available. The new version of the Lego construction kit, Lego Mindstorms NXT, will have Bluetooth communication facilities, which helps us to overcome this particular problem.

Another issue with the framework is the fact that the RCX unit has a ***limited capacity of computing***. Especially the amount of memory is small compared to the modern computers in which agents normally run. This sets limitations for the data which the LM agent collects and analyzes. Also, the thread implementation of the LeJOS platform is less reliable than for example that of the Java Virtual Machine by Sun Microsystem. Thus, a thread implementation of the LM Agent must be planned carefully to avoid any deadlock situation between the agent and the main thread in which the user's program runs. Again, we are looking forward the new RCX NXT since it facilitates a remarkable larger amount of memory than the current RCX 2.0. Generalization of the agents has been also an issue with the implementation. The LM and IPPE Agents have a similar functionality, but they can not be implemented in the similar way due to the limitations of the RCX unit. Thread implementation and data storing must be planned carefully especially for the LM Agent. The agents' internal functionality must be carefully planned to endure the fragmented data which sensors in the RCX unit can occasionally produce.

5. Conclusions

Educational robotics provides ideal tools for mobile and ubiquitous learning environments. However, teachers have to adopt new kind of teaching methods and traditional classroom situations do not fit very well to educational robotics settings which emphasis group-oriented learning and problem-based methods. Teacher may face difficulties to notice the needs for intervention.

In this paper, we have presented implementation issues of agent-based educational robotics environment. The implementation is based on the description presented by Jormanainen et. al [3,4]. The implementation of the system contains two major tracks: pedagogical agents for observing the learning environment and a visualization agent for the teacher to observe the processes in the classroom. Proposed multi-agent architecture includes altogether four separate agent modules. We use the Lego Mindstorms educational robotics set including a small-scale computer (RCX unit) and the IPPE programming environment as host platforms for the agents. For implementation of agents, we use Empirical Modelling environment and Java programming language.

The implemented prototype system has revealed the strengths and the weakness of the proposed architecture, and it has guided us towards the next version of the prototype which can be tested in real educational robotics classes during the first quarter of 2007.

6. References

[1] George, S. and Despres, C. (1999). A multi-agent system for distance support in educational robotics. *Proceedings of the International Conference on Telecommunication for Education and Training*, 344-353.

[2] Jormanainen, I., Kannusmäki, O., & Sutinen, E. (2002). IPPE - How to Visualize Programming with Robots. In M. Ben-Ari (Ed.), *Second Program Visualization Workshop* (pp. 69-73). University of Aarhus, Department of Computer Science.

[3] Jormanainen, I., Moroni, C., Zhang, Y., Kinshuk, & Sutinen, E. (2006). Implementation of Intelligent Agents with Mobility in Educational Robotics Settings. In *The 4th IEEE International Workshop on Wireless, Mobile and Ubiquitous Technologies in Education (WMUTE 2006)*. To appear.

[4] Jormanainen, I., Zhang, Y., Sutinen, E., & Kinshuk. (2006, July). Agency Architecture for Teacher Intervention in Robotics Classes. In Kinshuk, R. Koper, P. Kommers, P. Kirschner, D. G. Sampson, & W. Didderen (Ed.), *The 6th IEEE International Conference on Advanced Learning Technologies (ICALT 2006)* (pp. 142-143). Los Alamitos, CA: IEEE Computer Society.

[5] Kinshuk, Hong, H., Patel, A. (2002). Adaptivity through the Use of Mobile Agents in Web-based Student Modelling. *International Journal of E-Learning*, 1(3), 55-64.

[6] Laverdae, D. (Ed.). (2001). *Programming Lego Mindstorms with Java*. Rockland, MA, USA: Syngress Publishing.

[7] Roe, C., 2003. Computers for Learning: An Empirical Modelling perspective. Ph.D. thesis, Department of Computer Science, University of Warwick, UK.

[8] Wooldridge, M. and Jennings, N. R. (1995). Intelligent agents: theory and practice. *The Knowledge Engineering Review*, 10(2), 115-152.