An Agent-based Intelligent Tutoring System for Nurse Education

M. Hospers, E. Kroezen, A. Nijholt, R. op den Akker & D. Heylen

Department of Computer Science
University of Twente, Enschede, The Netherlands
anijholt@cs.utwente.nl

Abstract

This report describes the development of a teaching environment that uses agents to support learning. An Intelligent Tutoring System will be described, that guides students during learning. This system is meant for nurse education in the first place, but it is generic in the sense that the core is separated from the exercise modules and user interfaces. This means that the system can also be used for other (non-nursing) exercises. Exercises can be provided to the system in the form of XML data-files. A user interface can be text-based or 2D, but it can also be a 3D virtual reality environment. An application of the teaching environment for nurse training is described.

Keywords: educational environment, agents, health care, user interfaces

1 Introduction

This report describes the development of a teaching environment that uses agents to support learning. A software system called Ines will be described, that guides students during learning. This means that students can use the system to train certain tasks. Before the Ines project started, the idea was to make an Intelligent Tutoring System for nursing students. The reason was the lack of time, material and room at nursing schools. Because of these lacks, nursing students have too little possibilities to do practical exercises during their study, and they still have to learn a lot when they start working in a hospital. The intention of this project is to improve this situation. Hence the name: Intelligent Nursing Education Software.

The main goal of the Ines project is to provide an effective teaching environment. For an environment to be effective as a teaching environment, it has to fulfill two conditions. The first condition that has to be satisfied is that the teaching environment provides a correct and logical guidance system for the student. This means that a teaching system must be able to provide sensible feedback, demonstrations and/or explanations.

The second condition is that the user interface of the system is as optimal and accessible as possible. For this purpose, one could for example use a simple 2D graphical interface, but the interface could also consist of a highly accurate 3D virtual environment. Until now, the Ines project has mainly concentrated on the first condition. Independent of the task trained, the system is capable of providing feedback, explanations and demonstrations. For this purpose, a number of agents, which together form the main part of Ines, are implemented. The agents observe the students actions and check whether they are performed correctly and in a correct order. In the future, the intention is to concentrate more on the second condition.

Ines is a generic teaching system in the sense that the core is separated from the exercise modules and user interfaces. The exercises can be provided to Ines in the form of data-files. The information from these files is used to check if the actions from students are right, and to provide explanations and demonstrations if necessary. An exercise will typically consist of subtasks with some partial ordering defined for them. The tags in the data-files are general, and can be filled in for repairing a punctured tire as easy as for a nursing task, for instance.

Adding a new teaching task to Ines also means adding a new interface. A new interface may be text-based or 2D, but it can also be a 3D virtual reality environment. Even haptic feedback devices and a head-mounted device can be used, without making changes to the Ines system itself.

The implemented system is applied to a nurse training task. To demonstrate how the system works, an exercise-file and an interface were made for the subcutaneous injection.

The organization of this paper is as follows. First, in section 2 some Intelligent Tutoring Systems that gave the inspiration for the design of Ines are described. Section 3 contains a description of the architecture of Ines. Then, the most important part of the system, the cognition, is described extensively in section 4. A description of the implemented exercise, the subcutaneous injection, is given in section 5. Section 6 handles about the methods that are used by Ines for planning the order of the subtasks. The paper concludes with a section about testing and evaluation of the system and a section about future work.

2 Existing ITS's

There are a lot of programs that already teach certain tasks to users. The most important systems that gave the inspiration for the design of Ines will be mentioned in this section.

The Parlevink group, researching language, knowledge and interaction at the University of Twente, combines research on human-computer interaction, dialog systems, autonomous agents and virtual worlds. The group has built the VMC (Virtual Music Centre), which is a replica of an existing music centre and serves as an environment to experiment with multi-modal interaction with agents or other visitors to this virtual world [1]. In this context, Jacob was built, an instruction agent that helps users to solve a mathematical puzzle in virtual reality [2].

Based on the experience with Jacob, the idea arose for creating ADRI (Artificial Didactic Recital Instructor), a system intended to aid people learning to play the piano [3]. A 3D-world with a virtual piano and visualization of notes is connected with a real synthesizer using Midi to interact with the user. Also a multi-agent platform is used within the system. Each agent has a special expertise and knowledge domain and can give information or act when it decides that it is appropriate.

Other instruction agents that inspired the design of Ines are Steve [4], [5], [6] and Adele [4], [7]. Steve is an animated pedagogical agent. It gives instruction in procedural tasks in an immersive virtual environment. To allow Steve to operate in a variety of domains, its architecture has a clear separation between domain-independent capabilities and domain-specific knowledge. Adele (Agent for Distance Learning Environments) is developed by the USC/Information Sciences Institute’s Center for Advanced Research in Technology for Education (CARTE). Adele runs in a student’s web browser, and is designed to be integrated into web-based electronic learning materials. Adele-based courses are currently being developed for continuing medical education in family medicine and graduate level geriatric dentistry.

Another important system that inspired the design of Ines is LAHYSTOTRAIN [8]. This is a training system for two types of minimally invasive surgery techniques. It combines a Virtual Reality Simulator, a Basic Training System that provides web based theoretical training, and an agent-based tutoring system, the Advanced Training System, oriented to supervise the execution of practical exercises. Training in LAHYSTOTRAIN is carried out in two temporal consecutive phases: acquisition of theoretical knowledge with the Basic Training System, and acquisition of practical skills with the Advanced Training System.

3 System Architecture

For Ines, architecture is proposed that is based on two architectural styles [9]. The first architectural style that is embedded into Ines is the ‘data abstraction and object-oriented organization’. This means that single components within the Ines system will be represented by objects that have their own attributes and methods. These objects will be implemented by using several classes within the Java programming language. Objects that can be identified within Ines are, for example, all physical objects in the teaching environment (needles, swaps, and so on), but also the agents that are used are represented by objects.

The other architectural style that is used is the ‘event based, implicit invocation’ architecture. This architecture is mainly used to implement the agents in the Ines system. Each agent is capable of transmitting messages to other agents on a one-to-one level or a broadcast level. In addition, each agent is capable of receiving messages by observing all messages that are sent to that agent or that are broadcasted to all agents. The agent can then decide what to do for each type of incoming message.

Figure 1 shows the main parts of the Ines system. The patient model and the items in the abstract user interface are especially for nursing exercises. For other exercises, another model and other items can be used. As can be seen from the figure, the architecture of Ines consists of four main parts: input from devices, a concrete user interface, an abstract user interface and the cognition of Ines. The Ines Interaction API[1] is used for the communication between the concrete user interface and the abstract user interface.

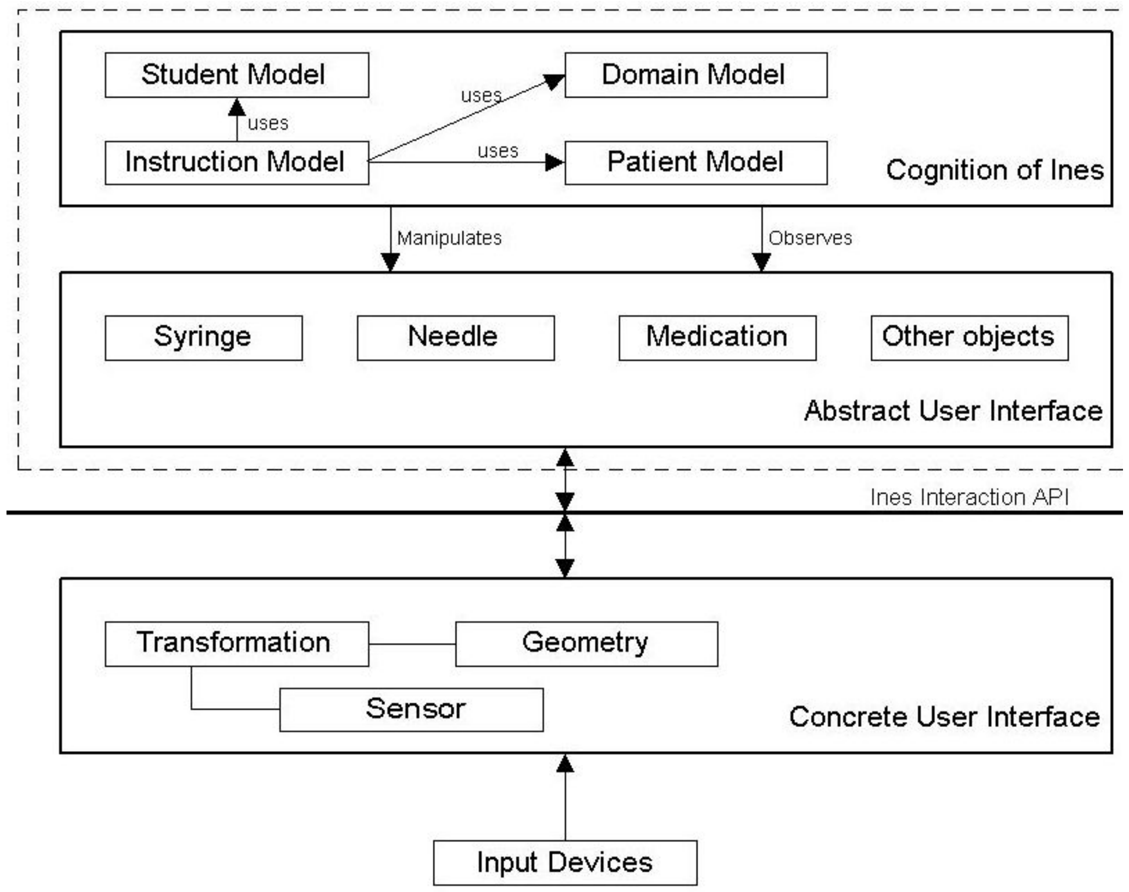


Figure 1. The system architecture of Ines.

3.1 Input from Devices

The first main part is formed by the input from the real world. This can be any input from any device, such as input from data gloves, mouse/keyboard, or haptic devices. The input is used to perform actions in the concrete user interface. For example, moving around in a virtual 3D user interface by means of a data glove will change the viewpoints of the user interface. In this way, the concrete user interface can be manipulated by the user.

Since Ines is completely separated from the concrete user interface, Ines does not care which kind of input device is used, but only expects to receive the necessary data from the concrete user interface, such as the position of objects like syringes and needles

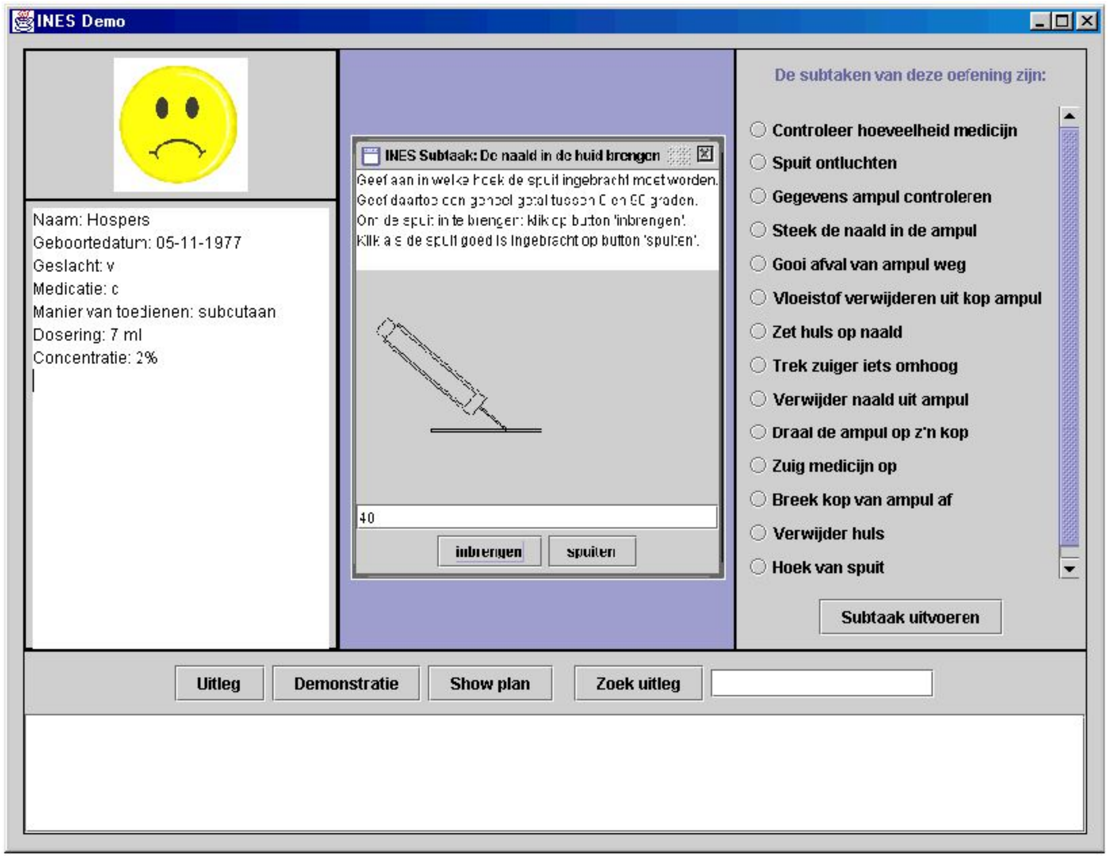
[1] Application Programming Interface. The programmer of the user interface should only use methods from the API to provide knowledge to Ines.

within that concrete user interface. The choice of input devices is thus dependent on the concrete user interface that is used, and not of Ines itself. Currently, Ines can display a user interface that is controlled by keyboard and mouse.

3.2 Concrete user interface

The concrete user interface takes care of the visualization of the exercise. This is a very broad definition, which allows a great freedom for the system implementer to alter or create a user interface for an exercise. Any user interface can be implemented, from a simple dialog interface or a 2D graphical interface, to a highly accurate 3D virtual environment. To this point, Ines contains a user interface that mainly uses text and buttons to communicate with the user. Simple illustrations and video demonstrations can be easily added. In figure 2 a particular situation during the interaction with the student is shown in the interface.

The architecture of Ines allows having several different user interfaces for the same exercise. This comes in handy when a different user interface is required for the same



Naam: Hospers
Geboortedatum: 05-11-1977
Geslacht: v
Medicatie: c
Manier van toedienen: subcutaan
Dosering: 7 ml
Concentratie: 2%

Geef aan in welke hoek de spuit ingebracht moet worden.
Geef daartoe een geheel getal tussen 0 en 90 graden.
Om de spuit in te brengen: klik op button 'inbrengen'.
Klik als de spuit goed is ingebracht op button 'spuiten'.

40

De subtaken van deze oefening zijn:

- Controleer hoeveelheid medicijn
- Spuit ontluchten
- Gegevens ampul controleren
- Steek de naald in de ampul
- Gooi afval van ampul weg
- Vloeistof verwijderen uit kop ampul
- Zet huls op naald
- Trek zuiger iets omhoog
- Verwijder naald uit ampul
- Draai de ampul op z'n kop
- Zuig medicijn op
- Breek kop van ampul af
- Verwijder huls
- Hoek van spuit

Figure 2 User interface of the nurse training application. The left side of the figure contains data from a patient; the right side contains descriptions of all subtasks. The middle part displays the subtask that is currently executed by the student. In this case, the student has to give the angle in which the syringe has to be injected. This angle should be between 30 and 45 degrees. The bottom part contains buttons that can be used to ask for explanations.

task at beginner or advanced level. Therefore, the content of the concrete user interface as shown in Figure 2 is merely the content of an example user interface.

All user interfaces that are to be used by Ines should be able to communicate important information to Ines. Without this information, it is impossible for Ines to perform her task. For example, Ines needs to know when the student has picked up an object before it can decide whether that is a correct action or not. To establish this communication from the concrete user interface towards Ines, the programmer of the user interface has the Ines Interaction API especially designed for this purpose at his/her disposal. This API is further explained in section 3.3.

3.3 Abstract user interface

The abstract user interface is a high level model of the concrete user interface. The abstract user interface is, contrary to the concrete user interface, a part of Ines herself. This means that the communication between the two interfaces in fact constitutes the communication between the concrete user interface and Ines. Important changes in the concrete user interface are reported to the abstract user interface and vice versa.

The concrete user interface directly communicates with the abstract user interface, with help of the methods from the Ines Interaction API. This API contains a number of methods that have to be called by a concrete user interface, because otherwise Ines will not be able to perform her teaching task. Every time the student takes an action, one of the methods is called. Example actions of the students are start and stop subtasks, or make an object sterile or not sterile. In this way, the system gets information about the actions from the student, and the agents in the instruction model use this information to decide what actions they should take.

By separating the concrete user interface from the abstract user interface, Ines does not have to contain the exact representation of each user interface, but only the general aspects that are applicable to all user interfaces, for example the position of important objects. Ines should know if a needle is on a syringe or not, but it is not important for Ines how the syringe and the needle look.

The fourth part in the Ines architecture is, from a teaching perspective, the most interesting part. This part represents the knowledge and skills of Ines and is therefore the core of the Intelligent Tutoring System. Section 4 will deal with this cognition of Ines.

4 The core of Ines: the cognition

The cognition of Ines consists of four models as can be seen in Figure 1: the domain model, the student model, the instruction model and the patient model. This last model was added to Ines for the purpose of exercises for nursing students. The patient model is an agent that at the moment only describes all knowledge that is known about a patient, like age, sex, religion and so on. In the future, this agent can be extended so that the patient can react to actions from the students, for example, pull back his or her arm in anxiety when a virtual reality representation is used. When exercises for non-nursing tasks are added to the system, the patient model can be ignored or replaced by a similar or other model. This is also the reason why the patient model is not a part of the domain model.

4.1 Domain Model

The domain model in Ines encapsulates knowledge about the exercises that should be practiced. Because Ines must be capable of training several different tasks, and because it is unknown which tasks Ines should train in future, it would be very awkward to program all data about all exercises in the system itself. This would lead to such a large amount of data within Ines, that it would become impossible to handle the data in a structured manner. The solution developed for this problem is to only define the elements of the domain model. These elements do not contain any exercise data yet. The real data can be read from an external information source: an XML-file. Thus, instead of programming each exercise into the system itself, a general domain model is used that is filled in by data presented to Ines.

The XML-file contains all the information that Ines needs to know about subtasks, their order, their constraints and explanations and demonstrations. All subtasks have preconditions and post-conditions, and these conditions are used by an algorithm to check the order of the subtasks and also to check if errors are made within the subtask. The preconditions describe the conditions that have to be present before the subtask is eligible for execution, and the post-conditions describe the condition changes that should have happened after the subtask has been completed.

Explanations associated with conditions will usually mention why a certain condition has to be met before the subtask can be performed. Explanations associated with a subtask itself will describe the necessity of the subtask within the exercise, but can also explain what the student has to do during the subtask. The XML-file also contains references to files that can be used to demonstrate how a subtask has to be performed, and references to the necessary user interface. It is possible to have several different user interfaces for the same exercise, for example for beginners, medium and advanced level students. These references are preventing Ines from containing all user interfaces and demonstrations herself.

When adding a new exercise to the system, the data should be added to an XML-file with exercise data. This can be done by hand, but since an average exercise consists of a lot of data, this task can be very error prone. To prevent mistakes from being made, an editing program has been created that can be used to alter or to create an exercise file.

This program is called “Ines Exercise Editor” and is a stand-alone application that can be executed without the presence of any Ines software. It provides an environment in which the user can create and alter exercise files in an obvious manner. For this, the INES Exercise Editor presents the user with a user interface in which the user can create the alterations in a graphical environment. Within this graphical environment, an exercise is represented by a tree, where each node represents a subtask and each arrow represents a relation between two nodes. An arrow from subtask 1 to subtask 2, for example, represents a relation between the two subtasks in which subtask 1 provides a post-condition that is a precondition of subtask 2. Thus, subtask 1 has to be executed before subtask 2.

The domain model used by Ines will only collect the information that is needed by the model to carry out its function from the XML-file. After reading the necessary data, Ines is able to perform her teaching task. The contents of the domain model are used to observe the abstract user interface. All actions within the interface are interpreted by Ines, and Ines will respond to certain events. This conduct of Ines is determined by the instruction model.

4.2 Student Model

The student model is a model of the user's characteristics and performance. This model contains data about all students that work with the program: all the information that is needed in order to identify the students and to interpret the results of the students.

For the same reason as with the domain model, the student model is also defined in terms of elements that can be filled with data from an XML-file. Examples of data are student numbers, grades, and which exercises have been performed by the student. The results of the student are used by Ines to determine how often Ines has to provide feedback or explanations.

A list of exercises is kept for each student. This list contains all the exercises that the student has done in the past, and the results that are achieved for these exercises: the number of times that the student has tried the exercise and, when the exercise is finished successfully, the grade that the student received for the latest attempt. This grade is calculated by one of the agents in the instruction model (the Examination Agent) and then stored in the student XML-file. Also for each subtask, the number of tries is kept, and for each try if the student has passed the subtask successfully.

4.3 Instruction Model

The instruction model observes the world, the task and the user's actions. It can manipulate the world and the user through utterances and actions. These actions include giving a demonstration of the next step of the task, telling what the next step is, or explaining a certain step.

Ines is intended to be used to train nursing exercises. This type of exercise usually consists of several subtasks that have a predefined order amongst them. Ines should teach such an exercise, in that the student learns to perform the subtasks in the correct order and in the correct way. The teaching strategy that is used is the practice and drill teaching method [10]. With this practice and drill method, the student has to perform the exercise over and over again, until (s)he masters the task.

Although not implemented in the Ines prototype, one could propose to add the problem solving teaching strategy to Ines, in that the student will be presented with a problem during the nursing task. An example of such a problem could be a noncooperating patient or an unsuitable environment to administer the injection. By using problem solving in the learning process, the student will obtain better learning results, because the student has to think of a solution himself. Adding problems to the exercise should be fairly simple, since the system is already equipped with an additional agent that represents the patient.

In addition of adding problem solving, one could also add questioning to the Ines system. In this case, Ines could present the student with a number of questions during the execution of the exercise. One could think of questions like: “How many ml of fluid do you have to inject into the patient?”, “How can you observe that the injection site has been well chosen?”, or “What is the correct order in the following subtasks?” Adding these kinds of questions to the Ines system poses a problem, however. Since Ines is a very general teaching system, she has no idea herself of the answers to these questions. It is thus necessary to add the questions to the exercise data. This can be done by specifying some questions with each subtask. Ines could then read the questions and the answers into her domain model and can subsequently use them to ask to the student.

Ines is an Intelligent Tutoring System based on agent technology. This means that agents are used in the instruction model to provide the instruction to the student.

One could also use simple method calls, implemented within normal Java classes, to provide this instruction. In that case, however, Ines can only be run on single machines. Since Ines is an ITS and will therefore be used by a lot of students on several different machines, one can imagine that it would be very convenient to have the Ines software distributed over several different computers. In case of distributing Ines over several machines, Ines must work with agent technology in order to provide instruction.

In addition, agents can be easily turned off or on by Ines whenever necessary. In future, for example, Ines could consist of several groups of agents, where each group provides instruction for a certain subject matter. One group of agents could be specifically for nursing tasks, whereas another group of agents could be for carpentry tasks. When a certain group of tasks has to be trained, Ines can turn on the agents concerning that group and turn off all other agents.

All agents within Ines are divided into two groups: sensor agents and processing agents. For the implementation of the agents, the agent platform developed by the Parlevink group is used. In this platform, there are two kinds of agents: proactive agents and reactive agents. Reactive agents only take actions if they receive messages from other agents, while proactive agents take actions once in a particular amount of time, for example per second. The sensor agents in Ines are proactive agents; the processing agents are reactive agents.

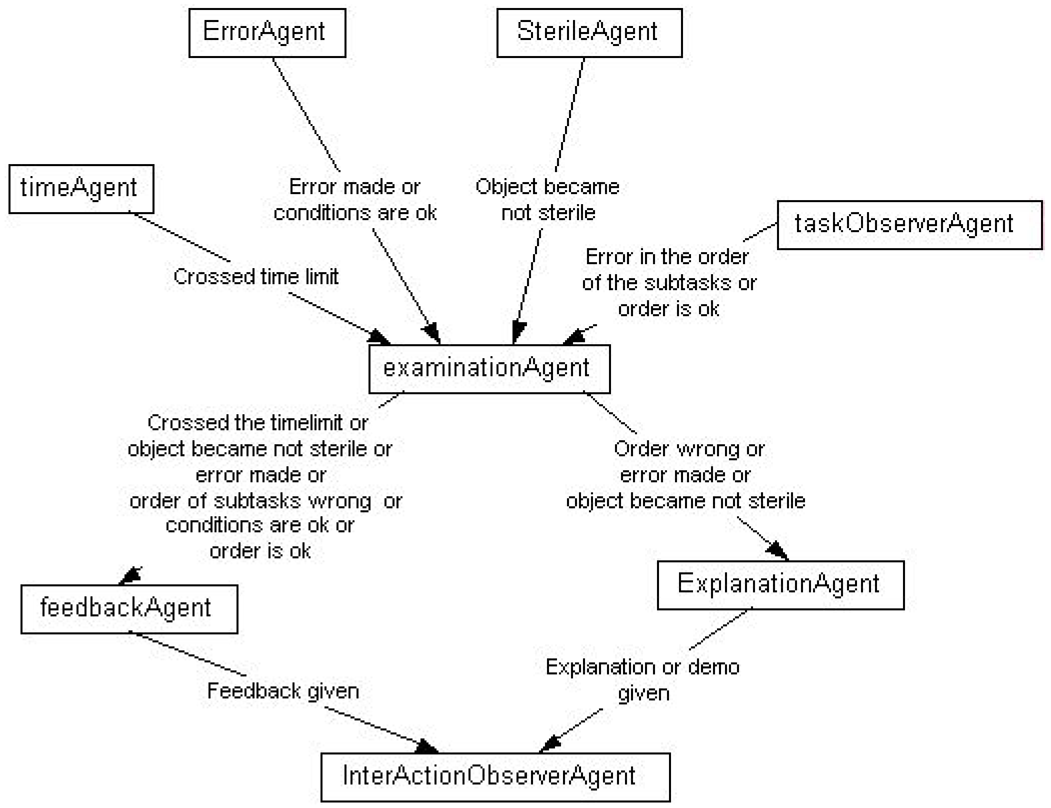

ErrorAgent
SterileAgent
timeAgent
Error made or conditions are ok
Object became not sterile
taskObserverAgent
Crossed time limit
Error in the order of the subtasks or order is ok
examinationAgent
Crossed the timelimit or object became not sterile or error made or order of subtasks wrong or conditions are ok or order is ok
Order wrong or error made or object became not sterile
ExplanationAgent
feedbackAgent
Explanation or demo given
Feedback given
InterActionObserverAgent

Figure 3. The agents. An arrow between two agents means that the first agent sends messages to the second agent.

Each proactive agent observes the user interface with respect to one single aspect. When such an agent has made an observation, it will send it to a reactive agent. Reactive agents receive the observations and respond to it. In Figure 3 can be found which agents send messages to which other agents.

Proactive agents

Distinction is made between four different proactive agents: the Time Agent, the Error Agent, the Task Observer Agent and the Sterile Agent.

The Time Agent keeps an eye on the elapsed time during a subtask. To do this, the Time Agent uses information from a time logger. This logger will log the time every time a subtask is started or finished. In addition, the subtask number is logged. All log-information is time-dependent, that is, if subtask number two is logged after subtask number one, then the latter subtask has been started/finished before subtask number two. The Time Agent will look for subsequent subtask numbers. If a pair has been found, then a subtask has been completed (due to the time-dependency of the log-information) and the agent will call for the time values of those subtask numbers. The time values are then used to compute the time-difference and thus the time that it took for the student to complete the specified subtask. The Time Agent will compare the time-difference with the allowed duration of that subtask. If the student has exceeded the time-limit, the agent will send a message to the Examination Agent. Otherwise, the agent will take no action what so ever.

The Error Agent checks if the student makes errors while carrying out the subtasks. Information from an error logger is used to complete this task. Each log from the error logger contains a subtask number and a list with conditions. Each time the student takes an action the new conditions are logged. Once per second, the Error Agent reads the new logs from the error logger. The subtask number is used to get the post-conditions of the subtask. The logged conditions are compared with these post-conditions. If a logged condition is a post-condition of the subtask, then the action of the student was correct. But if a logged condition is not a post-condition of the subtask, then the Error Agent deduces that an error was made. When the agent detects an error, there are two possibilities. If a condition is wrong, it can be the case that the opposite of the condition had to be true, or that neither the condition nor the opposite of the condition is a post-condition of the subtask. In both cases, the Error Agent sends a message to the Examination Agent. When the opposite of the condition had to be true, the message contains the subtask number, the opposite of the condition that was wrong, and an indication that the condition was wrong. When neither the condition nor the opposite of the condition is a post-condition of the subtask, the message contains the subtask number, the condition itself and an indication that the subtask was wrong. These indications are used by the Explanation Agent to decide what kind of explanation should be given: an explanation about the condition or an explanation about the subtask.

The Task Observer Agent is responsible for keeping an eye on the order in which the several subtasks are performed. This means that the Task Observer Agent will check whether the order in which the subtasks are performed is correct and, in addition, whether the student doesn't mix up a number of subtasks. An example of the latter is when a student first starts subtask one and then starts subtask two without finishing the first subtask at all. To be able to do his job correctly, the Task Observer Agent uses data provided by a logger, which is a simple Java-class that logs information whenever asked by the Ines Interaction API (and is thus not an agent). This logger will log on every occasion that the student either starts a subtask or finishes one. This log-information can subsequently be accessed by the Task Observer Agent and can thus be used to generate liable information about the order in which the subtasks are performed. Two different checks on this information are performed by the Task Observer Agent. First, the agent will check whether a subtask that has been started, has also been finished before a new subtask has been started. This means that the agent will make sure that the student does not mix up subtasks. To do this, the agent will check the log-information provided by the ordering logger and will check for pairs of subtask numbers in this information. This is sufficient, since the log-information is time-dependent. The second check performed by the Task Observer Agent is the order of the subtasks. To do this, the agent uses an internal variable that keeps track of the most recent completed subtask. In addition, the agent will store a set of subtask numbers that are eligible for execution with regard to this most recent completed subtask. These subtask numbers are determined by using the planner described in section 6. Every time the agent notices that a new subtask has been completed, he will check whether this subtask number is an element of the list of planned subtask numbers. If so, the order is correct, otherwise the order is incorrect. In either case, the Task Observer Agent will send an appropriate message to the Examination

Agent. After the check, the agent will update the most recent completed subtask and the planned subtasks to go along with it.

The Sterile Agent is the only agent specified within Ines that is specifically designed for teaching nursing tasks. Most tasks do not require an agent that keeps an eye on the sterility of the used objects, but nursing tasks and other medical tasks do. Thus, it is necessary to implement this agent within Ines. For all other tasks not using this agent, the agent can be replaced by another kind of agent. The Sterile Agent has to look if there are objects that became not sterile unintentionally. It uses a sterile logger to do this. The system always adds a log to a list in this sterile logger when an object becomes unsterile. The Sterile Agent reads the logs of the sterile logger once in a particular amount of time. For the new logs, it checks if the object became not sterile unintentionally. When the agent discovers that an object became not sterile unintentionally, it sends a message to the Examination Agent. This message contains the number of the subtask in which the object became not sterile, a description of the object, and the string representing the condition that the object became not sterile.

Reactive agents

In the system, there are also four different reactive agents: the Feedback Agent, the Explanation Agent, the Interaction Observer Agent and the Examination Agent.

The Feedback Agent is able to generate either positive or negative feedback. The process of the generation of both positive and negative feedback is identical, and therefore only one agent exists for both types of feedback. The Feedback Agent receives input information from the Examination Agent, which detects whether a student has performed a subtask (in)correctly. This received information is used to generate appropriate feedback. This means that Ines will give positive feedback in case the student has performed the subtask correctly, and negative feedback otherwise. It is, however, very undesirable that Ines will generate feedback on every occasion. Just imagine having feedback after completion of every subtask, especially when a novice has a try at an exercise. The novice would get depressed of all of the negative feedback that (s)he would receive! It is therefore possible to change the settings of the number of times that Ines will give feedback. The number of times that feedback is generated is dependent on the quality of the student (i.e. how good is the student at the exercise. A good student makes less mistakes and thus it is reasonable to be more strict with negative feedback and less strict with positive feedback), and the preferences of the teacher (i.e. teachers differ in strictness with regard to feedback, in that one teacher will provide feedback more often than another teacher). The strictness of the feedback frequency can be set by using the strictness variable. This variable is then used as a countdown towards the generation of feedback. It is also not very desirable that the Feedback Agent always presents feedback after exactly three times (or any other number of times). This would make Ines appear very rigid, which usually is not the case with real teachers. If Ines is to resemble a real teacher as much as possible, she should be able to ‘play’ a little with the feedback frequency. To provide this play area, the Feedback Agent is equipped with a method that adds a small variance in the strictness variable. This means that if the strictness is set to three, the agent will variate this strictness between two and four (e.g. sometimes the

Feedback Agent will have a strictness equal to two, whereas at other times the strictness will be either three or four).

The Explanation Agent determines when explanations or demonstrations are given, and which explanations and demonstrations are given. There are different kinds of explanations: explanations and demonstrations from subtasks, explanations from conditions and partial order plans. For the different kind of explanations, different algorithms are used to determine if an explanation should be given and which one. Dependent on the input, the Explanation Agent determines which kind of explanation should be given, and thus which algorithm should be used. The Explanation Agent always receives messages from the Examination Agent when something goes wrong with a subtask. The messages contain of the name of the agent who detected the mistake. The actions of the Explanation Agent depend on the agent that detected the mistake. Another type of input for the explanation agent comes if the student clicks on one of the buttons in the explanation panel (the bottom part of Figure 2). Dependent on the button selected, the Explanation Agent will generate an explanation of the current subtask or a demonstration of the current subtask, or it will return a list with all explanations (both from subtasks and from conditions) containing the string given by the user. Every time the agent gives an explanation or demonstration, it will also send a message to the Interaction Observer Agent. This message contains a string indicating the kind of feedback that was given, and if possible the explanation itself.

The Interaction Observer Agent is responsible for keeping up which feedback the system has given to the student. Every time the Feedback Agent gives feedback or the Explanation Agent gives an explanation or a demonstration, they send a message to the Interaction Observer Agent. Such a message contains the number of the subtask for which the feedback is given and the type of the feedback (positive feedback, negative feedback, explanation or demonstration). When the Interaction Observer Agent receives such a message, it logs the data in a file and in a list. In this way, the data can be used later to see what the student has done and what he or she should know.

The Examination Agent plays a central role in the complete arsenal of agents (see Figure 3). The Examination Agent receives all error messages from the sensor agents and acts upon them. Dependent on the agent that has sent the message and on the contents, a new message is sent by the Examination Agent to one of the other processing agents. In addition to handling the stream of messages between the agents, and thus acting like a traffic controller, the Examination Agent is also responsible for updating the student results during the execution of an exercise. This means that the Examination Agent will retrieve the new results of the student and subsequently add these results to the old results of the student. When the student finishes the exercise, the Examination Agent will derive a grade for the student and will write this to the student file. The last task of the Examination Agent is to control the creation of all other agents and their loggers. When instantiating the Examination Agent, all necessary other agents will be instantiated by the Examination Agent. When quitting the Examination Agent, all created agents will also quit.

5 The implemented exercise: the subcutaneous injection

This section describes the exercise that is implemented to demonstrate how the system works. There are two common techniques used for injections, the subcutaneous injection technique (injecting under the skin) and the intramuscular injection technique (injecting in a muscle). There are more injection techniques available, like the intravenous injection (in a blood vessel) and the very specific injections in the heart or a joint. An extra education on top of the standard nursing education is needed for intravenous injections. This type of injection is therefore not admitted to the nursing curriculum. In addition, only doctors and specialists are allowed and trained to perform specific injections into the heart or a joint, so these procedures are not in the nursing curriculum as well.

Using an injection for the administration of medication has a number of advantages with respect to orally administered medication, in that the medicine usually works faster and never comes into the digestive system. An injection also leads to a better balanced amount of medicine in the blood stream. But there are also a number of disadvantages that can be identified. The procedure for administering an injection is quite a bit more difficult than orally administering medication, and therefore more time-consuming. In addition, an injection is not without hazard in that blood vessels or nerves can be damaged easily. Most people are afraid of having an injection and so it is important to execute the injection as painlessly as possible.

There is chosen to implement the subcutaneous injection, because this exercise is a good example of a nursing task that has to be practiced much before it can be done with real patients. In a virtual environment or with help of simulation, it is possible to show what happens when the injection is not given in the right way. For example, it is possible to show what happens with the patient when there is air in the syringe when the injection is given.

Executing a subcutaneous injection consists of a number of steps. The order of these steps is fixed. The nurse has to check the data of the injection: what kind of liquid should be injected and how much. All materials must be made ready for use: a suction needle should be attached to the syringe, the medicine should be drawn up, and then the suction needle should be replaced by an injection needle. Then, the injection should be given, and after that, everything should be cleared up. These are in short the steps of giving an injection, but all the mentioned steps consist of a number of sub-steps.

For this task, an exercise file and an interface are made. The exercise file contains data about all the subtasks, for example their pre- and post-conditions and explanations. An example of such a precondition is that the injection needle should be sterile before the injection is given. An example of a post-condition is that the syringe is empty after giving the injection. The implemented interface can be found in figure 2. The student has to click on the description of a subtask on the right side of the interface that (s)he wants to execute. Then, a new window appears in the middle part of the interface. In this window, the student has to execute the chosen subtask, or a movie is played with a demonstration of the subtask.

6 Planning in Ines

In Ines, two kinds of planning are used. The first determines only all subtasks that may be performed next, and is used to determine if the student does the subtasks in a right order. The second is partial order planning, which is used to explain the right order of the subtasks to the student when (s)he makes too many errors. A partial order planner is a planner that can represent plans in which some steps are ordered (before or after) with respect to each other and other steps are unordered [11]. The partial order planning algorithm that is used in Ines is a deterministic implementation of the nondeterministic algorithm described in [11], section 11.6.

The order of the subtasks for the exercise that is used for the prototype, the subcutaneous injection, is fixed. For that reason, the choice could be made to tell exactly to the system in which order the subtasks can be performed. But in the future, the intention is to add new exercises, and to program as little as possible for this in Java. For the new exercises, it is quite possible that the order of the subtasks is not fixed. To prevent that all possible orders have to be programmed in the system (because the number of possible orders increases very fast if there is a number of choices in the order), another choice has been made: for both kinds of planning, the preconditions and post-conditions of the subtasks are used. To make it easy to put these conditions in the exercise file, and to write as little Java-code as possible, strings are used for these conditions in the interface and in the exercise file. When a condition is added to the system, it is put into a special class for conditions, which is in fact only a string, but in this way it is easier to use other kinds of conditions in the future, like propositions and predicates.

7 Testing and evaluation

Ines has been tested in two different ways. The first one is by performing unit testing on several classes contained by Ines. This type of test is intended to test the functioning of individual classes, e.g. testing whether the class “timeAgent” is doing what is expected. In addition, Ines has been tested as a whole. This means that exercises and a user interface have been designed for Ines (for the subcutaneous injection) and that the complete system has been tested. The quality of the user interface is not optimal and is not usable for teaching students a subcutaneous injection as if it were real. At most, the order of the subtasks of the exercise is taught. This, however, is sufficient to test Ines with respect to some functions, for example what happens if the student makes an error during a subtask and what happens if the student does the subtasks in a wrong order.

One problematic situation came to light. When a student merely guesses the order in which the subtasks should be executed (and therefore chooses the wrong subtask most of the times), but executes the subtasks itself correctly, Ines will present the student with positive and negative feedback or an explanation at the same moment. The reason for this is that the Feedback and Explanation Agents both receive messages from the Error Agent as well as the Task Observer Agent. The Error Agent communicates the correct execution of the subtask, whereas the Task Observer Agent communicates the wrong order of the subtasks. Both the Feedback and the Explanation Agent respond to both messages, causing the confusing output from Ines. This problem is solved in the following way: the Error Agent does not only check if the subtask is performed in the right way with help of the post-conditions of the subtask, but if the subtask is performed correctly, it will also check if the new conditions also hold in the environment conditions. The environment conditions are only updated when the order of the subtasks is correct. Thus, when the order of the subtasks is wrong, the Error Agent will not send a message to the Examination Agent that a subtask is performed correctly.

In all other tested cases, the system gave the expected explanations and feedback that was to the point.

The most important requirement of the system is that it should be flexible, thus maintainable and adaptable to other situations. It should be possible to add new exercises or new agents without any large changes of the system itself. For the exercises, this goal is achieved. Adding new instructional methods is also possible, but not in a fast and handy way. New agents can be added, but they have to be integrated with the other agents, in the sense that the new agent must be able to send messages to the existing agents, and eventually to receive messages from the other agents. To reach this, the existing agents should also be changed a bit. The new agents have to use the agent platform from the Parlevink group.

Ines distinguishes herself from most other ITSs in that is has become a generic learning system. Although implemented specifically for nursing tasks, Ines is capable of teaching other tasks as well, because the user interfaces and the data about the tasks are completely separated from Ines. It is not necessary to make any changes to the instruction- and student models of Ines when other exercises are added.

8 Future work

A next step in the Ines project is to make a more advanced user interface, for example a 3D virtual environment using haptic feedback devices. As said in section 5, in a virtual environment or with help of simulations, students can learn what happens when they make errors, without doing damage, for example to real persons. In this way, virtual environments and simulations offer possibilities for explorative learning. In the already implemented user interface, movies are used for most subtasks, and in this way, the student can only start and stop subtasks and sometimes click on a button. In the new user interface, the student must be able to take objects and to perform actions with them. With such an interface, Ines can be tested by students, and they can give feedback on the working of Ines and tips for improvement and extensions. When Ines is tested in this way and found to be working, new exercises can be added to the system and students can learn them.

Ines is extendable in many ways. A number of extensions we are working on are:

- Change Ines so that students can also be examinated. This mode differs from the training mode in that the student is not able to ask any questions and that Ines will not give any feedback or hints while the student is executing the assignment;

- Make Ines able to offer more or less guidance to the student depending on his/her performance;
- Make Ines able to offer feedback about the whole exercise when the student has finished the whole exercise. Now, feedback is only given at the moment that the student does something right or wrong;
- Make a virtual representation of the tutor that can be seen on the screen. This representation can be made so that it can give demonstrations, and facial expressions can be used to show the students if they are doing well or not.

Presently, the teacher has to use the student XML-file to look at the results of the students and to add new students to the system. A separate program should be written to make it easier to check the results. It should be possible to check which exercises the student has performed and how. It can also be used to look how many students have executed a particular exercise and to calculate the average results. The teacher must be able to print al the results and to add new students to the XML-file.

Because the idea was to make an Intelligent Tutoring System for nursing students, the following idea rose: connect Ines with a mechanical limb that has sensors to provide Ines with input. In this way, the student is able to train for example the subcutaneous injection on a life-like body part, whereas Ines will evaluate the input received from the limb's sensors. This input will then be used by Ines to guide the student through the training process.

9 References

1. Nijholt, A., Hulstijn, J.: Multimodal interactions with agents in virtual worlds. In *Future directions for Intelligent Information Systems and Information Science*, Studies in Fuzziness and Soft Computing (2000) 148–173.

2. Evers, M., Nijholt, A.: Jacob - an animated instruction agent for virtual reality. In *Advances in Multimodal Interfaces* - ICMI 2000, Proc. Third International Conference on Multimodal Interfaces (2000) 526–533.

3. Broersen, A., Nijholt, A.: Developing a virtual piano playing environment. In IEEE *International conference on Advanced Learning Technologies (ICALT 2002)* (2002) 278–282.

4. Johnson, W.: Pedagogical agents. In Global Education on the Net 1, Proceedings of ICCE'98, the Sixth International Conference on Computers in Education (1998) 13–22.

5. Rickel, J., Johnson, W.: Steve: A pedagogical agent for virtual reality. In Proceedings of the Second *International Conference on Autonomous Agents* (1998) 332–333.

6. Rickel, J., Johnson, W.: Animated agents for procedural training in virtual reality: Perception, cognition and motor control. In: *Applied Artificial Intelligence* **13** (1999) 343–382.

7. Shaw, E., Ganeshan, R., Johnson, W., Millar, D.: Building a case for agent-assisted learning as a catalyst for curriculum reform in medical education. In: Proceedings of the *Int. Conference on Artificial Intelligence in Education* (1999) 509–516.

8. Los Arcos, J., Muller, W., Fuente, O., Orue, L., Arroyo, E., Leaznibarrutia, I., Santander, J.: Lahystotrain - integration of virtual environments and ITS to surgery training. In: *Intelligent Tutoring System*s (2000) 43–52.

9. Shaw, M., Garlan, D.: *Software Architecture, perspectives on an emerging discipline*. Prentice Hall (1996).

10. Ornstein, A., Lasley, T.: *Strategies for effective teaching*. McGraw-Hill Higher Education (2000).

11. Russell, S., Norvig, P.: *Artificial Intelligence, a modern approach*. Prentice Hall, Inc. (1995).