FLEEBLE AGENT FRAMEWORK FOR TEACHING AN INTRODUCTORY COURSE IN AI

M. Pantic, R.J. Grootjans and R. Zwitserloot
Delft University of Technology – Electrical Engineering, Mathematics and Computer Science
Man-Machine Interaction Group, Mekelweg 4, 2628 CD Delft, The Netherlands
M.Pantic@ewi.tudelft.nl, grootjans@gmail.com, reinierz@gmail.com

ABSTRACT

This paper describes both a Java-implemented agent framework and the practical assignments designed for the purposes of teaching an introductory undergraduate Artificial Intelligence (AI) course. Although numerous agent frameworks have been suggested in the vast body of literature, none of these are simple enough for usage by first year undergraduate students. Hence we set out to create a new framework, which would realize the requirements of the course and be usable by first year students. The students' evaluation of the framework's prototypes and subsequent research into the current state of Java-based agent frameworks has led to a number of improvements which will be discussed in this paper. The agent framework embodies the concepts of multiagent systems, concurrency, persistency, distribution and mobility. Classroom experience suggests that the course material, that is, the developed agent framework and the set of practical assignments, is highly suitable for the purposes of teaching undergraduate students basic AI concepts including semantic networks, rule-based reasoning, and the intelligent agent paradigm.

1. INTRODUCTION

As a part of a new educational program, called Media and Knowledge Technology, of the Computer Science faculty at Delft University of Technology, the Netherlands, an introductory first-year undergraduate course on Knowledge Engineering has been introduced in the academic year 2001-2002. The main aim of this course is to achieve the following:

1. Introducing the basic AI concepts and relevant techniques including search algorithms, knowledge representation techniques, rule-based reasoning, and agent technology.
2. Explaining and instructing on issues related to AI programming in general and intelligent multi-agent systems in particular.

In contrast to the classic approach to teaching AI, which is an objectivist approach [1], this course approaches AI as a set of techniques for making software that is more intuitive and easier to use, and for making users more productive. Therefore, this course had to depart from the orthodox objectivist approach where assessment exercises have no real connection to how the student will apply the newly obtained knowledge and skills to previously unseen, real-world problems. Recent studies suggested, however, that the first-year undergraduates are not ready for a pure constructivist teaching approach, in which they are given a vaguely specified problem statement that they should refine and for which they should then develop a solution. Hence, a pragmatic synergy of objectivist and constructivist approaches to teaching programming to novices has been adopted for the AI course in question. The course has been envisioned to include 20 hours of lectures (part one of the course) and 80 hours of practical work (part two of the course). The practical work has been envisioned to build on the lectures by having the students create intelligent multi-agent systems that use the AI techniques taught in the course and represent solutions to well-defined assignments (objectivist approach) aimed at monitoring and retrieving relevant information from Internet (constructivist approach).

Although numerous agent frameworks have been suggested in the vast body of literature [2], none of these are simple enough to be used by first-year undergraduate students. They lack readable documentation and readily available examples or they lack a Graphical User Interface (GUI) altogether or the available GUI is not self-explicatory and learning to work with the GUI is time consuming [1]. Hence, we set out to create a novel Java-based agent framework that would yield a tool appropriate for use in the AI course in question.

In 2002, we developed a first prototype of the framework, which we called Simple Agent Framework (SAF) [3]. The aspects of SAF in need of enhancement, as indicated by the students who evaluated the course material, concerned the improvement of the GUI, enabling agents to load/instantiate other agents, and solving problems caused by the threaded nature of SAF [3]. The realization of these aspects led to a novel version of SAF [1]. This tool was used for the 2003 edition of the course. Although in 87% of cases, students evaluated the used course material (SAF and the assignments) as being suitable for the course in every aspect, a number of novel issues have emerged. First, several teachers of the Computer Sciences department have expressed their interest in using the framework in the future to teach existing and new Computer Science courses. The analysis of the courses in question suggested that the framework should support object-based communication, the mobile agent concept, and the simulation of multiple agent frameworks on a single platform. Second, we wanted to include a novel assignment in the coursework with the goal of introducing students to the concepts of distribution and mobility, the concepts that make the agent technology such a hot topic [4]. The realization of these aspects led to a new design of SAF, which we named *Fleeble*.

2. FLEEBLE AGENT FRAMEWORK

Fleeble can be seen as a common programming interface delimiting the behavior of all the agents integrated into the framework. Its functional specification can be summarized as follows.

Fleeble enables easy addition of intelligent agents. The design that was chosen is to have the framework instantiate and configure the agent and then to start it up in a separate thread [1]. This gives the agent some autonomy, although the agent is running in the framework's process space. An agent can also instruct the framework to start up another agent. The framework keeps track of all agents and their parent agents. So, a single agent can be created which starts up the appropriate agents for each multi-agent system. This *kickoff* agent is then the parent agent of all agents that form the multi-agent system in question. Hence, loading one or more multi-agent systems will result in one or more easily traversable trees that will clearly indicate which agents are working together to accomplish a single task (see Fig. 1).

Fleeble supports simple event processing, allowing the agents to handle events coming from the outside world and other agents and to signal events to the outside world. Fleeble offers a message distribution system for communication between agents that is based on a Publish/Subscribe system, which is centered on the concept of a *channel*. Channels are named entities that allow a single message to be delivered to any number of agents, using the following mechanism. An agent informs Fleeble that it is interested in events pertaining to a specific channel (i.e., it subscribes to that channel). An agent can ask Fleeble to deliver a message to a channel (i.e. it publishes to the channel in question). Fleeble creates a "handler" thread for each agent that has subscribed to the channel in question. All handler threads are started simultaneously and deliver the message to the subscribed agents.

Fleeble supports adding domain knowledge and intelligence to agents. A number of AI algorithms were designed and developed in Java, including forward and backward rule-based inference procedures, rule-base constructs, semantic networks constructs, and several search algorithms. The students can use the related Java classes to provide the pertinent functionalities to their agents [1].

Fleeble supports the concept of concurrency needed to allow agents to operate independently and yet at the same time. This has been achieved by starting each agent in a separate thread, allowing it to access the delivery system described above at its own convenience [1].

Fleeble supports the concept of data persistence. This has been achieved by enabling Fleeble agents to instruct the framework to store values referenced by a key. The framework stores this (key, value) pair and allows access to it at any time, even when the execution of the framework has ceased in the meantime [1].

Fleeble supports the concept of state persistence. State persistency allows the programmer to shut down a single agent (or even the entire framework) and to restore it from the point where it was suspended later on, even when the computer has been shut down in the meantime. A Java-implemented state persistency forces certain restrictions onto the programmer, such as the requirement to return control to the framework within a reasonable time span even if the agent was not stored since it was in the middle of processing an instruction. Allowing the agent (that is to be stored) to complete the processing of the current instruction minimizes this problem. Once the current instruction has been handled, the agent will be stopped, and Java serializing will be used to save all pertinent objects (the agent state) to the permanent storage.

Fleeble enables the user to handle malfunctioning agents. The main drawback of the used threaded approach becomes apparent when a malfunctioning agent ends up in a loop and crashes the whole system, causing other, properly functioning agents to become unresponsive as well. As a solution, the GUI of Fleeble runs in a different Java virtual machine and the communication thread is given the highest priority which ensures the GUI will be responsive to the user at all times. Since any item of an agent tree can be suspended, reawakened, or even destroyed, the user is enabled to manually find and destroy any malfunctioning agent.

Fleeble supports Object-based Communication. In the first versions of the framework this capability was limited to sending only simple text strings [3], [1]. In the current version of Fleeble any object can be sent via the delivery system. The ability of Java to inspect objects on the attribute level (using the reflection API) is used to display the contents of a sent object to the user. This removes the need to program the conversion of a given object into a string and back again, which was necessary in the previous versions of the framework [1].

Fleeble supports the concepts of Distribution and Mobility. Distribution is the possibility to exchange messages between agents residing on different frameworks. It has been enabled by establishing socket-to-socket connections between frameworks. Fleeble manages these connections; it creates and closes them as needed. Connections can be used for mirroring channels, translating the concept of a channel into something that transcends the host machine of the agent. Connections are also used to transport agents to another host (see the State Persistency explained above), allowing agents to physically move. This process is called agent mobility. The process of being moved to another host can be started either by the agent or by its parent agent.

Fleeble enables the programmer to emulate multiple frameworks. When testing an environment designed for distributed multi-agent systems, it is not always feasible to reserve a number of computers needed for the pertinent testing. To support proper testing of agents designed to function in a multiple-host environment, Fleeble offers the ability to encapsulate all channel communication for any agent such that the agent "gets the impression" that it has moved to a different host. Agents can communicate with each other (i.e., use the same channels) as long as they run under the same virtual framework.

Fleeble supports auto-compilation. During the software development process, code is usually improved incrementally. This makes it necessary to compile the code frequently. Doing this with an external compiler is time-consuming. To alleviate this problem, we adapted Fleeble so that it detects changes in the source code and automatically recompiles all files that have been changed, even if a multi-agent application is still running. A single click on the *reload* button starts the process of recompiling the source and replacing the running agents with the new version of these agents.

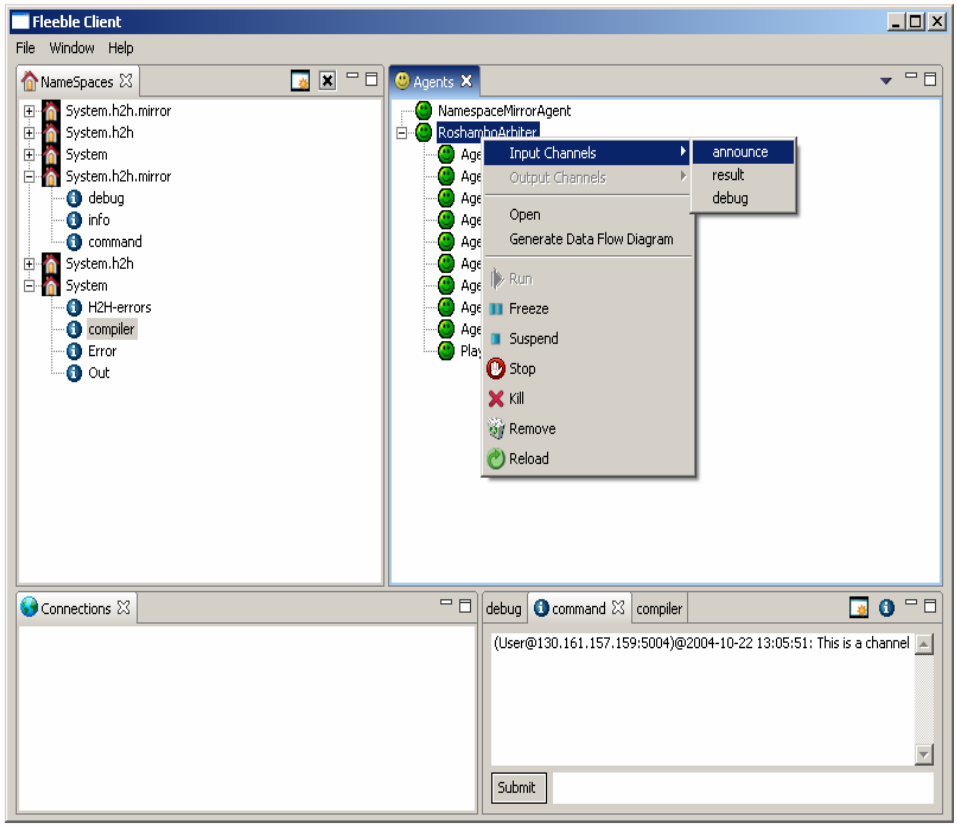

Figure 1. Screenshot of the GUI of Fleeble

Fleeble has a simple, easy to understand GUI. The goal was to develop a direct-manipulation GUI in which WYSIWYG (what you see is what you get) would be the guiding principle [1]. A simple self-explicatory GUI was developed that is easy to understand and use. As can be seen in Fig. 1, the GUI of Fleeble visualizes the overall traffic through channels, the agent hierarchy, and the agents themselves. By clicking the right mouse button on an appropriate entry, the user opens the relevant context menu with the related actions. This menu enables quick perception of and access to the relevant data.

The Fleeble GUI supports monitoring of external connections. Fleeble makes an active network connection when it should transfer a message or mobilize an agent to a remote platform. Connections to remote platforms and the related information such as the address and the port number are displayed in the connection view (see Fig. 1).

The Fleeble GUI supports remote connections to a Fleeble server. The Fleeble GUI runs independently of the main agent framework and it does not need to be started in the same physical computing environment in which the agent framework runs. A user can log into a Fleeble server either locally or remotely, via a network connection. The amount of control a remote connection can assert over the agent framework is nevertheless rather limited. Operations such as loading an agent are not possible from a remote location.

3. PRACTICAL ASSIGNMENTS

An introductory assignment and three programming assignments constitute the practical part of the AI course in question. The introductory exercise was created in order to identify students who do not have a sufficient level of experience with the Java programming language to complete the course. Furthermore, it introduces the students to the basics of programming an agent in Fleeble. The programming assignments delve into the issues of how to employ rule-based reasoning, semantic networks, search algorithms, and mobile agent concept to monitor, filter, and retrieve relevant information from Internet. The assignments have been designed for teams of 5 or 6 students.

The **Assignment A** is as follows: *Create an agent that can consistently beat 7 out of 9 Roshambo agents in a tournament. Explain the strategies the agents employ and describe the strategy used to defeat them.*

This first assignment introduces epistemic logic and agent technology by letting students play a game of Roshambo against the computer. Roshambo is also known as the Rock-Paper-Scissors game. A Roshambo tournament consists of a large number of games in which several opponents play against each other and against a set of agents that have a predictable strategy. By using a strategy that predicts the moves of other Roshambo agents an agent will become predictable and subject to other prediction strategies. The assignment starts with the requirement to play games of Rock-Paper-Scissors against 9 provided agents. Each agent uses a predictable strategy. Once the students have figured out the strategy, they should implement a single agent that can beat at least 7 of the 9 provided agents in a tournament. A tournament consists of hundred games of Roshambo. A game that has been won is awarded one point, a lost game deducts one point, and a tie does not change the score. To accomplish the goal of identifying students with an insufficient level of experience with Java, this assignment has to be completed individually, unlike the other assignments.

The **Assignment B** is as follows: *Create an agent-based system that translates a questionnaire, filled out by each participant in this course, into a chart that expounds the suitability of each participant for being a part of your team. Use Fleeble to build the required system and employ rule-based reasoning. Explain the choice of the utilized inference procedure and the final ranking of the students being a part of your team.*

This assignment has been used in all previous versions of the practical part of the AI course in question. This is because the classroom assessment suggested that this programming assignment is highly suitable for teaching undergraduate students concepts such as rule-based reasoning and multi-agent paradigm [1].

The **Assignment C** is as follows: *Create an agent-based system that retrieves and analyses the BBC news available via the Internet according to your own preferences. Use Fleeble to build the required system and employ the semantic network concepts. Explain the drawbacks (if any) of the used approach.*

This assignment has been used in all previous versions of the practical part of the AI course in question. This is because the classroom assessment suggested that this programming assignment is highly suitable for teaching undergraduate students concepts such as semantic networks and multi-agent paradigm [1].

The **Assignment D** is as follows: *Using the mobile-agent concept, create an agent-based system that the user can employ to download 100 different files in a time span of 10 minutes while being connected to an*

intermittent and unstable network. Each unstable link between two network nodes has a given (random) bandwidth and latency which influences the speed at which the file can be found and send to the host that made the request. Each file is between 2 and 5 megabytes in size. Explain the strategies used.

Due to the dynamic nature of the network in question (i.e., the links between nodes can appear and disappear and the related bandwidth and latency can change), solutions provided by the students ranged from very complex information-driven systems (where each node attempts to communicate any change to the network infrastructure as fast as possible), via heuristic systems (that simply try to go to the nearest node), to search systems (that send out many agents across multiple paths in an attempt to ensure that at least one of them will return to report the most efficient path). In brief, instead building agents that can operate independently of changes in the network, most students attempted to overcome the network's stochastic nature by designing a system that propagates information about changes through the network. Since such designs are too complex to implement in a relatively short time, the students evaluated this assignment as being too complex/ambitious given the time allotted for the assignment (see Table 1). As a consequence, we are currently reformulating the assignment so that the major drawbacks of the possible deterministic solutions are explicitly pointed out. This should inspire the students to try alternative, less deterministic solutions.

Table 1. Results of the questionnaire for surveying students' experiences with the course

Assessment	2002	2003	2004
The utilized tool is suitable for the course	67%	89%	92%
The assignments are motivating	87%	94%	92%
Happy with the produced system: Assignment A	×	×	95%
Happy with the produced system: Assignment B	91%	90%	91%
Happy with the produced system: Assignment C	88%	89%	89%
Happy with the produced system: Assignment D	×	×	21%

4. CLASSROOM ASSESSMENT

In 2002, 73 students participated in the AI course in question. They were divided into 14 teams of 5 or 6 students. Sixty students successfully completed assignments B and C constituting the 2002 version of the course (they had a grade > 5). The results achieved by the students are illustrated in Fig. 2. Of 73 participants, 68 filled out the questionnaire for surveying students' experiences with the course. The survey results are summarized in Table 1.

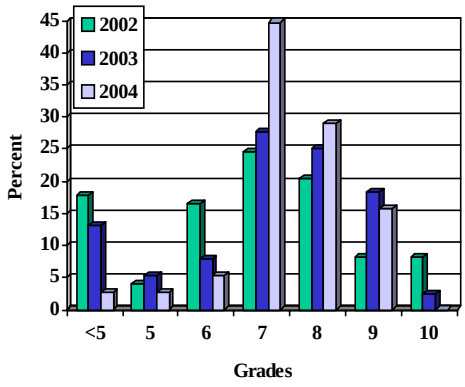

Figure 2. The grades assigned to the students for the quality of the delivered code. In 2002, from 73 participants, 17.8% did not pass the course (had a grade <5). In 2003, from 76 participants, 13.2% did not pass the course. In 2004, from 47 participants, 2.13% did not pass the course.

In 2003, 76 students participated in the course (15 teams). In total, 64 students successfully completed assignments B and C constituting the 2003 version of the course (Fig. 2) and 72 students filled out the questionnaire for surveying students' experiences with the course (Table 1).

In 2004, 47 students participated in the course (9 teams). In total, 39 students successfully completed the assignments A, B, and C (Fig 2). Due to the complexity of the assignment, only one group of students (5 students) completed the assignment D. This is why the scores achieved for the assignment D have not been taken into account when calculating the final grade of each student as shown in Fig. 2. Of the 8 students who did not successfully complete the course, 7 did not have a sufficient level of experience with Java. They were filtered out due to their failure to complete the introductory assignment A. These students have not been taken into account when making the overview shown in Fig. 2. Of the students who

successfully completed the course, 36 students filled out the questionnaire for surveying students' experiences (Table 1).

These results indicate that the developed educational tool (Fleeble) and the specified programming assignments are suitable for the purposes of teaching students the basics of AI, including knowledge representation schemes, rule-based reasoning, and the intelligent agent paradigm. Assignment D however has proven to be challenging for the students. We are currently working on adapting the assignment in order to overcome this problem.

5. CONCLUSION

This paper describes a flexible method of teaching introductory AI using a novel, Java-implemented, simple agent framework that we named *Fleeble*. Fleeble supports the development of intelligent agent applications and it embodies the concepts of concurrency, multi-agency, persistency, distribution and mobility. In contrast to the agent frameworks developed elsewhere [1], and as indicated by classroom experience, Fleeble is a suitable tool for teaching AI programming to novices because: (i) it is simple, it is accompanied by readable documentation, and it provides readily available, useful examples; (ii) it embodies simple, self-explained, visual interaction with the user that may concern one, more, or all currently running agents (Fig. 1). We expect that Fleeble can be widely useful for many different AI courses, including courses for non-computer-science majors.

Based on classroom experience, and in comparison to the commonly applied approaches to teaching introductory AI [1], the significance of the example-based teaching method proposed in this paper can be summarized as follows. The implemented pedagogy represents a synergy of objectivism and constructivism in teaching programming to novices: the students are presented with well-defined assignments (objectivist approach) aimed at monitoring, filtering and retrieving relevant information from the Internet (constructivist approach). Classroom experience indicates that this "hybrid" teaching method significantly increases the extent of learning compared to use of only the objectivist approach to teaching programming to novices [1]. These results strongly indicate that the implemented pedagogy forms a rather effective way of teaching AI to novices. However, to make the designed set of practical assignments a highly effective approach to teaching basic AI concepts including distributed AI, assignment D should be reformulated as suggested by the students who attended the course in 2004 (see section 3).

ACKNOWLEDGEMENT

The authors would like to thank all 175 MKT undergraduates who evaluated the course in 2002, 2003 and 2004. The work of Maja Pantic is supported by the Netherlands Organization for Scientific Research (NWO) Grant EW-639.021.202.

REFERENCES

[1] Pantic M. et al, 2005. Teaching Introductory Artificial Intelligence Using a Simple Agent Framework. In *IEEE Trans. On Education*, accepted for publication.

[2] http://www.agentlink.org/resources/agent-software.php

[3] Pantic M. et al, 2003. Simple Agent Framework: An educational tool introducing the basics of AI programming. *Proc. IEEE Int'l Conf. Information Technology in Research and Education*, Newark, USA, pp. 426-430.

[4] Shoham, Y, 1999. What we talk about when we talk about software agents. In *IEEE Inteligent Systems*, Vol. 14, No. 2, pp. 28-31.