Teaching Introductory Artificial Intelligence Using a Simple Agent Framework

Maja Pantic, *Member, IEEE*, Reinier Zwitserloot, and Robbert Jan Grootjans

***Abstract*—This paper describes a flexible method of teaching introductory artificial intelligence (AI) using a novel, Java-implemented, simple agent framework developed specifically for the purposes of this course. Although numerous agent frameworks have been proposed in the vast body of literature, none of these available frameworks proved to be simple enough to be used by first-year students of computer science. Hence, the authors set out to create a novel framework that would be suitable for the aims of the course, for the level of computing skills of the intended group of students, and for the size of this group of students. The content of the introductory AI course in question is a set of assignments that requires the students to use intelligent agents and other AI techniques to monitor, filter, and retrieve relevant information from the World Wide Web. It represents, therefore, a synthesis of the traditional objectivist approach and a real-world-oriented, constructivist approach to teaching programming to novices. The main aim of implementing such a pedagogy was to engage the students in learning to which they personally relate while attaining intellectual rigor. Classroom experience indicates that students learn more effectively when the traditional objectivist approach is combined with a constructivist approach than when this orthodox approach to teaching programming to novices is used alone.**

***Index Terms*—Agent framework, artificial intelligence (AI) course, intelligent agents, introductory engineering course, Java, rule-based reasoning, semantic network, World Wide Web search.**

I. Introduction

According to the Dutch Statistical Bureau, 77% of Dutch households have an Internet connection [1]. This ever-increasing role of computers in society clearly forecasts which type of working environment and information–communication space Dutch people, and people of the Western Hemisphere in general, are about to use in everyday activities. Even now, the majority of the people living in the Western Hemisphere use a computer to work and the Internet to communicate, to shop, to seek out new information, and to entertain themselves. This trend clearly indicates that in the future people will perform a larger and larger part of their daily activities with the aid of computers in cyberspace, across distance, cultures, and time. Of course, the specifics of such cyber worlds, smart environments, and the related interfaces (which should facilitate easy and natural communication within those environments and with a variety of embedded computing devices) are far from settled. Hence, computing technology breakthroughs are necessary. This necessity forms the main drive behind the abundance of job offers for information technology (IT) specialists in The Netherlands. It also explains why so many of the Dutch prospective students choose just this branch of exact sciences to examine their capabilities, to enhance their skills, and to turn into valuable experts in one of the IT fields. As computers become ever more ubiquitous in society, further developments in computing technology become more and more exciting and economically important in both the IT industry and the academic community.

This view on reality has motivated the development of a new educational program called Media and Knowledge Technology (MKT) of the Computer Science of Delft University of Technology, Delft, The Netherlands. In the academic year 2001–2002, this new program was officially introduced. The main objective of this program is to educate students to become engineers who are able to design and develop intelligent systems for multimedia and multimodal information and knowledge processing and who are able to design, realize, and deploy properly working man–machine interfaces.

As a part of this program, an introductory first-year course on AI was developed with the following aims:

1) to introduce the basic concepts of knowledge engineering and the relevant AI techniques, including search algorithms, knowledge representation techniques, rule-based reasoning algorithms, and agent technology;
2) to explain and instruct on issues related to AI programming in general and intelligent (multi-) agent applications in particular.

In contrast to the classic approach to teaching AI, which is an objectivist approach as shown in Table I, 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 to teaching programming to novices 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 [6]. An alternative, constructivist approach, where an authentic real-world environment is provided in which students apply and test their newly obtained knowledge and skills, seemed to represent a better choice for the AI course in question. Recent studies suggested, however, that the first-year undergraduates are not ready for a pure constructivist teaching approach (e.g., the approach used at Cornell University, Ithaca, NY; see Table I), in which they are given a vaguely specified problem statement that

Manuscript received March 10, 2004; revised September 23, 2004. The work of M. Pantic is supported by the Netherlands Organization for Scientific Research (NWO) under Grant EW-639.021.202.

The authors are with the Electrical Engineering, Mathematics, and Computer Science Department, Delft University of Technology, 2628 CD Delft, The Netherlands (e-mail: mpantic@ieee.org; reinier@zwitserloot.com; robbertjan@dds.nl).

Digital Object Identifier 10.1109/TE.2004.842906

TABLE I
AN OVERVIEW OF AI INTRODUCTORY COURSES LECTURED AT VARIOUS UNIVERSITIES

University	Book	P-Lang.	Teaching Method
Stanford (CS 121)	[2]	Java, C++	Objectivist (Robot navigation problem on a virtual grid with informed search)
Berkeley (CS 188)	[2]	Lisp	Objectivist (Mathematical functions in Lisp like power of, powerset, symbolic differentiation; rationality of slot machine agent and vacuum cleaner agent)
UCLA (CS 161)	[2]	Lisp	Objectivist (Text structure analysis)
Cornell (CS 472)	[2]	open	Constructivist (Design and implement an intelligent system)
Harvard (CS 182)	[2]	Prolog	Objectivist (Connect 4 games)
Toronto (CSC 384	[3]	Lisp	Objectivist (8-puzzle problem solving with informed search, generating a rule base given a set of statements)
Delft (IN 2420)	[4]	none	Objectivist (experiments with a trained neural network for function optimization)
Delft (IN 1851)	[2][5]	Java	Hybrid constructivist-objectivist (see section IV of *this paper*)

they should refine and for which they should then develop a solution [7]. Hence, a pragmatic synergy of objectivist and constructivist approaches to teaching programming to novices has been adopted for the AI course in question. The overall course was envisioned to include 20 hours of lectures (part one of the course) and 80 hours of practical work (part two of the course). This second part of the course, which is the main subject of this paper, addresses various practical issues in AI programming in general and issues in creating properly working intelligent agent applications in particular. It builds on the first part of the course by teaching students how to use AI algorithms, about which they have been lectured, for solving well-defined assignments (objectivist approach) aimed at monitoring, filtering, and retrieving relevant information from the World Wide Web (constructivist approach).

This paper will begin by examining the requirements for the educational tool to be used in the second part of the AI course in question. It will then survey several agent frameworks developed elsewhere which were considered for use, and it will explain the authors' motivations to develop a novel simple agent framework (SAF) for the purposes of the practical part of the AI course in question. Next, the paper will explain the design of the SAF in detail. Two programming assignments constituting the practical part of the AI course in question will receive particular attention. Finally, the paper will discuss experiences of both the class of 2002 and that of 2003, and it will provide concluding remarks.

II. EDUCATIONAL TOOL REQUIREMENTS

In the process of selecting the appropriate educational tool to be used in the AI course in question, the authors identified several constraints. Since the students attending the course are first-year students who have only attended a course on the programming language Java, all programs and examples to be developed have to be implemented in Java. Another important issue, valid for any introductory course, is the focus of the course. The focus should be on learning how to use the basic AI algorithms to solve real-world problems, rather than on learning how to use complex software tools. Hence, the requirements that an educational tool should fulfill in order to be appropriate for the purposes of the AI course in question can be summarized as follows.

1) The tool should support the development of intelligent agent applications aimed at monitoring, filtering, and retrieving relevant information from the World Wide Web.
2) It should be simple and user friendly, and it should include built-in, Java-implemented, agent *templates* that facilitate example-based learning and can be edited to include the desired AI algorithms according to the goals of a programming assignment.
3) It should embody the concepts of concurrency (a kind of multithreaded setup), multiagency (by allowing simple communication from one agent to the other), and persistency (saving settings between executions).

Numerous agent frameworks are proposed in the vast body of literature [8], and many software packages enabling the development of agent-based applications are currently available [9]. Nevertheless, none of these available agent frameworks satisfies all the requirements listed previously. Table II summarizes the properties of the currently existing Java-based agent frameworks with respect to the following issues.

1) Does the developer provide support for the tool?
2) Is the tool available for free?
3) Are useful examples readily available?
4) Is the related documentation readable?
5) Is synchronous agent-to-agent communication (i.e., waiting for reply) supported?
6) Is asynchronous agent-to-agent communication (continuing immediately) supported?
7) What is the communication transmission form?
8) Can the framework control the agents' resources (e.g., the disk or network capacity used)?
9) Can the framework ask an agent to shut down?
10) Can the framework terminate the execution of a malfunctioning agent?
11) Can the framework store agents' states between executions?
12) Can the framework store objects (e.g., a database) between executions?
13) Does a self-explained graphical user interface (GUI) per agent exist?
14) Does the GUI support an overview of all running agents?

As can be seen from Table II, none of the available Java-based agent frameworks is simple enough to be used by first-year undergraduate students. The reason is that virtually all of these frameworks have at least one of the following drawbacks.

- They lack readable documentation and readily available useful examples (see the third and the fourth column of Table II).
- They lack a GUI altogether, or the available GUI is not self-explained and time-consuming consultation of the

TABLE II
OVERVIEW OF THE AVAILABLE JAVA-BASED AGENT FRAMEWORKS

Name	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Agent Factory [9]	●	-	×	×	×	×	PP	×	×	×	×	×	×	×
IBM Aglets [9]	●	Free	×	×	●	●	PP	●	●	×	×	●	●	×
AMETAS [9]	●	-	×	×	×	●	M	×	●	×	×	×	●	×
Beegent [9]	×	-	●	×	×	●	PP	×	×	×	×	-	×	×
Cougaar [9]	●	Free	×	×	×	●	M	●	×	×	×	●	×	×
CIAgent [5]	●	$	●	●	×	●	PP	×	●	×	×	-	●	×
DECAF [9]	●	-	×	×	×	●	PP	×	×	×	×	×	●	×
FIPA-OS [10]	●	Free	×	×	●	●	PP	×	×	×	×	●	×	×
Grasshopper [9]	●	-	●	×	●	●	M	●	-	-	●	×	●	●
Hive [9]	×	Free	×	×	×	×	None	×	×	×	×	×	●	●
JACK [9]	●	$	×	×	×	●	PP	×	×	×	×	-	×	×
JADE [9]	●	Free	●	×	×	●	M	×	●	×	×	×	●	●
JAFMAS / Jive [9]	×	-	×	×	●	×	M	×	×	×	×	×	×	×
Kaariboga [9]	●	Free	●	●	×	●	PP	●	●	×	×	×	×	×
LIME [9]	●	Free	●	●	×	●	PS	×	×	×	×	×	×	×
MadKit [9]	●	Free	●	×	●	●	M	×	●	×	×	×	×	●
NOMADS [9]	×	Free	×	×	×	×	None	●	●	●	×	×	●	×
OpenCybele [9]	●	Free	●	●	●	●	PS	×	●	●	×	×	×	×
Pathwalker [11]	●	Free	×	×	×	●	PP	×	×	×	×	×	×	×
SeMoA [9]	●	Free	●	●	×	●	None	●	●	●	●	●	×	×
Tagent [12]	×	Free	×	×	●	●	PP	×	×	×	×	×	×	×
Tryllian [9]	●	$	●	×	●	×	PP	×	-	-	●	●	×	×
Voyager [9]	●	$	×	×	●	●	PS	●	×	×	●	×	×	×
ZEUS [9]	×	Free	●	×	●	×	PP	×	×	×	×	×	×	×
SAF (this paper)	●	-	●	●	×	●	PS	●	×	●	×	●	●	●

Legend: ● = "yes", × = "no", - = unknown, PP = Peer to Peer, M = Multicast, PS = Publish-Subscribe

documentation is necessary, or a GUI is available per agent but there is no overview of all running agents at the framework level (see the thirteenth and fourteenth column of Table II).

Hence, the authors set out to create a novel Java-based agent framework that would fulfill all the requirements listed previously and that would yield a tool appropriate for use in the introductory AI course in question.

III. SIMPLE AGENT FRAMEWORK

The design of the *simple agent framework (SAF)*, the first version of which was developed in 2002 [13], has been driven by the knowledge about the end users (i.e., first-year undergraduates) and the stipulated purpose of the educational tool to be developed (Section II). SAF 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 (see also the last row of Table II for a summary of the functional specification):

SAF enables easy addition of intelligent agents. This goal could be achieved by having the framework to instantiate and configure the agent and then to call the agent's methods as service routines. That way the framework would be always in control, and it could use intelligent functions as appropriate. This outcome is easy to achieve but hardly results in what one would consider an intelligent agent. Therefore, the design that has been chosen is to have the framework instantiate and configure the agent and then to start it up in a separate thread. In this way, the agent has some autonomy, although the agent is running in the framework's process space.

An agent can instruct the framework to start up another agent (in contrast to the preliminary version of the SAF [13]). The framework, in turn, keeps track of all agents and their parent agents. A highly beneficial effect of this new capability is that the framework no longer needs the clumsy and difficult-to-grasp "batch" concept, which was first used to load different parts of a multiagent system simultaneously. Instead of working with text files containing ordered lists of agents, a single agent can be created which starts up the appropriate agents for each multiagent system. This "kickoff" agent is then the parent agent of all agents that form the multiagent system in question. Hence, loading one or more multiagent systems will result in one or more easily traversable trees that will clearly indicate which agents are working together to accomplish a single task (Fig. 1).

SAF supports simple event processing, allowing the agents to handle the events coming from the outside world or from other agents and to signal events to the outside world. Java uses an event-processing model for various features, including the features of the graphical toolkit (swing). However, this model relies on the source of events to maintain a list of registered event listeners and to deliver the subject events. To alleviate the difficult task of programming agents, a Publish–Subscribe event-processing system has been used in the SAF. This "delivery system" manages events and dispatches them to the interested agents. Both the events coming from the user and those coming from other agents are represented as text strings, each of which is called a *message*. All messages are sent to a *channel*, each of which is analogous to a blackboard. The blackboard concept is known to all students of Delft University of Technology since it is the design pattern used to develop the system for posting and distributing information about courses, exams, homework,

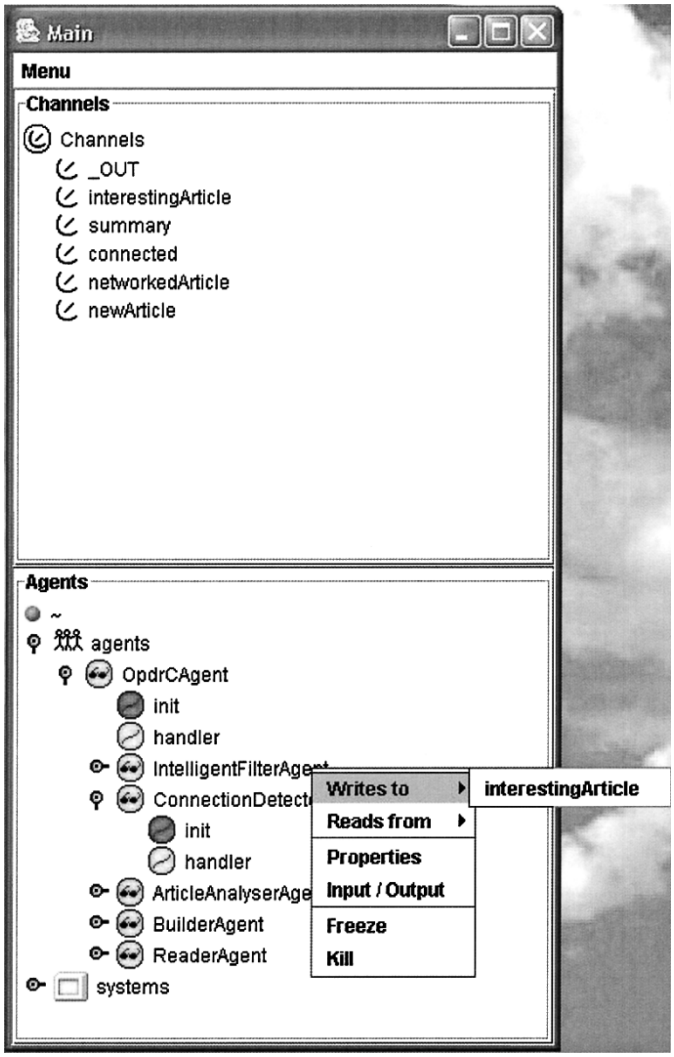

Fig. 1. Screenshot of the GUI of the SAF.

etc. Each agent is accompanied by a text file listing all channels from which and to which the agent reads and writes. If an agent reads from a certain channel, it is notified of messages delivered to that channel. Messages are received through a locally defined (i.e., at the agent level) *handle (channel, message)* method. To send a message, an agent should invoke a globally defined (i.e., at the framework level) *write (channel, message)* method. A complete list of methods supported by the framework is given in Table III.

SAF supports adding domain knowledge and intelligence to agents. Facilitating a composite design has attained this functionality of the framework. Namely, a number of AI algorithms were designed and developed in Java, including forward and backward rule-based inference procedures, rule-based constructs, semantic networks constructs, and several search algorithms. The students can use the related Java classes to provide the pertinent functions to their agents.

SAF supports the concept of concurrency needed to allow agents to operate independently and yet at the same time. This concept has been realized by starting each agent in a separate thread, allowing it to access the delivery system described previously at its own convenience. To prevent overloads, the framework initiates a queue of messages for each agent, stores it locally (i.e., at the agent level), and processes the messages one by one.

SAF supports the concept of persistence. By enabling SAF agents to instruct the framework to store values referenced by a key, an easily understood form of object persistency has been attained. The framework retains this *(key, value)* pair and allows access to it at any time, even when the execution of the framework has ceased in the meantime. To do so, the framework stores *(key, value)* pairs to a text file and loads them when its execution has commenced once again. On the other hand, SAF does not support state persistency, which allows the programmer to shut down a single agent (or even the entire framework) and to restore it later from the point where it was suspended. Although a useful mechanism, a Java-implemented state persistency would have to force certain restrictions onto the programmer, such as the requirement to return control to the framework within a reasonable time span. As a result, a state persistency system is convenient for the user and inconvenient for the programmer. Since the students should use the SAF mainly to program multiagent applications, the decision was made to forgo the implementation of state persistency.

SAF is a graphical, agent-building tool. The goal was to develop a direct–manipulation interface in which WYSIWYG (what you see is what you get) would be the guiding principle. Another goal was simplicity and comprehensibility of the GUI that could make up for differences in computing skills and experience of the target users. Given that the target users were first-year students with rather limited computing skills in the majority of cases, the authors wanted to omit extensive technical terminology, complex screen layouts, irreversible actions, incomprehensible error messages, and unexpected crashes. Thus, a simple self-explained GUI was developed that is easy to understand and use. The GUI of the SAF visualizes the overall traffic through channels, the agent hierarchy, and the agents themselves (Fig. 1). Each agent is represented by an appropriate icon suggesting the state of the agent—a green smiley for an active agent, a red smiley for an inactive agent, and a blue one for a suspended agent. By clicking the right mouse button on an appropriate entry, the user opens the relevant *context menu* with the related actions (Fig. 1). This menu enables quick perception of and access to the relevant data, in contrast to the preliminary version of the SAF [13], which only showed two rigid lists (i.e., a list of channels and one of active agents).

SAF enables the user to handle malfunctioning agents. As is the case with the majority of the existing agent frameworks [8], the SAF relies heavily on threads. In the case of multiagent systems, the main drawback of this 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. A way to tackle this problem is to eliminate the agents that are using too many system resources. However, Java does not offer enough information about system processes to facilitate easy identification of the ones that consume most of the available processing power. As an alternative solution, the GUI of the SAF has been assigned the highest priority, and it has been made responsive to the user at all times.

TABLE III
METHODS SUPPORTED BY SAF

Framework-level methods	*Agent-level methods*
setProperty(*key, value, persistent*) This method allows the agent to change an existing or to create a new *(key, value)* pair. A flag *persistent* enables the agent to define whether the framework should store the property between sessions.	**Init()** This method is called, in a separate thread, when the agent is loaded.
getProperty(*key*): *value* This method returns values set by using the *setProperty* method.	**handle(*channelname, messagecontent*)** Messages that have been sent to a channel from which the agent reads are received and handled by this method.
write(*channelname, messagecontent*) This method allows the agent to send messages to a channel.	

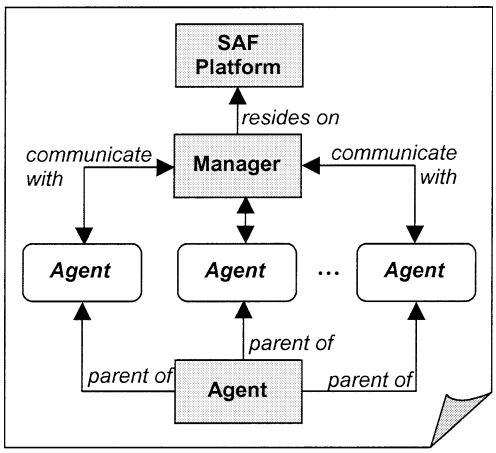

Fig. 2. Relationships between the SAF primary classes and their instances.

Since any item of the agent tree can be suspended, reawakened, or even forcibly destroyed, the user can manually find and destroy any malfunctioning agent. Neither the fastest nor the most elegant solution, it is still considerably better than the one that was available in the preliminary version of the SAF, restarting the framework each time an ill-coded agent entered a loop [13].

SAF uses only two primary classes. The *Manager* is the base class. It defines a common programming interface and behavior for all the agents in the SAF. Within this class, all three global methods listed in Table III are defined. All agents in the SAF communicate with the environment and with each other by using these methods (Fig. 2). Each agent extends the abstract *Agent* class and uses its own versions of the *handle* and *init* methods given in Table III. In turn, the users of the SAF only need to know about these two classes to be able to create new SAF-compatible agents.

IV. CONTENT OF THE COURSE

A. Layout of the Course: New-Media Utilization

Printed media have dominated education for ages. This dominance arose because the knowledge captured in writing and reproduced through printing was effectively searchable, accessible to diverse persons at many locations, and transmittable from one generation to another. Large knowledge collections available via new media such as the World Wide Web and the Internet have recently become searchable and on-demand retrievable, an advantage long enjoyed by printed resources only. Introducing these new media and the appropriate mechanisms to search and retrieve available multimedia (textual, audio, and visual) information into classrooms can enable teachers and students to apply them in the course of their work. Although "surfing the Internet" is not likely to become the main technique in helping students master existing components of the curriculum, it will become an important element in education. The stock of human knowledge is rapidly going online. Developing skills in finding appropriate information and knowledge resources has always been an important curriculum goal for students going to higher education. Hence, since the stock of human knowledge increasingly becomes available online to any person, at any place, at any time, developing the ability to select and retrieve those resources most relevant to one's purposes becomes an important educational objective generally.

Furthermore, educators seem to face a difficult but common dilemma. On the one hand, they are expected to make education intellectually demanding. This goal is usually accompanied by rigid educational projects, which alienate many students. On the other hand, educators also want to engage each student so that he or she can personally relate to what is being learned. However, "fetching", free-choice projects that are often developed for this purpose, turn out to be superficial and dubious in intellectual value. If students can be given a standardized plan for realizing an educational project but can be allowed to build personalized versions of the pertinent project from the wealth of materials available on the Web, this approach may prove to be a pedagogy that brings intellectual rigor while being engaging for the student. This ideal synthesis of an objectivist and a constructivist approach to teaching has recently been proposed in the literature [14]–[16], and the authors attempted to use it in practice while defining the programming assignments of the introductory AI course in question.

Two main programming assignments constitute the introductory AI course in question. The first one delves into the issues of how to incorporate rule-based reasoning into an intelligent agent and then to use it for constructive purposes—for example, to rank available information according to a set of stipulated preferences. The second exercise focuses on constructing intelligent agents using the semantic network concepts and deploying them to monitor, filter, and retrieve relevant information from the Web. In both cases, the subject problems have been defined so that the students are incited to program a variety of agents which, in collaboration, achieve the given goal.

To alleviate "hard-core" programming tasks and to free students' time for efforts in understanding and exploring AI

concepts and applications, chunks of code were handed out. Namely, the SAF contains a couple of example agents (i.e., templates) that students can use as a starting point for building their own agents (e.g., they can edit, enlarge, or enhance the existing code). This capability also enables students to explore the framework with running agents before they attempt to design agents themselves. As already mentioned previously, the authors implemented rule-based inference procedures, semantic network constructs, and a few search methods in Java. The related Java classes were made available to students.

Fostering teamwork skills and team spirit is an important school curriculum goal. Hence, all the existing project-oriented components of the Computer Science curriculum have been designed for teams of students. In the case of the AI course in question, the assignments have been designed for teams of five or six students.

B. The First Assignment: Rule-Based Reasoning

The first assignment 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 SAF 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.

Participating teams of students usually tackled this problem in the following way (Fig. 3).

1) Build a *Reader Agent* that retrieves filled-out questionnaires (so-called survey data) from the *Survey Data* Web page.
2) Build an *Inference Agent* that applies forward chaining to determine the profiles of the fellow students based upon their survey data. Begin by developing a knowledge base containing a set of rules for determining the profile of a student. The utilized questionnaire has been designed so that it allows students to define a large variety of such rules. For example, based upon an affirmative answer to the question "Do you often take the initiative in a project team?" students may define rules for deducing labels such as *leader*, *arrogant*, or *stupid*. By combining different survey data and already deduced labels, students may define rules for deducing labels such as *creative*, *dependable*, *nerd*, etc.
3) Build a *Score Agent* that assigns to each of the fellow students an estimate of his suitability for being a part of the team. For example, being *dependable* can be a desired property of a student and given a score of $+10$, while *know-it-all* can be deemed to be an unwanted aspect and given a score of -5.
4) Build a *Lister Agent* that assembles a ranking list by sorting the scores provided by the *Score Agent*.
5) Provide the required explanations. The main reason that students were asked to explain their choice of the inference procedure was to estimate whether they understood the difference between forward and backward chaining. They were asked to explain the final ranking generated by their system to incite them to think about their roles in the team. For example, if a member of a particular team was estimated to be unsuitable for the pertinent team, then either the subject member did not participate in the development of the utilized rules and *Score Agent*, or he was not able to defend his position on how the mechanisms in question are to be developed.

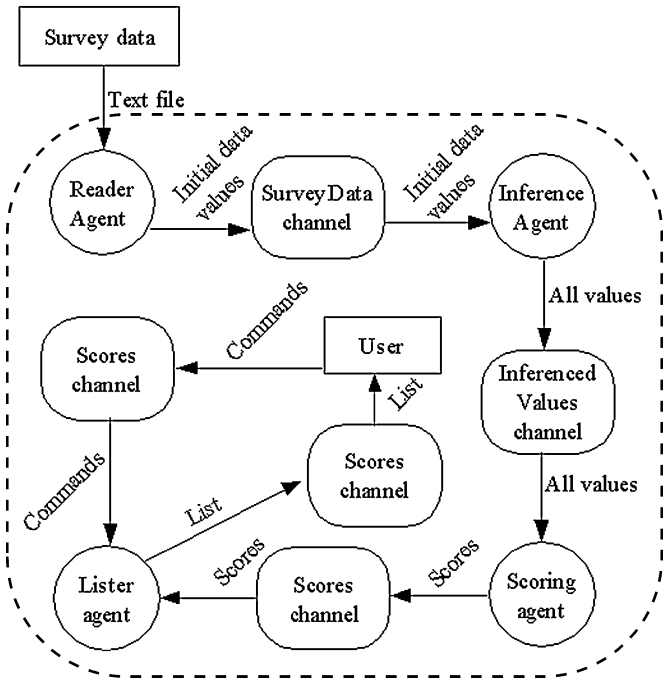

Fig. 3. General structure of the multiagent system to be developed for the first assignment.

C. The Second Assignment: Semantic Networks

The second assignment is as follows.

> Create an agent-based system that retrieves and analyzes the BBC news available via the Internet according to your own preferences. Use SAF to build the required system and employ the semantic network concepts. Explain the drawbacks (if any) of the used approach.

Participating teams of students usually tackled this problem in the following way (Fig. 4).

1) Build a *Reader Agent* that monitors the BBC Web site and flags the system when a fresh news article is posted.
2) Build a *Make Network Agent* that constructs a semantic network representation for each article. Label each word as either *trivial* (words like "a," "an," "the," "is," "has," etc.) or *nontrivial* and represent each nontrivial word as a node of the network. Associate a *relevance* property with each node to expound the frequency with which the word in question occurs in the article. Connect the nodes of the network and assign a *match* property to each such link. This *match* property should expound the frequency with which the connection in question occurs within a single sentence.
3) Build an *Analyzer Agent* that selects ± 5 nodes of the semantic network having the highest values associated

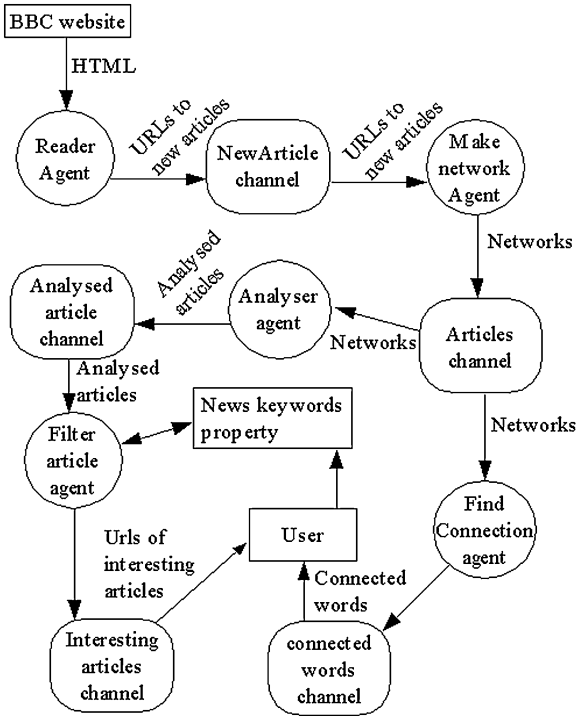

Fig. 4. General structure of the multiagent system to be developed for the second assignment.

with their *relevance* property. Output the selected nodes, their links to other nodes, and the pertinent linked nodes themselves.

4) Build a *Find Connection Agent* that monitors the generated semantic networks and outputs the series of usually connected words (e.g., "New York", "World Trade Center", "Tony Blair", etc.). Use values assigned to the *match* property of the links.
5) Build a *Filter Article Agent* that labels the examined article as being either important or unimportant. Check whether the word related to a selected node is in the list of keywords (in the list of preferred topics). Check also whether a selected node has a strong link (i.e., is usually connected) with another node and, if so, check whether the word related to that node is in the list of keywords. If either of these checks is in the affirmative, label the analyzed article as being important.
6) Provide the required explanation. Students were asked to list the drawbacks of their system to help to inspire the conclusion that homonyms and synonyms might (and probably will) affect the results generated by their system. Another reason for doing so was to estimate whether they understood the drawbacks of the utilized search algorithm, the advantages that could be achieved by including a "trained" *Find Connection Agent* into the filtering process, etc.

V. Classroom Assessment

Evaluation of the utilized educational methods and materials by students is an element of immense importance in the educational programs of Delft University of Technology. Based upon

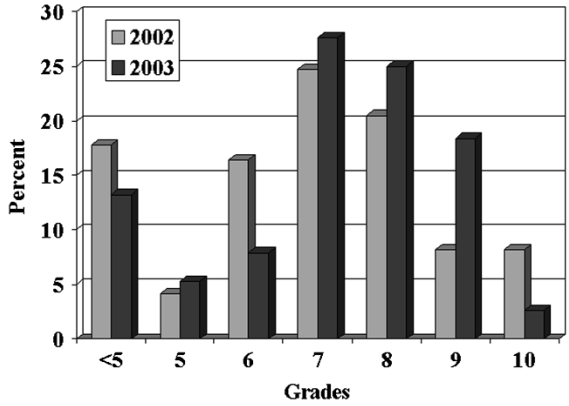

Fig. 5. Grades assigned to the students for the quality of the delivered code. In 2002, from 73 participants, 17.8% did not pass the course (had a grade <5). In 2003, from 76 participants, 13.2% did not pass the course.

the feedback provided by students, the utilized tools and readings can be improved to fit better the knowledge/skills of the current generation of students, their preferences, and the curriculum goals of the educational program in question. A standard questionnaire is available for eliciting students' opinions on computer science courses, including all the relevant questions about the suitability of the used tools and readings, the parts of the course that taught them most, the overall educational experiences of the student, and the ways the course could be enhanced.

In 2002, 73 students participated in the AI course in question. They were divided into 14 teams of five or six students. Sixty students successfully completed the two assignments described previously (Fig. 5). Of 73 participants, 68 filled out the questionnaire for surveying students' educational experience. In 2003, 76 students participated in the course (15 teams). In total, 66 students successfully completed the assignments (Fig. 5), and 72 students filled out the pertinent questionnaire.

In 2002, 67% of the students who filled out the questionnaire indicated that the utilized educational tool is suitable for the goals of the course. They commonly identified two drawbacks of the 2002 version of the SAF [13]. One concerned the outdated look and feel of the GUI (composed only of a rigid list of channels and a list of active agents). The other referenced that whenever an ill-coded agent entered a loop, the framework became unresponsive. As already mentioned previously, the current version of the SAF addresses both limitations of the 2002 version. Hence, in 2003, 89% of the surveyed students indicated that the utilized educational tool is suitable for the goals of the course.

The students judged the two assignments described previously as being interesting and motivating (2002, 87%; 2003, 94%). The students claimed that they enhanced their skills and acquired new, valuable, and applicable knowledge on the instructed subjects (2002, 83%; 2003, 93%). They indicated that most of the learning occurred with the design and implementation of the two multiagent systems (2002, 81%; 2003, 88%). Explaining the choice of a particular AI technique and the drawbacks of the realized multiagent system was instructive for the students (2002, 71%; 2003, 74%). Overall, the students

were happy with the results of their work in terms of the performance and usefulness of the developed multiagent systems (2002, 86%; 2003, 89%).

These results indicate that the students found the developed educational tool (SAF) and the specified programming assignments motivating and highly suitable for the purposes of teaching the basics of AI. In addition, in comparison to another introductory AI course for all computer science students of Delft University of Technology (the course IN 2420, Table I), which uses a traditional objectivist approach, students reported that they learned significantly more from the AI course presented in this paper (the course IN 1851, Table I). In 2003, out of 124 participants in the introductory AI course IN 2420, 83 students filled out the questionnaire surveying students' educational experience. Of those, only 43% claimed that they had enhanced their skills and acquired new, valuable, and applicable knowledge on the instructed subjects. This finding indicates the students' perception of learning more effectively when the traditional objectivist approach is combined with a constructivist approach than when the objectivist approach to teaching programming to novices is used alone.

However, as remarked by Schell [17], although students may frequently report a perception of learning more than in traditional courses, quantitative measures do not always confirm this perception. Therefore, in order to evaluate objectively the educational benefit of the AI programming experience obtained during the practical part of the AI course in question, the authors performed an assessment. They compared the performance of students' answering the questions about rule-based reasoning and semantic networks and the performance of students' answering the other four questions (search algorithms, predicate logic, knowledge acquisition, and distributed AI) on the test. The grade obtained for each question was compared with those obtained for the other questions by computing the ratio between the average score achieved by the students to the maximum score for each question. The semantic networks question rated the highest, 0.89, with the scores for the other questions equal to 0.69 (rule-based reasoning question and the distributed AI question), 0.59 (search algorithms question), 0.45 (predicate logic question), and 0.38 (knowledge acquisition question). These results indicate that the specified programming assignments improved the students' ability to apply the pertinent ideas to novel problems. They also suggest that the course material is highly suitable for the purposes of teaching students the basics of AI, including semantic network concepts, rule-based reasoning, and the intelligent agents paradigm.

VI. Summary

This paper describes a flexible method of teaching introductory artificial intelligence (AI) using a novel Java-implemented simple agent framework. The simplicity of the proposed agent framework, a requisite quality for a tool aimed at teaching programming to novices, has in its possession only those functionalities that are specified by the intended educational purposes and no other. The simple agent framework (SAF) supports the development of intelligent agent applications aimed at monitoring, filtering, and retrieving relevant information from the World Wide Web and embodies the concepts of concurrency, multiagency, and persistency. It does not possess an object-oriented database or a mechanism for performing remote procedure calls since these provisions are not necessary for the development of the intelligent agent applications in question, and they would make the tool unnecessarily complex. In contrast to the existing agent frameworks developed elsewhere (Table II), and as indicated by classroom experience, SAF is a suitable tool for teaching AI programming to novices for the following reasons.

- It is simple; it is accompanied by readable documentation; and it provides readily available, useful examples (i.e., built-in agent templates written in Java which can be edited and to which "intelligence" can be added).
- It embodies simple, self-expained, visual interaction with the user that may concern one, more, or all currently running agents (Fig. 1).

Based on classroom experience, and in comparison to the commonly applied approaches to teaching introductory AI (Table I), 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. It departs from commonly adopted objectivism to avoid an approach that treats learning as simple acquisition of facts, where the newly obtained knowledge and skills are not transferred to previously unseen, real-world problems [6]. It does not adopt a fully constructivist approach either, in which students are given a vaguely specified problem that they should refine and then solve, since this approach has been proven unsuitable for teaching programming to freshmen [7], [16]. Rather, it blends the two approaches: the students are presented with well-defined assignments (objectivist approach) aimed at monitoring, filtering, and retrieving relevant information from the World Wide Web (constructivist approach). Classroom experience indicates that this "hybrid" teaching method significantly increases the extent of learning compared with use of only the objectivist approach to teaching programming to novices.
- The implemented pedagogy has intellectual rigor while being engaging for students. The students are given a standardized plan for realizing the assignments, but they build personalized versions of the intended multiagent systems according to their preferences from the large stock of information available on the Web.
- The implemented pedagogy represents a method of teaching intelligent, multiagent, system development that provides the student with a deeper understanding of the process in a real-world-oriented (Web-based), cooperative (team-based) setting.

These results strongly indicate that the implemented pedagogy and the utilized newly developed educational tool presented in this paper form a highly effective way of teaching introductory AI. In turn, the SAF system and the specified AI programming projects can be widely useful for many different AI courses, including courses for noncomputer-science majors.

Finally, the SAF system provides a basis for the development of other courses having intelligent agents programmed to perform specific tasks with the goal of solving more complex problems. Robotics and computer games are examples of such courses. The authors are currently engaged in the development of the practical part of a machine learning course as a set of assignments that requires the students to use SAF intelligent agents and machine learning techniques to do the following:

- to detect the shown facial expression in an input video of the face;
- to interpret this information in terms of emotions;
- to retrieve a video from a remote database of face images picturing the same emotion displayed by a face being the most similar to the face in the input video.

Once this machine learning course is developed, one could assess the usefulness of SAF in contexts other than the one discussed in this paper.

ACKNOWLEDGMENT

The authors would like to thank the 68 MKT-undergraduate students who participated in the course described in this paper in 2002 and provided the authors with numerous useful comments on how to enhance the preliminary version of the SAF. The authors would like to thank all 139 MKT undergraduates who evaluated the course in 2002 and 2003, as well as M. J. Nieman and the anonymous reviewers for their helpful comments and suggestions.

REFERENCES

[1] Dutch Statistical Bureau. [Online]. Available: http://statline.cbs.nl/StatWeb/table.asp?PA=70655ned&D1=11-71, 76-90, 120, 123, 146&D2=0 &D3=(l-11)-l&DM=SLNL&LA=nl&TT=2

[2] S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*, 2nd ed. Upper Saddle River, NJ: Pearson Education, 2003.

[3] D. Poole, A. Macworth, and R. Goebel, *Computational Intelligence a Logical Approach*. Oxford, U.K.: Oxford Press, 1998.

[4] M. Negnevitsky, *Artificial Intelligence*. Harlow, London, U.K.: Pearson Education Limited, 2002.

[5] J. P. Bigus and J. Bigus, *Constructing Intelligent Agents Using Java*. New York: Wiley, 2001.

[6] J. Biggs, *Teaching for Quality Learning at University*. Buckingham, U.K.: Open University Press, 1999.

[7] M. McCracken, V. Almstrum, D. Diaz, M. Guzdial, D. Hagan, Y. B.-D. Kolikant, C. Laxer, L. Thomas, and I. Utting, "Multi-national, multiinstitutional study of assessment of programming skills of first-year CS students," in *SIGCSE Bull.*, vol. 33, Apr. 2001, pp. 1–16.

[8] AgentLink. Agents in the Press—European co-ordination action for agent-based computing. [Online]. Available: http://www.agentlink.org/press/

[9] AgentLink. Agent Software—European co-ordination action for agent–based computing. [Online]. Available: http://www.agentlink.org/resources/agent-software.php

[10] FIPA–OS Agent Toolkit. [Online]. Available: http://www.emorphia.com/research/about.htm

[11] Fujitsu Laboratories, Ltd. Pathwalker—Agent-oriented programming library. [Online]. Available: http://www.labs.fujitsu.com/en/freesoft/paw/

[12] IEEE Distributed Systems Online—Distributed Agents Projects. [Online]. Available: http://dsonline.computer.org/agents/projects.htm

[13] M. Pantic, R. Zwitserloot, and R. J. Grootjans, "Simple agent framework," in *IEEE Int. Conf. Information Technology in Research Education*, Newark, NJ, Aug. 2003, pp. 426–430.

[14] J. B. Black and R. McClintock, "An interpretation construction approach to constructivist design," in *Constructivist Learning Environments: Case Studies in Instructional Design*, B. G. Wilson, Ed. Englewood Cliffs, NJ: Educational Technology, 1995, pp. 25–31.

[15] S.-F. Chang, A. Eleftheriadis, and R. McClintock, "Next-generation content representation, creation, and searching for new-media applications in education," *Proc. IEEE*, vol. 86, no. 5, pp. 884–904, May 1998.

[16] R. Lister and J. Leaney, "Bad theory versus bad teachers: Toward a pragmatic synthesis of constructivism and objectivism," in *Int. Conf. Higher Education Research and Development Society of Australasia, Inc.*, Christchurch, New Zealand, Jul. 2003, pp. 429–436.

[17] G. P. Schell, "Universities marginalize online courses," *Commun. ACM*, vol. 47, no. 7, pp. 53–56, Jul. 2004.

Maja Pantic (S'98–M'02) received the M.S. and Ph.D. degrees in computer science from Delft University of Technology, Delft, The Netherlands, in 1997 and 2001, respectively.

She joined the Data and Knowledge Systems Group (currently the Man–Machine Interaction Group) of the Electrical Engineering, Mathematics, and Computer Science Department at Delft University of Technology as an Assistant Professor in 2001. Her research interests pertain to the application of AI and computational intelligence techniques in the analysis of different aspects of human behavior for the realization of perceptual, context-aware, multimodal human–machine interfaces. She is also actively involved in research activities aimed at supporting learning in a more modern, natural, flexible, portable, and on-demand manner.

Reinier Zwitserloot received the B.S. degree in computer science from Delft University of Technology, Delft, The Netherlands, in 2003. He is currently working toward the M.S. degree in computer science at the same university.

His major research interests include multiagent systems, distributed artificial intelligence, and intelligent agent frameworks.

Robbert Jan Grootjans received the B.S. degree in computer science from Delft University of Technology, Delft, The Netherlands, in 2003. He is currently working toward the M.S. degree in computer science at the same university.

His major research interests include multiagent systems and agent framework technology.